Resampling statistics

Julian Lincoln Simon Matthew Brett Stéfan van der Walt

Table of contents

Python version

There are two versions of this book; one with examples in the R programming language[1], and another with examples in the Python language [2].

This is the Python version.

We wrote this book in RMarkdown with Quarto, and configured Github to rebuild the textbook HTML and PDF files from the RMarkdown source.

[1] https://resampling-stats.github.io/edition-3-r
[2] https://resampling-stats.github.io/edition-3-python

Preface to the third edition

The book in your hands, or on your screen, is the third edition of a book originally called "Resampling: the new statistics", by Julian Lincoln Simon (1992).

One of the pleasures of writing a new edition of a work by another author, is that we can praise the previous version of our own book. We will do that, in the next section. Next we talk about the resampling methods in this book, and their place at the heart of "data science". We then discuss what we have changed, what we haven't, and why. Finally, we make some suggestions about where this book could fit into your learning and teaching.

What Simon saw

Simon gives the early history of this book in the original preface. He starts with the following observation:

> In the mid-1960's, I noticed that most graduate students — among them many who had had several advanced courses in statistics — were unable to apply statistical methods correctly…

Simon then applied his striking capacity for independent thought to the problem — and came to two essential conclusions.

The first was that introductory courses in statistics use far too much mathematics. Most students cannot follow along and quickly get lost, reducing the subject to — as Simon puts it — "mumbo-jumbo".

On its own, this was not a new realization. Simon quotes a classic textbook by Wallis and Roberts (1956), to the effect that teaching statistics through mathematics is like teaching philosophy in ancient Greek. More recently, other teachers of statistics have come to the same conclusion. Cobb (2007) argues that it is not practical to teach students the level of mathematics they would need to understand standard introductory courses. As you will see below, Cobb also agrees with Simon about the solution.

Simon's great contribution was to see *how* we can replace the mathematics, to better reveal the true heart of statistical thinking. His starting point appears in the original preface: "Beneath the logic of a statistical inference there necessarily lies a physical process". Drawing conclusions from noisy data means building a *model* of the noisy world, and seeing how that model behaves.

That model can be physical, where we generate the noisiness of the world using physical devices like dice and spinners and coin-tosses. Simon used exactly these kinds of devices in his first experiments in teaching (Simon 1969). He then saw that it was much more efficient to build these models with simple computer code, and the result was the first and second editions of this book, with their associated software, the *Resampling Stats* language.

Simon's second conclusion follows from the first. Now he had found a path round the unnecessary barrier of mathematics, he had got to the heart of what is interesting and difficult in statistics. Drawing conclusions from noisy data involves a lot of hard, clear thinking. We should be honest with our students about that; statistics is hard, not because it is obscure (it need not be), but because it deals with difficult problems. It is exactly that hard logical thinking that can make statistics so interesting to our best students; "statistics" is just reasoning about the world when the world is noisy. Simon writes eloquently about this in a section in the introduction — "Why is statistics such a difficult subject" (Section 1.6).

We need both of Simon's conclusions to make progress. We cannot hope to teach two hard subjects at the same time; mathematics, and statistical reasoning. He replaced the mathematics with something that is much easier for most of us to reason about. By doing that, he can concentrate on the real, interesting problem — the hard thinking about data, and the world it comes from. To quote from a later section in this book (Section 2.4): "Once we get rid of the formulas and tables, we can see that statistics is a matter of *clear thinking, not fancy mathematics*." Instead of asking "where would I look up the right recipe for this?", you find yourself asking "what kind of world do these data come from?" and "How can I reason about that world?". Like Simon, we have found that this way of thinking and teaching brings rich rewards — for insight and practice. We hope and believe that you will find the same.

Resampling and data science

The ideas in Simon's book, first published in 1992, have found themselves at the center of the modern movement of *data science*.

In the section above, we described Simon's path in discovering physical models as a way of teaching and explaining statistical tests. He saw that code was the right way to express these physical models, and therefore, to build and explain statistical tests.

Meanwhile, the wider world of data analysis has been coming to the same conclusion, but from the opposite direction. Simon saw the power of resampling for explanation, and then that code was the right way to express these explanations. The data science movement discovered first that code was essential for data analysis, and then that code was the right way to explain statistics.

The modern use of the phrase "data science" comes from the technology industry. From around 2007, companies such as LinkedIn and Facebook began to notice that there was a new type of data analyst that was much more effective than their predecessors. They came to call these

analysts "data scientists", because they had learned how to deal with large and difficult data while working in scientific fields such as ecology, biology, or astrophysics. They had done this by learning to use code:

> Data scientists' most basic, universal skill is the ability to write code. (Davenport and Patil 2012)

Further reflection (Donoho 2017) suggested that something deep was going on: that *data science* was the expression of a radical change in the way we analyze data, in academia, and in industry. At the center of this change — was code. Code is the language that allows us to tell the computer what it should do with data; it is the native language of data analysis.

This insight transforms the way with think of code. In the past, we have thought of code as a separate, specialized skill, that some of us learn. We take coding courses — we "learn to code". But if we us code as the fundamental language for analyzing data, then we need code to express what data analysis does, and explain how it works. Here we "code to learn". Code is not an aim in itself, but a language we can use to express the simple ideas behind data analysis and statistics.

Thus the data science movement started from code as the foundation for data analysis, to using code to explain statistics. It ends at the same place as this book, from the other side of the problem.

The growth of data science is the inevitable result of taking computing seriously in education and research. We have already cited Cobb (2007) on the impossibility of teaching the mathematics students would need in order to understand traditional statistics courses. He goes on to explain why there is so much mathematics, and why we should remove it. In the age before ubiquitous computing, we needed mathematics to simplify calculations that we could not practically do by hand. Now we have great computing power in our phones and laptops, we do not have this constraint, and we can use simpler ideas from resampling methods to solve the same problems. As Simon shows, these are much easier to describe and understand. Data science, and teaching with resampling, are the obvious consequences of ubiquitous computing.

What we changed

This diversion, through data science, leads us to the changes that we have made for the new edition. The previous edition of this book is still excellent, and you can read it freely at http://www.resample.com/intro-text-online. It continues to be ahead of its time, and ahead of our time. Its one major drawback is that Simon bases much of the book around code written in a special language that he developed with Dan Weidenfeld, called *Resampling Stats*[3]. The Resampling Stats language is well designed for expressing the steps in simulating worlds

[3]If you are interested, https://statistics101.sourceforge.io has a free modern version of the original Resampling Stats language.

that include elements of randomness, and it was a useful contribution at the time that it was written. Since then, and particularly in the last decade, there have been many improvements in more powerful and general languages, such as Python and R. These languages are particularly suitable for beginners in data analysis, and they come with a huge range of tools and libraries for a many tasks in data analysis, including the kinds of models and simulations you will see in this book. We have updated the book to use Python, instead of *Resampling Stats*. If you already know Python or a similar language, such as R, you will have a big head start in reading this book, but even if you do not, we have written the book so it will be possible to pick up the Python code that you need to understand and build the kind of models that Simon uses. The advantage to us, your authors, is that we can use the very powerful tools associated with Python to make it easier to run and explain the code. The advantage to you, our readers, is that you can also learn these tools, and the Python language. They will serve you well for the rest of your career in data analysis.

Our second and minor change is that we have added some content that Simon specifically left out. Simon knew that his approach was radical for its time, and designed his book as a commentary, correction, and addition to traditional courses in statistics. He assumes some familiarity with the older world of normal distributions, standard deviations, and correlation. We want this book to useful to the true beginner, so we have added some explanation of standard deviation, standard scores and the correlation coefficient. We have also updated some of the examples.

In this third edition, we have deliberately been light in our edits, to preserve the fresh and creative flavor of Simon's book, as he worked through the landscape of traditional statistics with his radical eye.

The third edition is the director's cut, where Simon is the director

As you see from the section above, the largest change for this edition is to update the code sections to use Python. We intend this edition to be as close as possible to the book that Simon intended, but updated to use modern tools and a standard, widely-used programming language. Read this edition as our service to Simon for his visionary work — this is Simon's book, and, for this edition, we (Matthew and Stéfan) intend to serve as his editors and interpreters. We release this edition so you can see Simon's ideas updated to current technology.

Who should read this book, and when

As you have seen in the previous sections, this book uses a radical approach to explaining *statistical inference* — the science of drawing conclusions from noisy data. This approach is quickly becoming the standard in teaching of data science, partly because it is so much easier to explain, and partly because of the increasing role of code in data analysis.

Our book teaches the basics of using the Python language, basic probability, statistical inference through simulation and resampling, confidence intervals, and basic Bayesian reasoning, all through the use of model building in simple code.

Statistical inference is an important part of research methods for many subjects; so much so, that research methods courses may even be called "statistics" courses, or include "statistics" components. This book covers the basic ideas behind statistical inference, and how you can apply these ideas to draw practical statistical conclusions. We recommend it to you as an introduction to statistics. If you are a teacher, we suggest you consider this book as a primary text for first statistics courses. We hope you will find, as we have, that this method of explaining through building is much more productive and satisfying than the traditional method of trying to convey some "intuitive" understanding of fairly complicated mathematics. We hope you will see the relationship of these resampling techniques to traditional methods. Even if you do need to teach your students t-tests, and analysis of variance, we hope you will share our experience that this way of explaining the underlying ideas is much more compelling than the traditional approach.

Simon wrote this book for students and teachers who were interested to discover a radical new method of explanation in statistics and probability. The book will still work well for that purpose. If you have done a statistics course, but you kept feeling that you did not really understand it, or there was something fundamental missing that you could not put your finger on — well done for sensing the problem! — then, please, read this book. There is a good chance that it will give you deeper understanding, and reveal the logic behind the often arcane formulations of traditional statistics.

Our book is only part of a data science course. There are several important aspects to data science. A data science course needs all the elements we list above, but it should also cover the process of reading, cleaning, and reorganizing data using Python, or another language, such as R. It may also discuss the problems of experimental design, and cover prediction techniques, such as classification with machine learning, as well as data exploration with plots, tables, and summary measures. We do not cover those here. If you are teaching a full data science course, we suggest that you use this book as your first text, as an introduction to code, and statistical inference, and then add some of the many excellent resources on these other aspects of data science that assume some knowledge of statistics and programming.

The book as a public resource

Simon was passionate about this approach to teaching, as you can see from his preface to the second edition, and from his generosity in publishing the second edition on the web. We feel the same way, and we have released the third edition in the same way. Technology and technical culture have evolved since the second edition, and we can now give you the tools to edit and improve this book, for the benefit of all our readers. If you see an error in the book,

or you have thought of a better way of explaining something, please send us a fix or an edit. See Appendix E for the procedure, and accept our thanks in advance.

Welcome to resampling

We hope you will agree that Simon's insights for understanding and explaining are — really extraordinary. We are catching up slowly. If you are like us, your humble authors, you will find that Simon has succeeded in explaining what statistics is, and *exactly* how it works, to anyone with the patience to work through the examples, and think hard about the problems. If you have that patience, the rewards are great. Not only will you understand statistics down to its deepest foundations, but you will be able to think of your own tests, for your own problems, and have the tools to implement them yourself.

Matthew Brett

Stéfan van der Walt

Preface to the second edition

> **Note**
>
> This is a slightly edited version of the original preface to the second edition. We removed an introduction to the original custom software, and a look ahead at the original contents of the book.

Brief history of the resampling method

This book describes a revolutionary — but now fully accepted — approach to probability and statistics. Monte Carlo resampling simulation takes the mumbo-jumbo out of statistics and enables even beginning students to understand completely everything that is done.

Before we go further, let's make the discussion more concrete with an example. Ask a class: What are the chances that three of a family's first four children will be girls? After various entertaining class suggestions about procreating four babies, or surveying families with four children, someone in the group always suggests flipping a coin. This leads to valuable student discussion about whether the probability of a girl is exactly half (there are about 105 males born for each 100 females), whether .5 is a satisfactory approximation, whether four coins flipped once give the same answer as one coin flipped four times, and so on. Soon the class decides to take actual samples of coin flips. And students see that this method quickly arrives at estimates that are accurate enough for most purposes. Discussion of what is "accurate enough" also comes up, and that discussion is valuable, too.

The Monte Carlo method itself is not new. Near the end of World War II, a group of physicists at the Rand Corp. began to use random-number simulations to study processes too complex to handle with formulas. The name "Monte Carlo" came from the analogy to the gambling houses on the French Riviera. The application of Monte Carlo methods in *teaching* statistics also is not new. Simulations have often been used to illustrate basic concepts. What *is* new and radical is using Monte Carlo methods routinely as problem-solving tools for everyday problems in probability and statistics.

From here on, the related term *resampling* will be used throughout the book. Resampling refers to the use of the observed data or of a data generating mechanism (such as a die)

to produce new hypothetical samples, the results of which can then be analyzed. The term *computer-intensive methods* also is frequently used to refer to techniques such as these.

The history of resampling is as follows: In the mid-1960's, I noticed that most graduate students — among them many who had had several advanced courses in statistics — were unable to apply statistical methods correctly in their social science research. I sympathized with them. Even many experts are unable to understand intuitively the formal mathematical approach to the subject. Clearly, we need a method free of the formulas that bewilder almost everyone.

The solution is as follows: Beneath the logic of a statistical inference there necessarily lies a physical process. The resampling methods described in this book allow us to work directly with the underlying physical model by simulating it, rather than describing it with formulae. This general insight is also the heart of the specific technique Bradley Efron felicitously labeled 'the bootstrap' (1979), a device I introduced in 1969 that is now the most commonly used, and best known, resampling method.

The resampling approach was first tried with graduate students in 1966, and it worked exceedingly well. Next, under the auspices of the father of the "new math," Max Beberman, I "taught" the method to a class of high school seniors in 1967. The word "taught" is in quotation marks because the pedagogical essence of the resampling approach is that the students discover the method for themselves with a minimum of explicit instruction from the teacher.

The first classes were a success and the results were published in 1969 (Simon and Holmes 1969). Three PhD experiments were then conducted under Kenneth Travers' supervision, and they all showed overwhelming superiority for the resampling method (Simon, Atkinson, and Shevokas 1976). Subsequent research has confirmed this success.

The method was first presented at some length in the 1969 edition of my book *Basic Research Methods in Social Science* (Simon 1969) (third edition with Paul Burstein (1985)).

For some years, the resampling method failed to ignite interest among statisticians. While many factors (including the accumulated intellectual and emotional investment in existing methods) impede the adoption of any new technique, the lack of readily available computing power and tools was an obstacle. (The advent of the personal computer in the 1980s changed that, of course.)

Then in the late 1970s, Efron began to publish formal analyses of the bootstrap — an important resampling application (Efron 1979). Interest among statisticians has exploded since then, in conjunction with the availability of easy, fast, and inexpensive computer simulations. The bootstrap has been the most widely used, but across-the-board application of computer intensive methods now seems at hand. As Noreen (1989) noted, "there is a computer-intensive alternative to just about every conventional parametric and non-parametric test." And the bootstrap method has now been hailed by an official *American Statistical Association* volume as the only "great breakthrough" in statistics since 1970 (Kotz and Johnson 1992).

It seems appropriate now to offer the resampling method as the technique of choice for beginning students as well as for the advanced practitioners who have been exploring and applying the method.

Though the term "computer-intensive methods" is nowadays used to describe the techniques elaborated here, this book can be read either with or without the accompanying use of the computer. However, as a practical matter, users of these methods are unlikely to be content with manual simulations if a quick and simple computer-program alternative is available.

The ultimate test of the resampling method is how well you, the reader, learn it and like it. But knowing about the experiences of others may help beginners as well as experienced statisticians approach the scary subject of statistics with a good attitude. Students as early as junior high school, taught by a variety of instructors and in other languages as well as English, have — in a matter of 6 or 12 short hours — learned how to handle problems that students taught conventionally do not learn until advanced university courses. And several controlled experimental studies show that, on average, students who learn this method are more likely to arrive at correct solutions than are students who are taught conventional methods.

Best of all, the experiments comparing the resampling method against conventional methods show that students *enjoy* learning statistics and probability this way, and they don't suffer statistics panic. This experience contrasts sharply with the reactions of students learning by conventional methods. (This is true even when the same teachers teach both methods as part of an experiment.)

A public offer: The intellectual history of probability and statistics began with gambling games and betting. Therefore, perhaps a lighthearted but very serious offer would not seem inappropriate here: I hereby publicly offer to stake $5,000 in a contest against any teacher of conventional statistics, with the winner to be decided by whose students get the larger number of simple and complex numerical problems correct, when teaching similar groups of students for a limited number of class hours — say, six or ten. And if I should win, as I am confident that I will, I will contribute the winnings to the effort to promulgate this teaching method. (Here it should be noted that I am far from being the world's most skillful or charming teacher. It is the subject matter that does the job, not the teacher's excellence.) This offer has been in print for many years now, but no one has accepted it.

The early chapters of the book contain considerable discussion of the resampling method, and of ways to teach it. This material is intended mainly for the instructor; because the method is new and revolutionary, many instructors appreciate this guidance. But this didactic material is also intended to help the student get actively involved in the learning process rather than just sitting like a baby bird with its beak open waiting for the mother bird to drop morsels into its mouth. You may skip this didactic material, of course, and I hope that it does not get in your way. But all things considered, I decided it was better to include this material early on rather than to put it in the back or in a separate publication where it might be overlooked.

Brief history of statistics

In ancient times, mathematics developed from the needs of governments and rich men to number armies, flocks, and especially to count the taxpayers and their possessions. Up until the beginning of the 20th century, the term *statistic* meant the number of something — soldiers, births, taxes, or what-have-you. In many cases, the term *statistic* still means the number of something; the most important statistics for the United States are in the *Statistical Abstract of the United States*. These numbers are now known as descriptive statistics. This book will not deal at all with the making or interpretation of descriptive statistics, because the topic is handled very well in most conventional statistics texts.

Another stream of thought entered the field of probability and statistics in the 17th century by way of gambling in France. Throughout history people had learned about the odds in gambling games by repeated plays of the game. But in the year 1654, the French nobleman Chevalier de Méré asked the great mathematician and philosopher Pascal to help him develop correct odds for some gambling games[4]. Pascal, the famous Fermat, and others went on to develop modern probability theory.

Later these two streams of thought came together. Researchers wanted to know how accurate their descriptive statistics were — not only the descriptive statistics originating from sample surveys, but also the numbers arising from experiments. Statisticians began to apply the theory of probability to the accuracy of the data arising from sample surveys and experiments, and that became the theory of *inferential statistics*.

Here we find a guidepost: probability theory and statistics are relevant whenever there is uncertainty about events occurring in the world, or in the numbers describing those events.

Later, probability theory was also applied to another context in which there is uncertainty — decision-making situations. Descriptive statistics like those gathered by insurance companies — for example, the number of people per thousand in each age bracket who die in a five-year period — have been used for a long time in making decisions such as how much to charge for insurance policies. But in the modern probabilistic theory of decision-making in business, politics and war, the emphasis is different; in such situations the emphasis is on methods of *combining* estimates of probabilities that depend upon each other in complicated ways in order to arrive at the best decision. This is a return to the gambling origins of probability and statistics. In contrast, in standard insurance situations (not including war insurance or insurance on a dancer's legs) the probabilities can be estimated with good precision without complex calculation, on the basis of a great many observations, and the main statistical task is gathering the information. In business and political decision-making situations, however, one often works with probabilities based on very limited information — often little better than guesses. There the task is how best to combine these guesses about various probabilities into an overall probability estimate.

[4] https://en.wikipedia.org/wiki/Problem_of_points

Estimating probabilities with conventional mathematical methods is often so complex that the process scares many people. And properly so, because its difficulty leads to errors. The statistics profession worries greatly about the widespread use of conventional tests whose foundations are poorly understood. The wide availability of statistical computer packages that can easily perform these tests with a single command, regardless of whether the user understands what is going on or whether the test is appropriate, has exacerbated this problem. This led John Tukey to turn the field toward descriptive statistics with his techniques of "exploratory data analysis" (Tukey 1977). These descriptive methods are well described in many texts.

Probabilistic analysis also is crucial, however. Judgments about whether the government should allow a new medicine on the market, or whether an operator should adjust a screw machine, require more than eyeball inspection of data to assess the chance variability. But until now the teaching of probabilistic statistics, with its abstruse structure of mathematical formulas, mysterious tables of calculations, and restrictive assumptions concerning data distributions — all of which separate the student from the actual data or physical process under consideration — have been an insurmountable obstacle to intuitive understanding.

Now, however, the resampling method enables researchers and decision-makers in all walks of life to obtain the benefits of statistics and predictability without the shortcomings of conventional methods, free of mathematical formulas and restrictive assumptions. Resampling's repeated experimental trials on the computer enable the data (or a data-generating mechanism representing a hypothesis) to express their own properties, without difficult and misleading assumptions.

So — good luck. I hope that you enjoy the book and profit from it.

Julian Lincoln Simon

1997

1 Introduction

1.1 Uses of Probability and Statistics

This chapter introduces you to probability and statistics. First come examples of the kinds of practical problems that this knowledge can solve for us. One reason that the term "statistic" often scares and confuses people is that the term has several sorts of meanings. We discuss the meanings of "statistics" in the section "Types of statistics". Then comes a discussion on the relationship of probabilities to decisions. Following this we talk about the limitations of probability and statistics. And last is a discussion of why statistics can be such a difficult subject. Most important, this chapter describes the types of problems the book will tackle.

At the foundation of sound decision-making lies the ability to make accurate estimates of the probabilities of future events. Probabilistic problems confront everyone — a company owner considering whether to expand their business, to the scientist testing a vaccine, to the individual deciding whether to buy insurance.

1.2 What kinds of problems shall we solve?

These are some examples of the kinds of problems that we can handle with the methods described in this book:

1. *You are a doctor* trying to develop a treatment for Covid. Currently you are working on a medicine labeled AntiAnyVir. You have data from patients to whom medicine AntiAnyVir was given. You want to judge on the basis of those results whether AntiAnyVir really improves survival or whether it is no better than a sugar pill.
2. *You are the campaign manager* for the Republicrat candidate for President of the United States. You have the results from a recent poll taken in New Hampshire. You want to know the chance that your candidate would win in New Hampshire if the election were held today.
3. *You are the manager and part owner* of one of several contractors providing ambulances to a hospital. You own 16 ambulances. Based on past experience, the chance that any one ambulance will be unfit for service on any given day is about one in ten. You want to know the chance on a particular day — tomorrow — that three or more of them will be out of action.

4. *You are an environmental scientist* monitoring levels of phosphorus pollution in a lake. The phosphorus levels have been fluctuated around a relatively low level until recently, but they have been higher in the last few years. Do these recent higher levels indicate some important change or can we put them down to ordinary variation from year to year?

The core of all these problems, and of the others that we will deal with in this book, is that you want to know the "chance" or "probability" — different words for the same idea — that some event will or will not happen, or that something is true or false. To put it another way, we want to answer questions about "What is the probability that…?", given the body of information that you have in hand.

The question "What is the probability that…?" is usually not the ultimate question that interests us at a given moment.

Eventually, a person wants to use the estimated probability to help make a *decision* concerning some action one might take. These are the kinds of decisions, related to the questions about probability stated above, that ultimately we would like to make:

1. *Should you (the researcher) advise doctors to prescribe medicine AntiAnyVir* for Covid patients, or, should you (the researcher) continue to study AntiAnyVir before releasing it for use? A related matter: should you and other research workers feel sufficiently encouraged by the results of medicine AntiAnyVir so that you should continue research in this general direction rather than turning to some other promising line of research? These are just two of the possible decisions that might be influenced by the answer to the question about the probability that medicine AntiAnyVir is effective in treating Covid.
2. *Should you advise the Republicrat presidential candidate to go to New Hampshire* to campaign? If the poll tells you conclusively that she or he will not win in New Hampshire, you might decide that it is not worthwhile investing effort to campaign there. Similarly, if the poll tells you conclusively that they surely will win in New Hampshire, you probably would not want to campaign further there. But if the poll is not conclusive in one direction or the other, you might choose to invest the effort to campaign in New Hampshire. Analysis of the chances of winning in New Hampshire based on the poll data can help you make this decision sensibly.
3. *Should your company buy more ambulances*? Clearly the answer to this question is affected by the probability that a given number of your ambulances will be out of action on a given day. But of course this estimated probability will be only one part of the decision.
4. *Should we search for new causes of phosphorus pollution* as a result of the recent measurements from the lake? If the causes have not changed, and the recent higher values were just the result of ordinary variation, our search will end up wasting time and money that could have been better spent elsewhere.

The kinds of questions to which we wish to find probabilistic and statistical answers may be found throughout the social, biological and physical sciences; in business; in politics; in engineering; and in most other forms of human endeavor.

1.3 Types of statistics

The term *statistics* sometimes causes confusion and therefore needs explanation.

Statistics can mean two related things. It can refer to a certain sort of *number* — of which more below. Or it can refer to the field of inquiry that studies these numbers.

A *statistic* is a *number* that we can calculate from a larger collection of numbers we are interested in. For example, table Table 1.1 has some yearly measures of "soluble reactive phosphorus" (SRP) from Lough Erne — a lake in Ireland (Zhou, Gibson, and Foy 2000).

Table 1.1: Soluble Reactive Phosphorus in Lough Erne

Year	SRP
1974	26.2
1975	22.8
1976	37.2
1983	54.7
1984	37.7
1987	54.3
1989	35.7
1991	72.0
1992	85.1
1993	86.7
1994	93.3
1995	107.2
1996	80.3
1997	70.7

We may want to *summarize* this set of SRP measurements. For example, we could add up all the SRP values to give the *total*. We could also divide the total by the number of measurements, to give the *average*. Or we could measure the spread of the values by finding the *minimum* and the *maximum* — see table Table 1.2). All these numbers are *descriptive statistics*, because they are numbers that summarize and therefore *describe* the collection of SRP measurements.

Table 1.2: Statistics for SRP levels

	Descriptive statistics for SRP
Total	863.9
Mean	61.7
Minimum	22.8
Maximum	107.2

Descriptive statistics are nothing new to you; you have been using many of them all your life.

We can calculate other numbers that can be useful for drawing *conclusions* or *inferences* from a collection of numbers; these are *inferential statistics*. Inferential statistics are often probability values that give the answer to questions like "What are the chances that …".

For example, imagine we suspect there was some environmental change in 1990. We see that the average SRP value before 1990 was 38.4 and the average SRP value after 1990 was 85. That gives us a difference in the average of 46.6. But, could this difference be due to chance fluctuations from year to year? Were we just unlucky in getting a few larger measurements in later years? We could use methods that you will see in this book to calculate a *probability* to answer that question. The probability value is an *inferential* statistic, because we can use it to draw an *inference* about the measures.

Inferential statistics use descriptive statistics as their input. Inferential statistics can be used for two purposes: to aid scientific *understanding* by estimating the probability that a statement is true or not, and to aid in making *sound decisions* by estimating which alternative among a range of possibilities is most desirable.

1.4 Probabilities and decisions

There are two differences between questions about probabilities and the ultimate decision problems:

1. Decision problems always involve *evaluation of the consequences* — that is, taking into account the benefits and the costs of the consequences — whereas pure questions about probabilities are estimated without evaluations of the consequences.

2. Decision problems often involve a *complex combination* of *sets of probabilities and consequences*, together with their evaluations. For example: In the case of the contractor's ambulances, it is clear that there will be a monetary loss to the contractor if she makes a commitment to have 14 ambulances available for tomorrow and then cannot produce that many. Furthermore, the contractor must take into account the further consequence

that there *may* be a loss of goodwill for the future if she fails to meet her obligations tomorrow — and then again there *may not* be any such loss; and if there is such loss of goodwill it might be a loss worth $10,000 *or* $20,000 *or* $30,000. Here the decision problem involves not only the probability that there will be fewer than 14 ambulances tomorrow but also the immediate monetary loss and the subsequent possible losses of goodwill, and the valuation of all these consequences.

Continuing with the decision concerning whether to do more research on medicine AntiAnyVir: If you do decide to continue research on AntiAnyVir, (a) you may, or (b) you may not, come up with an important general treatment for viral infections within, say, the next 3 years. If you do come up with such a general treatment, of course it will have very great social benefits. Furthermore, (c) if you decide not to do further research on AntiAnyVir now, you can direct your time and that of other people to research in other directions, with some chance that the other research will produce a less-general but nevertheless useful treatment for some relatively infrequent viral infections. Those three possibilities have different social benefits. The probability that medicine AntiAnyVir really has some benefit in treating Covid, as judged by your prior research, obviously will influence your decision on whether or not to do more research on medicine AntiAnyVir. But that judgment about the probability is only one part of the overall web of consequences and evaluations that must be taken into account when making your decision whether or not to do further research on medicine AntiAnyVir.

Why does this book limit itself to the specific probability questions when ultimately we are interested in decisions? A first reason is division of labor. The more general aspects of the decision-making process in the face of uncertainty are treated well in other books. This book's special contribution is its new approach to the crucial process of estimating the chances that an event will occur.

Second, the specific elements of the overall decision-making process taught in this book belong to the interrelated subjects of *probability theory* and *statistics*. Though probabilistic and statistical theory ultimately is intended to be part of the general decision-making process, often only the estimation of probabilities is done systematically, and the rest of the decision-making process — for example, the decision whether or not to proceed with further research on medicine AntiAnyVir — is done in informal and unsystematic fashion. This is regrettable, but the fact that this is standard practice is an additional reason why the treatment of statistics and probability in this book is sufficiently complete.

A third reason that this book covers only statistics and not numerical reasoning about decisions is because most college and university statistics courses and books are limited to statistics.

1.5 Limitations of probability and statistics

Statistical testing is not equivalent to research, and research is not the same as statistical testing. Rather, statistical inference is a handmaiden of research, often but not always necessary

in the research process.

A working knowledge of the basic ideas of statistics, especially the elements of probability, is unsurpassed in its general value to everyone in a modern society. Statistics and probability help clarify one's thinking and improve one's capacity to deal with practical problems and to understand the world. To be efficient, a social scientist or decision-maker is almost certain to need statistics and probability.

On the other hand, important research and top-notch decision-making have been done by people with absolutely no formal knowledge of statistics. And a limited study of statistics sometimes befuddles students into thinking that statistical principles are guides to research design and analysis. This mistaken belief only inhibits the exercise of sound research thinking. Alfred Kinsey long ago put it this way:

> … no statistical treatment can put validity into generalizations which are based on data that were not reasonably accurate and complete to begin with. It is unfortunate that academic departments so often offer courses on the statistical manipulation of [data from human behavior] to students who have little understanding of the problems involved in securing the original data. … When training in these things replaces or at least precedes some of the college courses on the mathematical treatment of data, we shall come nearer to having a science of human behavior. (Kinsey, Pomeroy, and Martin 1948, p 35).

In much — even most — research in social and physical sciences, statistical testing is not necessary. Where there are large differences between different sorts of circumstances for example, if a new medicine cures 90 patients out of 100 and the old medicine cures only 10 patients out of 100 — we do not need refined statistical tests to tell us whether or not the new medicine really has an effect. And the best research is that which shows large differences, because it is the large effects that matter. If the researcher finds that s/he must use refined statistical tests to reveal whether there are differences, this sometimes means that the differences do not matter much.

To repeat, then, some or even much research — especially in the physical and biological sciences — does not need the kind of statistical manipulation that will be described in this book. But most decision problems *do* need the kind of probabilistic and statistical input that is described in this book.

Another matter: If the raw data are of poor quality, probabilistic and statistical manipulation cannot be very useful. In the example of the contractor and her ambulances, if the contractor's estimate that a given ambulance has a one-in-ten chance of being unfit for service out-of-order on a given day is very inaccurate, then our calculation of the probability that three or more ambulances will be out of order on a given day will not be helpful, and may be misleading. To put it another way, one cannot make bread without flour, yeast, and water. And good raw data are the flour, yeast and water necessary to get an accurate estimate of a probability. The most refined statistical and probabilistic manipulations are useless if the input data are poor

— the result of unrepresentative samples, uncontrolled experiments, inaccurate measurement, and the host of other ways that information gathering can go wrong. (See Simon and Burstein (1985) for a catalog of the obstacles to obtaining good data.) Therefore, we should constantly direct our attention to ensuring that the data upon which we base our calculations are the best it is possible to obtain.

1.6 Why is Statistics Such a Difficult Subject?

Why is statistics such a tough subject for so many people?

"Among mathematicians and statisticians who teach introductory statistics, there is a tendency to view students who are not skillful in mathematics as unintelligent," say two of the authors of a popular introductory text (McCabe and McCabe 1989, p 2). As these authors imply, this view is out-and-out wrong; lack of general intelligence on the part of students is *not* the root of the problem.

Scan this book and you will find almost no formal mathematics. Yet nearly every student finds the subject very difficult — as difficult as anything taught at universities. The root of the difficulty is that the *subject matter* is extremely difficult. Let's find out *why*.

It is easy to find out with high precision which movie is playing tonight at the local cinema; you can look it up on the web or call the cinema and ask. But consider by contrast how difficult it is to determine with accuracy:

1. Whether we will save lives by recommending vitamin D supplements for the whole population as protection against viral infections. Some evidence suggests that low vitamin D levels predispose to more severe lung infections, and that taking supplements can help (Martineau et al. 2017). But, how certain can we be of the evidence? How safe are the supplements? Does the benefit, and the risk, differ by ethnicity?
2. What will be the result of more than a hundred million Americans voting for president a month from now; the best attempt usually is a sample of 2000 people, selected in some fashion or another that is far from random, weeks before the election, asked questions that are by no means the same as the actual voting act, and so on;
3. How men feel about women and vice versa.

The cleverest and wisest people have pondered for thousands of years how to obtain answers to questions like these, and made little progress. Dealing with uncertainty was completely outside the scope of the ancient philosophers. It was not until two or three hundred years ago that people began to make any progress at all on these sorts of questions, and it was only about one century ago that we began to have reasonably competent procedures — simply because the problems are inherently difficult. So it is no wonder that the body of these methods is difficult.

So: The bad news is that the subject is extremely difficult. The good news is that you — and that means *you* — can understand it with hard thinking, even if you have no mathematical background beyond arithmetic and you think that you have no mathematical capability. That's because the difficulty lies in such matters as pin-pointing the right question, but not in any difficulties of mathematical manipulation.

2 The resampling method

This chapter is a brief introduction to the resampling method of solving problems in probability and statistics. We're going to dive right in and solve a problem hands-on.

You will see that the resampling method is easy to understand and apply: all it requires is to understand the physical problem. You then simulate a statistical model of the physical problem with techniques that are intuitively obvious, and estimate the probability sought with repeated random sampling.

After finding a solution, we will look at the more conventional formulaic approach, and how that compares. Here's the spoiler: it requires you to understand complex formulas, and to choose the correct one from many.

After reading this chapter, you will understand why we are excited about the resampling method, and why it will allow you to approach even even hard problems without knowing sophisticated statistic techniques.

2.1 The resampling approach in action

Recall the problem from section Section 1.2 in which the contractor owns 16 ambulances:

> *You are the manager and part owner* of one of several contractors providing ambulances to a hospital. You own 16 ambulances. Based on past experience, the chance that any one ambulance will be unfit for service on any given day is about one in ten. You want to know the chance on a particular day — tomorrow — that three or more of them will be out of action.

The resampling approach produces the estimate as follows.

2.1.1 Randomness from physical methods

We collect 10 coins, and mark one of them with a pen or pencil as being the coin that represents "out-of-order;" the other nine coins stand for "in operation". For any one ambulance, this set of 10 coins provides a "model" for the one-in-ten chance — a probability of .10 (10 percent) — of it being out of order on a given day. We put the coins into a little jar or bucket.

For ambulance #1, we draw a single coin from the bucket. This coin represents whether that ambulance is going to be broken tomorrow. If we draw the marked coin, we label this ambulance as out-of-order, otherwise we label the ambulance as in-operation. After replacing the coin and shaking the bucket, we repeat the same procedure for ambulance #2, ambulance #3 and so forth. Having repeated the procedure *16* times, we now have a representation of all ambulances for a single day.

We can now repeat this whole process as many times as we like: each time, we come up with a representation for a different day, telling us how many ambulances will be out-of-service on that day.

After collecting evidence for, say, 50 experimental days we determine the proportion of the experimental days on which three or more ambulances are out of order. That proportion is an estimate of the probability that three or more ambulances will be out of order on a given day — the answer we seek. This procedure is an example of Monte Carlo simulation, which is the heart of the resampling method of statistical estimation.

A more direct way to answer this question would be to examine the firm's actual records for the past 100 days or, better, 500 days (if that's available) to determine how many days had three or more ambulances out of order. But the resampling procedure described above gives us an estimate even if we do not have such long-term information. This is realistic; it is frequently the case in the real world that we must make estimates on the basis of insufficient history about an event.

A quicker resampling method than the coins could be obtained with 16 ten-sided dice or spinners (like those found in Dungeons & Dragons games). For each die, we identify one of its ten sides as "out-of-order".

Standard 10-sided dice have the numbers 0 through 9 on their faces, rather than 1 through 10. Figure 2.1 shows a standard 10-sided die:

We decide, arbitrarily, that the 9 side means "out-of-order". We could even put a little bit of paint on the 9 side to remind us. The die represents an ambulance. If we roll the die, and get this face, this indicates that the ambulance was out of order. If we get any of the other faces — 0 through 8 — this ambulance was in working order. A single throw of all 16 dice will be our experimental trial that represents a single day; we just have to count whether three or more ambulances turn up "out of order". Figure 2.2 show the result of one trial — throwing 16 dice:

As you can see, the trial in Figure 2.2 gave us a single 9, so there was only one ambulance out of order.

In a hundred throws of the 16 dice — which probably takes less than 5 minutes — we can get a fast and reasonably accurate answer to our question.

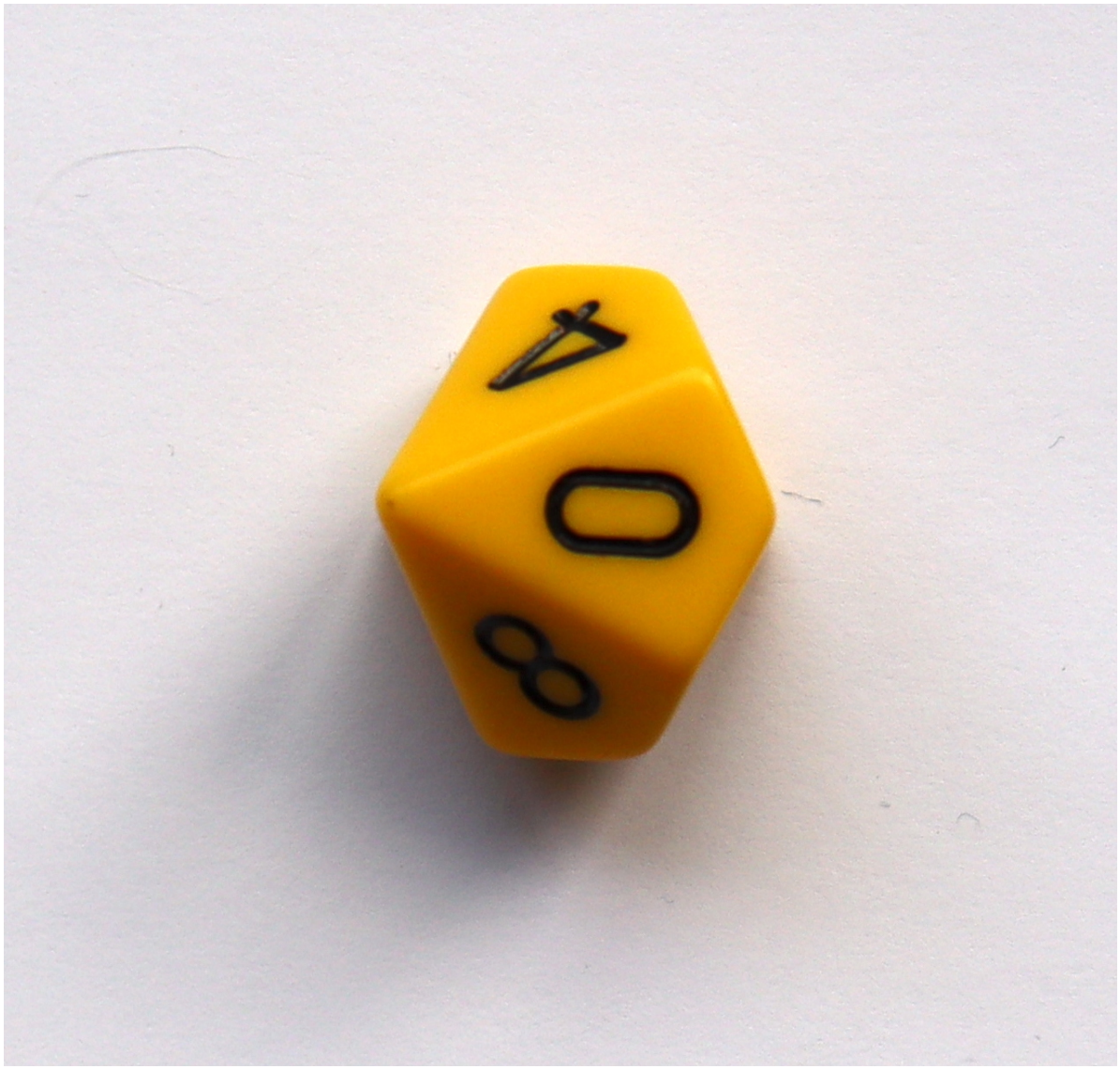


Figure 2.1: 10-sided die

Figure 2.2: 16 10-sided dice

2.2 Randomness from your computer

Computers make it easy to generate random numbers for resampling.

> **What do we mean by random?**
>
> Random numbers are numbers where it is impossible to predict which number is coming next. If we ask the computer for a number between 0 and 9, we will get one of the numbers 0 though 9, but we cannot do any better than that in predicting *which* number it will give us. There is an equal (10%) chance we will get any of the numbers 0 through 9 — just as there is when we roll a fair 10-sided die. We will go into more detail about what exactly we mean by *random* and *chance* later in the book (Section 3.8).

We can use random numbers from computers to simulate our problem. For example, we can ask the computer to choose a random number between 0 and 9 to represent one ambulance. Let's say the number 9 represents "out-of-order" and 0 through 8 "in operation", then any one random number gives us a trial observation for a single ambulance. To get an experimental trial for a single day we look at 16 numbers and count how many of them are 9. We then look at, say, one hundred sets of 16 numbers and count the proportion of sets whose 16 numbers show three or more ambulances being "out-of-order". Once again, that proportion estimates the probability that three or more ambulances will be out-of-order on any given day.

Soon we will do all these steps with some Python code, but for now, consider Table 2.1. In each row, we placed 16 numbers, each one representing an ambulance. We added 25 such rows, each representing a simulation of one day.

Table 2.1: 25 simulations of 16 ambulances

Day	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
1	5	4	4	5	9	8	2	9	1	5	8	2	1	8	2	6
2	6	5	0	5	2	7	4	4	6	3	9	5	2	5	8	1
3	2	5	4	9	0	5	8	4	5	9	1	2	8	7	5	3
4	8	9	2	6	9	0	7	2	5	2	2	2	2	4	7	6
5	0	4	5	1	3	7	6	3	2	9	5	8	0	6	0	4
6	7	4	8	9	1	5	1	2	3	6	4	8	5	1	7	5
7	0	9	8	7	7	3	9	1	7	7	9	9	6	8	4	7
8	7	2	0	2	4	6	9	2	3	9	5	3	7	1	3	0
9	8	0	0	3	3	0	0	3	8	6	4	6	0	4	6	7
10	9	7	1	9	8	1	8	7	0	4	4	7	0	5	6	1
11	0	9	0	7	0	1	6	0	8	6	0	3	1	9	8	3
12	1	2	7	8	8	6	1	0	8	3	4	5	8	8	4	9

13	1	0	8	6	9	2	0	7	7	0	0	7	9	2	3	0
14	0	0	5	5	4	0	1	7	8	2	0	8	3	2	2	4
15	6	3	9	6	8	8	7	6	6	4	3	8	7	0	4	3
16	4	2	6	9	0	0	8	5	3	1	5	1	8	7	6	8
17	3	6	3	5	3	1	2	4	3	1	6	2	9	5	2	4
18	0	6	1	9	0	7	9	4	2	0	1	5	8	5	8	1
19	3	2	2	7	8	2	2	1	2	9	2	5	9	9	6	0
20	6	3	3	2	6	8	3	9	0	5	7	8	8	3	8	6
21	8	3	0	0	1	5	3	7	0	9	6	4	1	2	5	0
22	1	8	7	1	7	1	2	6	4	3	0	0	7	5	6	2
23	9	2	8	0	3	1	9	1	5	6	5	9	8	4	3	0
24	6	7	4	9	4	2	0	6	1	0	4	1	0	5	5	9
25	9	4	3	4	1	6	9	2	4	3	1	8	6	8	0	2

To know how many ambulances were "out of order" on any given day, we count number of nines in that row. We place the counts in the final column called "#9" (for "number of nines"):

Table 2.2: 25 simulations of 16 ambulances, with counts

Day	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	**#9**
1	5	4	4	5	9	8	2	9	1	5	8	2	1	8	2	6	**2**
2	6	5	0	5	2	7	4	4	6	3	9	5	2	5	8	1	**1**
3	2	5	4	9	0	5	8	4	5	9	1	2	8	7	5	3	**2**
4	8	9	2	6	9	0	7	2	5	2	2	2	2	4	7	6	**2**
5	0	4	5	1	3	7	6	3	2	9	5	8	0	6	0	4	**1**
6	7	4	8	9	1	5	1	2	3	6	4	8	5	1	7	5	**1**
7	0	9	8	7	7	3	9	1	7	7	9	9	6	8	4	7	**4**
8	7	2	0	2	4	6	9	2	3	9	5	3	7	1	3	0	**2**
9	8	0	0	3	3	0	0	3	8	6	4	6	0	4	6	7	**1**
10	9	7	1	9	8	1	8	7	0	4	4	7	0	5	6	1	**2**
11	0	9	0	7	0	1	6	0	8	6	0	3	1	9	8	3	**2**
12	1	2	7	8	8	6	1	0	8	3	4	5	8	8	4	9	**1**
13	1	0	8	6	9	2	0	7	7	0	0	7	9	2	3	0	**2**
14	0	0	5	5	4	0	1	7	8	2	0	8	3	2	2	4	**0**
15	6	3	9	6	8	8	7	6	6	4	3	8	7	0	4	3	**1**
16	4	2	6	9	0	0	8	5	3	1	5	1	8	7	6	8	**1**
17	3	6	3	5	3	1	2	4	3	1	6	2	9	5	2	4	**1**
18	0	6	1	9	0	7	9	4	2	0	1	5	8	5	8	1	**2**
19	3	2	2	7	8	2	2	1	2	9	2	5	9	9	6	0	**3**

20	6	3	3	2	6	8	3	9	0	5	7	8	8	3	8	6	**1**
21	8	3	0	0	1	5	3	7	0	9	6	4	1	2	5	0	**1**
22	1	8	7	1	7	1	2	6	4	3	0	0	7	5	6	2	**0**
23	9	2	8	0	3	1	9	1	5	6	5	9	8	4	3	0	**3**
24	6	7	4	9	4	2	0	6	1	0	4	1	0	5	5	9	**2**
25	9	4	3	4	1	6	9	2	4	3	1	8	6	8	0	2	**2**

Each value in the last column of Table 2.2 is the count of 9s in that row and, therefore, the result from our simulation of one day.

We can estimate how often three or more ambulances would break down by looking for values of three or greater in the last column (labeled "#9"). We find there are 3 rows with three or more in the last column. Finally we divide this number of rows by the number of trials (25) to get an estimate of the *proportion* of days with three or more breakdowns. The result is 0.12.

2.3 Solving the problem using Python

Here we rush ahead to show you how to do this simulation in Python.

We go through the Python code for the simulation, but we don't expect you to understand all of it right now. The rest of this book goes into more detail on reading and writing Python code, and how you can use Python to build your own simulations. Here we just want to show you what this code looks like, to give you an idea of where we are headed.

While you can *run* the code below on your own computer, for now we only need you to read it and follow along; the text explains what each line of code does.

> **Coming back to the example**
>
> If you are interested, you can come back to this example later, and run it for yourself. To do this, we recommend you read Chapter 4 that explains how to execute notebooks online or on your own computer.

> **Note 1: Notebook: Ambulances**
>
> - Download notebook
> - Interact

> **Note 2: Comments in code**
>
> The first thing to say about the code you will see below is there are some lines that do not do anything; these are the lines beginning with a `#` character (read `#` as "hash"). Lines beginning with `#` are called *comments*. When Python sees a `#` at the start of a line, it ignores everything else on that line, and skips to the next. Here's an example of a comment:
>
> ```
> # Python will completely ignore this text.
> ```
>
> Because Python ignores lines beginning with `#`, the text after the `#` is just for us, the humans reading the code. The person writing the code will often use comments to explain what the code is doing.

Our next task is to use Python to simulate a single day of ambulances. We will again represent each ambulance by a random number from 0 through 9. 16 of these numbers represents a simulation of all 16 ambulances available to the contractor. We call a simulation of all ambulances for a specific day — one *trial*.

Before we begin our first trial, we need to load some helpful routines from the Numpy software library. Numpy is a Python library that has many important functions for creating and working with numerical data. We will use routines from Numpy in almost all our examples.

```
# Get the Numpy library, and call it "np" for short.
import numpy as np
```

We also need to ask Numpy for something (that we will call an "object") that can generate random numbers. Such an object is known as a "random number generator".

```
# Ask Numpy for a random number generator.
# Name it `rnd` - short for "random"
rnd = np.random.default_rng()
```

Recall that we want 16 10-sided dice — one per ambulance. Our dice should be 10-sided, because each ambulance has a 1-in-10 chance of being out of order.

The program to simulate one trial of the ambulances problem therefore begins with these commands:

```
# Ask Numpy to generate 16 numbers from 0 through 9.

# These are the numbers we will ask Numpy to select from.
```

```
# We store the numbers together in an *array*.
numbers = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Get 16 (size=16) values from the *numbers* array.
# Store the 16 numbers with the name "a"
a = rnd.choice(numbers, size=16)

# The result is a sequence (array) of 16 numbers.
a
```

```
array([5, 4, 4, 5, 9, 8, 2, 9, 1, 5, 8, 2, 1, 8, 2, 6])
```

The commands above ask the computer to store the results of the random drawing in a location in the computer's memory to which we give a name such as `a` or `ambulances` or `aardvark` — the name is up to us.

Next, we need to count the number of defective ambulances:

```
# Count the number of nines in the random numbers.
# The "a == 9" part identifies all the numbers equal to 9.
# The "sum" part counts how many numbers "a == 9" found.
b = np.sum(a == 9)
# Show the result
b
```

```
np.int64(2)
```

You see that we have code like this:

```
np.sum(a == 9)
```

```
np.int64(2)
```

to count the number of values equal to 9 in the sequence `a`. For now just read np.sum(a == 9) as "count the number of 9s in `a`". We will explain how this code works fairly soon (Section 5.13).

The `sum` command is a *counting* operation. It asks the computer to *count* the number of `9`s among the 16 numbers that are in location `a` following the random draw carried out by the `rnd.choice` operation. The result of the `sum` operation will be somewhere between 0 and 16,

the number of simulated ambulances that were out-of-order on a given simulated day. The result is then placed in another location in the computer's memory that we label `b`.

Above you see that we have worked out how to tell the computer to do a single trial — one simulated day.

2.3.1 Repeating trials

We could run the code above for one trial over and over, and write down the result on a piece of paper. If we did this 100 times we would have 100 counts of the number of simulated ambulances that had broken down for each simulated day. To answer our question, we will then count the number of times the count was three or more, and divide by 100, to get an estimate of the proportion of days with three or more out-of-order ambulances.

One of the most useful things about the computer is that it is very good at repeating tasks many times. Our next job is to ask the computer to repeat the single trial many times — say 1000 times — and count up the results for us.

Of course Python is very good at repeating things, but the instructions to tell Python to repeat things will take a little while to get used to. Soon, we will spend some time going over it in more detail. For now though, we show you how what it looks like, and ask you to take our word for it.

The standard way to repeat steps in Python is a `for` loop. For example, let us say we wanted to display (`print`) "Hello" five times. Here is how we would do that with a `for` loop:

```
# Read the next line as "repeat the following steps five times".
for i in np.arange(0, 5):
    # The indented stuff is the code we repeat five times.
    # Print "Hello" to the screen.
    print("Hello")
```

```
Hello
Hello
Hello
Hello
Hello
```

You can probably see where we are going here. We are going to put the code for one trial inside a `for` loop, to repeat that trial code many times.

Our next job is to *store* the results of each trial. If we are going to run 1000 trials, we need to store 1000 results.

To do this, we start with a sequence of 1000 zeros, that we will fill in later, like this:

```
# Ask Numpy to make a sequence of 1000 zeros that we will use
# to store the results of our 1000 trials.
# Call this sequence "z"
z = np.zeros(1000)
```

For now, `z` contains 1000 zeros, but we will soon use a `for` loop to execute 1000 trials. For each trial we will calculate our result (the number of broken-down ambulances), and we will store the result in the `z` store. We end up with 1000 trial results stored in `z`.

With these parts, we are now ready to solve the ambulance problem, using Python.

2.3.2 The solution

This is our big moment! We will combine the elements shown above to perform our ambulance simulation over, say, 1000 days. Just a quick reminder: we do not expect you to understand all the detail of the code below; we will cover that later. For now, see if you can follow along with the gist of it.

To solve resampling problems, we typically proceed as we have done above. We figure out the structure of a single trial and then place that trial in a `for` loop that executes it multiple times (once for each day, in our case).

Now, let us apply this procedure to our ambulance problem. We simulate 1000 days. You will see that we have just taken the parts above, and put them together. The only new part here, is the step at the end, where we store the result of the trial. Bear with us for that; we will come to it soon.

```
# Ask Numpy to make a sequence of 1000 zeros that we will use
# to store the results of our 1000 trials.
# Call this sequence "z"
z = np.zeros(1000)

# These are the numbers we will ask Numpy to select from.
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Read the next line as "repeat the following steps 1000 times".
for i in np.arange(0, 1000):
    # The indented stuff is the code we repeat 1000 times.

    # Get 16 (size=16) values from the *numbers* list.
    # Store the 16 numbers with the name "a"
    a = rnd.choice(numbers, size=16)
```

```
    # Count the number of nines in the random numbers.
    # The "a == 9" part identifies all the numbers equal to 9.
    # The "sum" part counts how many numbers "a == 9" found.
    b = np.sum(a == 9)

    # Store the result from this trial in the sequence "z"
    z[i] = b

    # Now go back and repeat the trial, until done.
```

The `z[i] = b` statement that follows the `sum` *counting* operation simply keeps track of the results of each trial, placing the number of defective ambulances for each trial inside the sequence called `z`. The sequence has 1000 positions: one for each trial.

When we have run the code above, we have stored 1000 trial results in the sequence `z`. These are 1000 counts of out-of-order ambulances, one for each of our simulated days. Our last task is to calculate the proportion of these days for which we had more than three broken-down ambulances.

Since our aim is to count the number of days in which three or more defective ambulances occur, we use another *counting* `sum` command at the end of the 1000 trials. This command *counts* how many times three or more defects occurred in the 1000 days recorded in our `z` sequence, and we place the result in another location, `k`. The value of `k` gives us the total number of days where 3 or more defective ambulances are seen to occur. Then we divide the number in `k` by 1000, the number of trials. Thus we obtain an estimate of the chance, expressed as a probability between 0 and 1, that 3 or more ambulances will be defective on a given day. And we store that result in a location that we call `kk`, which Python subsequently prints to the screen.

```
# How many trials resulted in 3 or more ambulances being out of order?
# (The test counts the number of days where the out-of-order count was greater
# than or equal to 3, written as ">= 3").
k = np.sum(z >= 3)

# Convert to a proportion.
kk = k / 1000

# Print the result.
print(kk)
```

```
0.215
```

This is the estimate we wanted; the proportion of days where three or more ambulances were out of action.

We have crept up on the solution, so it might not be clear to you how few steps you needed to do this task. Here is the whole solution to the problem, without the comments:

```
import numpy as np
rnd = np.random.default_rng()

z = np.zeros(1000)
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for i in np.arange(0, 1000):
    a = rnd.choice(numbers, size=16)
    b = np.sum(a == 9)
    z[i] = b

k = np.sum(z >= 3)
kk = k / 1000
print(kk)
```

```
0.215
```

> **i** End of notebook: Ambulances
>
> `ambulances` starts at Note 1.

Notice that the code above is *exactly* the same as the code we built up in steps. But notice too, that the answer we got from this code was slightly different from the answer we got first.

Why did we get a different answer from the same code? See if you can come up with a good answer before reading the explanation below.

> **i** Randomness in estimates
>
> This is an essential point — our code uses *random numbers* to get an *estimate* of the quantity we want — in this case, the probability of three or more ambulances being out of order. Every run of our code block above will use a different set of random numbers. Therefore, every run of the code will give us a very slightly different number. As you will soon see, we can make our estimate more and more accurate, and less and less different between each run, by doing many trials in each run. Here we did 1000 trials, but we will usually do 10000 trials, to give us a good estimate, that does not vary much from run to run.

Don't worry about the detail of how each of these commands works — we will cover those details gradually, over the next few chapters. But, we hope that you can see, in principle, how each of the operations that the computer carries out are analogous to the operations that you yourself executed when you solved this problem using the ten-sided dice. This is exactly the procedure that we will use to solve every problem in probability and statistics that we must deal with.

While writing programs like these take a bit of getting used to, it is vastly simpler than the older, more conventional approaches to such problems routinely taught to students.

2.4 How resampling differs from the conventional approach

In the standard approach the student learns to choose and solve a formula. Doing the algebra and arithmetic is quick and easy. The difficulty is in choosing the correct formula. Unless you are a professional statistician, it may take you quite a while to arrive at the correct formula — considerable hard thinking, and perhaps some digging in textbooks. More important than the labor, however, is that you may come up with the wrong formula, and hence obtain the wrong answer. And how would you know if you were wrong?

Most students who have had a standard course in probability and statistics are quick to tell you that it is not easy to find the correct formula, even immediately after finishing a course (or several courses) on the subject. After leaving school or university, it is harder still to choose the right formula. Even many people who have taught statistics at the university level (including this writer) must look at a book to get the correct formula for a problem as simple as the ambulances, and then we are often still not sure we have the right answer. This is the grave disadvantage of the standard approach.

In the past few decades, resampling and other Monte Carlo simulation methods have come to be used extensively in scientific research. But in contrast to the material in this book, simulation has mostly been used in situations so complex that mathematical methods have not yet been developed to handle them. Here are examples of such situations:

1. Imagine a large train station such as Grand Central Terminal in New York or King's Cross in London. We are responsible for planning the new station layout so that passengers can move as quickly as possible to and from their trains in rush-hour. It will likely be far too complicated to make formulas to represent the passenger flows, but we could use the computer to simulate passengers, and their movements, and try different potential layouts within the simulation.

2. The Navy might want to know how long the average ship will have to wait for dock facilities. The time of completion varies from ship to ship, and the number of ships waiting in line for dock work varies over time. This problem can be handled quite easily with the experimental simulation method, but formal mathematical analysis would be difficult or impossible.

3. What are the best tactics in baseball? Should one bunt? Should one put the best hitter up first, or later? By trying out various tactics with dice or random numbers, Earnshaw Cook (in his book *Percentage Baseball*), found that it is best never to bunt, and the highest-average hitter should be put up first, in contrast to usual practice. Finding this answer would have been much more difficult with the analytic method.

4. Which search pattern will yield the best results for a ship searching for a school of fish? Trying out "models" of various search patterns with simulation can provide a fast answer.

5. What strategy in the game of Monopoly will be most likely to win? The simulation method systematically plays many games (with a computer) testing various strategies to find the best one.

But those five examples are all complex problems. This book and its earlier editions break new ground by using this method for *simple rather than complex problems*, especially in statistics rather than pure probability, and in teaching *beginning rather than advanced* students to solve problems this way. (Here it is necessary to emphasize that the resampling method is used to *solve the problems themselves rather than as a demonstration device to teach the notions found in the standard conventional approach.* Simulation has been used in elementary courses in the past, but only to demonstrate the operation of the analytical mathematical ideas. That is very different than using the resampling approach to solve statistics problems themselves, as is done here.)

Once we get rid of the formulas and tables, we can see that statistics is a matter of *clear thinking, not fancy mathematics.* Then we can get down to the business of learning how to do that clear statistical thinking, and putting it to work for you. *The study of probability* is purely mathematics (though not necessarily formulas) and technique. But *statistics has to do with meaning.* For example, what is the meaning of data showing an association just discovered between a type of behavior and a disease? Of differences in the pay of men and women in your firm? Issues of causation, acceptability of control, and design of experiments cannot be reduced to technique. This is "philosophy" in the fullest sense. Probability and statistics calculations are just one input. Resampling simulation enables us to get past issues of mathematical technique and focus on the crucial statistical elements of statistical problems.

We hope you will find, as you read through the chapters, that the resampling way of thinking is a good way to think about the more traditional statistical methods that some of you may already know. Our approach will be to use resampling to understand the ideas, and then apply this understanding to reason about traditional methods. You may also find that the resampling methods are not only easier to understand — they are often more useful, because they are so general in their application.

3 What is probability?

> Uncertainty, in the presence of vivid hopes and fears, is painful, but must be endured if we wish to live without the support of comforting fairy tales." — Bertrand Russell (1945 p. *xiv*).

3.1 Introduction

The central concept for dealing with uncertainty is probability. Hence we must inquire into the "meaning" of the term probability. (The term "meaning" is in quotes because it can be a confusing word.)

You have been using the notion of probability all your life when drawing conclusions about what you expect to happen, and in reaching decisions in your public and personal lives.

You wonder: Will the kick from the 45 yard line go through the uprights? How much oil can you expect from the next well you drill, and what value should you assign to that prospect? Will you make money if you invest in tech stocks for the medium term, or should you spread your investments across the stock market? Will the next Space-X launch end in disaster? Your answers to these questions rest on the probabilities you estimate.

And you act on the basis of probabilities: You pay extra for an low-interest loan, if you think that interest rates are going to go up. You bet heavily on a poker hand if there is a high probability that you have the best hand. A hospital decides not to buy another ambulance when the administrator judges that there is a low probability that all the other ambulances will ever be in use at once. NASA decides whether or not to send off the space shuttle this morning as scheduled.

The idea of probability is essential when we reason about uncertainty, and so this chapter discusses what is meant by such key terms as "probability," "chance", "sample," and "universe." It discusses the nature and the usefulness of the concept of probability as used in this book, and it touches on the source of basic estimates of probability that are the raw material of statistical inferences.

3.2 The "Meaning" of "Probability"

Probability is difficult to define (Feller 1968), but here is a useful informal starting point:

> A probability is a number from 0 through 1 that reflects how likely it is that a particular event will happen.

Any particular stated probability is an assertion that indicates how likely you believe it is that an event will occur.

If you give an event a probability of 0 you mean that you are certain it will *not* happen. If you give probability 1 to an event, you mean you are certain that it *will* happen. For example, if I give you one card from deck that you know contains only the standard 52 cards — before you look at the card, you can give probability 0 to the card being a joker, because you are certain the pack does not contain any joker cards. If I then select only the 14 spades from that deck, and give you a card from that selection, you will say there is probability 1 that the card is a black card, because all the spades are black cards.

A probability estimate of .2 indicates that you think there is twice as great a chance of the event happening as if you had estimated a probability of .1. This is the rock-bottom interpretation of the term "probability," and the heart of the concept.

i Expressing probability

A given probability may be expressed in terms of probability, odds, or chances, and I shall use all three terms to help familiarize you with them.
Let us say we think there is a probability of 0.1 that it will rain tomorrow.
We can restate this probability by saying there is a one in 10 *chance* that it will rain tomorrow ($1/10 = 0.1$). Giving the *chances* as 1 in 10, or 2 in 20, or 10 in 100, is the same as saying the probability is 0.1.
If we multiply the probability by 100 we get the *percent chance* — another way of saying the probability. Here we have a $0.1 * 100 = 10\%$ chance of rain. We could also say that the chances of rain are 10 in 100.
Odds are still another way of expressing probability. Here we think of our outcome of interest — a day *with rain* and compare it to our outcome that is not of interest — a day *without rain*. Our probability of 0.1 means that we expect one day *with rain* in every 10 days, and therefore, one day *with rain* for every nine days *without rain*. We can express the 0.1 probability of rain as *odds* 1 to 9 (of a rainy day), or 9 to 1 *against* a rainy day.
"Likelihood" is a term related to "probability" but is not a complete synonym for it — it has a specific and technical meaning in probability and statistics.

The idea of probability arises when you are not sure about what will happen in an uncertain situation. For example, you may lack information and therefore can only make an estimate. If someone asks you your name, you do not use the concept of probability to answer; you know

the answer to a very high degree of surety. To be sure, there is some chance that you do not know your own name, but for all practical purposes you can be quite sure of the answer. If someone asks you who will win tomorrow's baseball game, however, there is a considerable chance that you will be wrong no matter what you say. Whenever there is a reasonable chance that your prediction will be wrong, the concept of probability can help you.

The concept of probability helps you to answer the question, "How likely is it that…?" The purpose of the study of probability and statistics is to help you make sound appraisals of statements about the future, and good decisions based upon those appraisals. The concept of probability is especially useful when you have a sample from a larger set of data — a "universe" — and you want to know the probability of various degrees of likeness between the sample and the universe. (The universe of events you are sampling from is also called the "population," a concept to be discussed below.) Perhaps the universe of your study is all high school graduates in 2018. You might then want to know, for example, the probability that the universe's average SAT (university entrance) score will not differ from your sample's average SAT by more than some arbitrary number of SAT points — say, ten points.

We have said that a probability statement is about the future. Well, usually. Occasionally you might state a probability about your future knowledge of past events — that is, "I think I'll find out that…" — or even about the unknown past. (Historians use probabilities to measure their uncertainty about whether events occurred in the past, and the courts do, too, though the courts hesitate to say so explicitly.)

Sometimes one knows a probability, such as in the case of a gambler playing black on an honest roulette wheel, or an insurance company issuing a policy on an event with which it has had a lot of experience, such as a life insurance policy. But often one does not *know* the probability of a future event. Therefore, our concept of probability must include situations where extensive data are not available.

All of the many techniques used to estimate probabilities should be thought of as *proxies* for the actual probability. For example, if Mission Control at Space Central simulates what should and probably will happen in space if a valve is turned aboard a space craft just now being built, the test result on the ground is a proxy for the real probability of what will happen when the crew turn the valve in the planned mission.

In some cases, it is difficult to conceive of *any* data that can serve as a proxy. For example, the director of the CIA, Robert Gates, said in 1993 "that in May 1989, the CIA reported that the problems in the Soviet Union were so serious and the situation so volatile that Gorbachev had only a 50-50 chance of surviving the next three to four years unless he retreated from his reform policies" (*The Washington Post*, January 17, 1993, p. A42). Can such a statement be based on solid enough data to be more than a crude guess?

The conceptual probability in any specific situation is *an interpretation of all the evidence that is then available*. For example, a wise biomedical worker's estimate of the chance that a given therapy will have a positive effect on a sick patient should be an interpretation of the results of not just one study in isolation, but of the results of that study plus everything else that is

known about the disease and the therapy. A wise policymaker in business, government, or the military will base a probability estimate on a wide variety of information and knowledge. The same is even true of an insurance underwriter who bases a life-insurance or shipping-insurance rate not only on extensive tables of long-time experience but also on recent knowledge of other kinds. Each situation asks us to make a choice of the best method of estimating a probability — whether that estimate is *objective* — from a frequency series — or *subjective*, from the distillation of other experience.

3.3 The nature and meaning of the concept of probability

It is confusing and unnecessary to inquire what probability "really" is. (Indeed, the terms "really" and "is," alone or in combination, are major sources of confusion in statistics and in other logical and scientific discussions, and it is often wise to avoid their use.) Various concepts of probability — which correspond to various common definitions of the term — are useful in particular contexts. This book contains many examples of the use of probability. Work with them will gradually develop a sound understanding of the concept.

There are two major concepts and points of view about probability — *frequency* and *degrees of belief*. Each is useful in some situations but not in others. Though they may seem incompatible in principle, there almost never is confusion about which is appropriate in a given situation.

1. *Frequency*: The probability of an event can be said to be the proportion of times that the event has taken place in the past, usually based on a long series of trials. Insurance companies use this when they estimate the probability that a thirty-five-year-old teacher will die during a period for which he wants to buy an insurance policy. (Notice this shortcoming: Sometimes you must bet upon events that have never or only infrequently taken place before, and so you cannot reasonably reckon the proportion of times they occurred one way or the other in the past.)
2. *Degree of belief*: The probability that an event will take place or that a statement is true can be said to correspond to the odds at which you would bet that the event will take place. (Notice a shortcoming of this concept: You might be willing to accept a five-dollar bet at 2-1 odds that your team will win the game, but you might be unwilling to bet a hundred dollars at the same odds.)

See (Barnett 1982, chap. 3) for an in-depth discussion of different approaches to probability.

The connection between gambling and immorality or vice troubles some people about gambling examples. On the other hand, the immediacy and consequences of the decisions that the gambler has to make give the subject a special tang. There are several reasons why statistics use so many gambling examples — and especially tossing coins, throwing dice, and playing cards:

1. *Historical*: The theory of probability began with gambling examples of dice analyzed by Cardano, Galileo, and then by Pascal and Fermat.
2. *Generality*: These examples are not related to any particular walk of life, and therefore they can be generalized to applications in any walk of life. Students in any field — business, medicine, science — can feel equally at home with gambling examples.
3. *Sharpness*: These examples are particularly stark, and unencumbered by the baggage of particular walks of life or special uses.
4. *Universality*: Many other texts use these same examples, and therefore the use of them connects up this book with the main body of writing about probability and statistics.

Often we'll begin with a gambling example and then consider an example in one of the professional fields — such as business and other decision-making activities, biostatistics and medicine, social science and natural science — and everyday living. People in one field often can benefit from examples in others; for example, medical students should understand the need for business decision-making in terms of medical practice, as well as the biostatistical examples. And social scientists should understand the decision-making aspects of statistics if they have any interest in the use of their work in public policy.

3.4 Back to Proxies

Example of a proxy: The "probability risk assessments" (PRAs) that are made for the chances of failures of nuclear power plants are based, not on long experience or even on laboratory experiment, but rather on theorizing of various kinds — using pieces of prior experience wherever possible, of course. A PRA can cost a nuclear facility many millions of dollars.

Another example: If a manager of a high-street store looks at the sales of a particular brand of smart watches in the last two Decembers, and on that basis guesses how likely it is that she will run out of stock if she orders 200 smart watches, then the last two years' experience is serving as a proxy for future experience. If a sales manager just "intuits" that the odds are 3 to 1 (a probability of .75) that the main local competitor will not meet a price cut, then all her past experience summed into her intuition is a proxy for the probability that it will really happen. Whether any proxy is a good or bad one depends on the wisdom of the person choosing the proxy and making the probability estimates.

How does one estimate a probability in practice? This involves practical skills not very different from the practical skills required to estimate with accuracy the length of a golf shot, the number of carpenters you will need to build a house, or the time it will take you to walk to a friend's house; we will consider elsewhere some ways to improve your practical skills in estimating probabilities. For now, let us simply categorize and consider in the next section various ways of estimating an ordinary garden variety of probability, which is called an "unconditional" probability.

3.5 The various ways of estimating probabilities

Consider the probability of drawing an even-numbered spade from a deck of poker cards (consider the queen as even and the jack and king as odd). Here are several general methods of estimation, where we define each method in terms of the operations we use to make the estimate:

1. **Experience.**

   The first possible source for an estimate of the probability of drawing an even-numbered spade is the purely empirical method of *experience*. If you have watched card games casually from time to time, you might simply guess at the proportion of times you have seen even-numbered spades appear — say, "about 1 in 15" or "about 1 in 9" (which is almost correct) or something like that. (If you watch long enough you might come to estimate something like 6 in 52.)

   General information and experience are also the source for estimating the probability that the sales of a particular brand of smart watch this December will be between 200 and 250, based on sales the last two Decembers; that your team will win the football game tomorrow; that war will break out next year; or that a United States astronaut will reach Mars before a Chinese astronaut. You simply put together all your relevant prior experience and knowledge, and then make an educated guess.

   Observation of repeated events can help you estimate the probability that a machine will turn out a defective part or that a child can memorize four nonsense syllables correctly in one attempt. You watch repeated trials of similar events and record the results.

   Data on the mortality rates for people of various ages in a particular country in a given decade are the basis for estimating the probabilities of death, which are then used by the actuaries of an insurance company to set life insurance rates. This is *systematized experience* — called a *frequency series*.

   No frequency series can speak for itself in a perfectly objective manner. Many judgments inevitably enter into compiling every frequency series — deciding which frequency series to use for an estimate, choosing which part of the frequency series to use, and so on. For example, should the insurance company use only its records from last year, which will be too few to provide as much data as is preferable, or should it also use death records from years further back, when conditions were slightly different, together with data from other sources? (Of course, no two deaths — indeed, no events of any kind — are *exactly* the same. But under many circumstances they are *practically* the same, and science is only interested in such "practical" considerations.)

   Given that we have to use judgment in probability estimates, the reader may prefer to talk about "degrees of belief" instead of probabilities. That's fine, just as long as it is understood that we operate with degrees of belief in exactly the same way as we operate with probabilities; the two terms are working synonyms.

There is no *logical* difference between the sort of probability that the life insurance company estimates on the basis of its "frequency series" of past death rates, and the manager's estimates of the sales of smart watches in December, based on sales in that month in the past two years.[1]

The concept of a probability based on a frequency series can be rendered almost useless when all the observations are repetitions of a single magnitude — for example, the case of all successes and zero failures of space-shuttle launches prior to the Challenger shuttle tragedy in the 1980s; in those data alone there was almost no basis to estimate the probability of a shuttle failure. (Probabilists have made some rather peculiar attempts over the centuries to estimate probabilities from the length of a zero-defect time series — such as the fact that the sun has never failed to rise (foggy days aside!) — based on the undeniable fact that the longer such a series is, the smaller the probability of a failure; see e.g., (Whitworth 1897, xix–xli). However, one surely has more information on which to act when one has a long series of observations of the same magnitude rather than a short series).

2. **Simulated experience.**

   A second possible source of probability estimates is empirical scientific investigation with repeated trials of the phenomenon. This is an empirical method even when the empirical trials are simulations. In the case of the even-numbered spades, the empirical scientific procedure is to shuffle the cards, deal one card, record whether or not the card is an even-number spade, replace the card, and repeat the steps a good many times. The proportions of times you observe an even-numbered spade come up is a probability estimate based on a frequency series.

   You might reasonably ask why we do not just *count* the number of even-numbered spades in the deck of fifty-two cards — using the *sample space analysis* you see below. No reason at all. But that procedure would not work if you wanted to estimate the probability of a baseball batter getting a hit or a lighter producing flame.

   Some varieties of poker are so complex that experiment is the only feasible way to estimate the probabilities a player needs to know.

[1] At one time, some writers believed there was a difference between "objectively sharply defined" and "objectively vague" probabilities. Raiffa (1968) gives a clear example of why this is not so:

Suppose you are confronted with two options. In option 1, you must toss coin 1 (which is fair and true), guess heads or tails, and win $1.00 if you match and lose $1.00 if you fail to match. In option 2, you have a 50-50 chance of getting coin 2, which has two heads, or of getting coin 3, which has two tails. Not knowing whether you are tossing coin 2 or 3, you must call, toss, and get the payoffs as in option 1. With option 1, the probability of the toss coming out heads is .5; with option 2, the same probability is either 0 or 1, and since the chance of each in turn is .5, the probability of heads is ultimately .5 once again. Nothing is to be gained by saying that one .5 is sharply defined and that the other is fuzzy. Of course, *if*, and this is a big "if," you could experiment with the coin you will toss before you are obliged to declare, then the two options are manifestly asymmetrical. Barring this privilege, the two options are equivalent (Raiffa 1968, 108).

The resampling approach to statistics produces estimates of most probabilities with this sort of experimental "Monte Carlo" method. More about this later.

3. **Sample space analysis and first principles.**

   A third source of probability estimates is *counting the possibilities* — the quintessential theoretical method. For example, by examination of an ordinary die one can determine that there are six different numbers that can come up. One can then determine that the probability of getting (say) either a "1" *or* a "2," on a single throw, is $2/6 = 1/3$, because two among the six possibilities are "1" or "2." One can similarly determine that there are two possibilities of getting a "1" *plus* a "6" out of thirty-six possibilities when rolling two dice, yielding a probability estimate of $2/36 = 1/18$.

   Estimating probabilities by counting the possibilities has two requirements: 1) that the possibilities all be known (and therefore limited), and few enough to be studied easily; and 2) that the probability of each particular possibility be known, for example, that the probabilities of all sides of the dice coming up are equal, that is, equal to 1/6.

4. **Mathematical shortcuts to sample-space analysis.**

   A fourth source of probability estimates is *mathematical calculations*. (We will introduce some probability calculation rules in Chapter 9.) If one knows by other means that the probability of a spade is 1/4 and the probability of an even-numbered card is 6/13, one can use probability calculation rules to calculate that the probability of turning up an even-numbered spade is 6/52 (that is, 1/4 x 6/13). (This is *multiplication rule* introduced in Section 8.12). If one knows that the probability of a spade is 1/4 and the probability of a heart is 1/4, one can then calculate that the probability of getting a heart *or* a spade is 1/2 (that is 1/4 + 1/4). (We are using the *addition rule* from Section 8.7.) The point here is not the particular calculation procedures, which we will touch on later, but rather that one can often calculate the desired probability on the basis of already-known probabilities.

   It is possible to estimate probabilities with mathematical calculation only if one knows *by other means* the probabilities of some related events. For example, there is no possible way of mathematically calculating that a child will memorize four nonsense syllables correctly in one attempt; empirical knowledge is necessary.

5. **Kitchen-sink methods.**

   In addition to the above four categories of estimation procedures, the statistical imagination may produce estimates in still other ways such as a) the salesman's seat-of-the-pants estimate of what the competition's price will be next quarter, based on who-knows-what gossip, long-time acquaintance with the competitors, and so on, and b) the probability risk assessments (PRAs) that are made for the chances of failures of nuclear power plants based, not on long experience or even on laboratory experiment, but rather on theorizing of various kinds — using pieces of prior experience wherever possible, of course. Any of these methods may be a combination of theoretical and empirical methods.

As an example of an organization struggling with kitchen-sink methods, consider the estimation of the probability of failure for the tragic flight of the Challenger shuttle, as described by the famous physicist Nobelist Richard Feynman. This is a very real case that includes just about every sort of complication that enters into estimating probabilities.

> …Mr. Ullian told us that 5 out of 127 rockets that he had looked at had failed — a rate of about 4 percent. He took that 4 percent and divided it by 4, because he assumed a manned flight would be safer than an unmanned one. He came out with about a 1 percent chance of failure, and that was enough to warrant the destruct charges.
>
> But NASA [the space agency in charge] told Mr. Ullian that the probability of failure was more like 1 in 10^5.
>
> I tried to make sense out of that number. "Did you say 1 in 10^5?"
>
> "That's right; 1 in 100,000."
>
> "That means you could fly the shuttle *every day* for an average of *300 years* between accidents — every day, one flight, for 300 years — which is obviously crazy!"
>
> "Yes, I know," said Mr. Ullian. "I moved my number up to 1 in 1000 to answer all of NASA's claims — that they were much more careful with manned flights, that the typical rocket isn't a valid comparison, etcetera."
>
> But then a new problem came up: the Jupiter probe, *Galileo*, was going to use a power supply that runs on heat generated by radioactivity. If the shuttle carrying *Galileo* failed, radioactivity could be spread over a large area. So the argument continued: NASA kept saying 1 in 100,000 and Mr. Ullian kept saying 1 in 1000, at best.
>
> Mr. Ullian also told us about the problems he had in trying to talk to the man in charge, Mr. Kingsbury: he could get appointments with underlings, but he never could get through to Kingsbury and find out how NASA got its figure of 1 in 100,000 (Feynman and Leighton 1988, 179–80).

Feynman tried to ascertain more about the origins of the figure of 1 in 100,000 that entered into NASA's calculations. He performed an experiment with the engineers:

> …"Here's a piece of paper each. Please write on your paper the answer to this question: what do you think is the probability that a flight would be uncompleted due to a failure in this engine?"
>
> They write down their answers and hand in their papers. One guy wrote "99-44/100% pure" (copying the Ivory soap slogan), meaning about 1 in 200. Another

guy wrote something very technical and highly quantitative in the standard statistical way, carefully defining everything, that I had to translate — which also meant about 1 in 200. The third guy wrote, simply, "1 in 300."

Mr. Lovingood's paper, however, said:

"Cannot quantify. Reliability is judged from:

- past experience
- quality control in manufacturing
- engineering judgment"

"Well," I said, "I've got four answers, and one of them weaseled." I turned to Mr. Lovingood: "I think you weaseled."

"I don't think I weaseled."

"You didn't tell me *what* your confidence was, sir; you told me *how* you determined it. What I want to know is: after you determined it, what *was* it?"

He says, "100 percent" — the engineers' jaws drop, my jaw drops; I look at him, everybody looks at him — "uh, uh, minus epsilon!"

So I say, "Well, yes; that's fine. Now, the only problem is, WHAT IS EPSILON?"

He says, "10^−5." It was the same number that Mr. Ullian had told us about: 1 in 100,000.

I showed Mr. Lovingood the other answers and said, "You'll be interested to know that there *is* a difference between engineers and management here — a factor of more than 300."

He says, "Sir, I'll be glad to send you the document that contains this estimate, so you can understand it."

Later, Mr. Lovingood sent me that report. It said things like "The probability of mission success is necessarily very close to 1.0" — does that mean it *is* close to 1.0, or it *ought to be* close to 1.0? — and "Historically, this high degree of mission success has given rise to a difference in philosophy between unmanned and manned space flight programs; i.e., numerical probability versus engineering judgment." As far as I can tell, "engineering judgment" means they're just going to make up numbers! The probability of an engine-blade failure was given as a universal constant, as if all the blades were exactly the same, under the same conditions. The whole paper was quantifying everything. Just about every nut and bolt was in there: "The chance that a HPHTP pipe will burst is 10^−7." You can't estimate things like that; a probability of 1 in 10,000,000 is almost impossible to estimate. It was clear that the numbers for each part of the engine were chosen

so that when you add everything together you get 1 in 100,000. (Feynman and Leighton 1988, 182–83).

We see in the Challenger shuttle case very mixed kinds of inputs to actual estimates of probabilities. They include frequency series of past flights of other rockets, judgments about the relevance of experience with that different sort of rocket, adjustments for special temperature conditions (cold), and much much more. There also were complex computational processes in arriving at the probabilities that were made the basis for the launch decision. And most impressive of all, of course, are the extraordinary differences in estimates made by various persons (or perhaps we should talk of various statuses and roles) which make a mockery of the notion of objective estimation in this case.

Working with different sorts of estimation methods in different sorts of situations is not new; practical statisticians do so all the time. We argue that we should make no apology for doing so.

The concept of probability varies from one field of endeavor to another; it is different in the law, in science, and in business. The concept is most straightforward in decision-making situations such as business and gambling; there it is crystal-clear that one's interest is entirely in making accurate predictions so as to advance the interests of oneself and one's group. The concept is most difficult in social science, where there is considerable doubt about the aims and values of an investigation. In sum, one should not think of what a probability "is" but rather how best to estimate it. In practice, neither in actual decision-making situations nor in scientific work — nor in classes — do people experience difficulties estimating probabilities because of philosophical confusions. Only philosophers and mathematicians worry — and even they really do not *need* to worry — about the "meaning" of probability[2].

3.6 The relationship of probability to other magnitudes

An important argument in favor of approaching the concept of probability as an *estimate* is that an estimate of a probability often (though not always) is the opposite side of the coin from an estimate of a physical quantity such as time or space.

For example, uncertainty about the probability that one will finish a task within 9 minutes is another way of labeling the uncertainty that the time required to finish the task will be less than 9 minutes. Hence, if estimation is appropriate for time in this case, it should be equally appropriate for probability. The same is true for the probability that the quantity of smart watches sold will be between 200 and 250 units.

[2]This does not mean that I think that people should confine their learning to what they need in their daily work. Having a deeper philosophical knowledge than you ordinarily need can help you deal with extraordinary problems when they arise.

Hence the concept of probability, and its estimation in any particular case, should be no more puzzling than is the "dual" concept of time or distance or quantities of smart watches. That is, lack of certainty about the probability that an event will occur is not different in nature from lack of certainty about the amount of time or distance in the event. There is no essential difference between whether a part 2 inches in length will be the next to emerge from the machine, or what the length of the next part will be, or the length of the part that just emerged (if it has not yet been measured).

The information available for the measurement of (say) the length of a car or the location of a star is exactly the same information that is available with respect to the concept of probability in those situations. That is, one may have ten disparate observations of a car's length which then constitute a probability distribution, and the same for the altitude of a star in the heavens.

In a book of puzzles about probability (Mosteller 1987, problem 42), this problem appears: "If a stick is broken in two at random, what is the average length of the smaller piece?" This particular puzzle does not even mention probability explicitly, and no one would feel the need to write a scholarly treatise on the meaning of the word "length" here, any more than one would one do so if the question were about an astronomer's average observation of the angle of a star at a given time or place, or the average height of boards cut by a carpenter, or the average size of a basketball team. Nor would one write a treatise about the "meaning" of "time" if a similar puzzle involved the average time between two bird calls. Yet a rephrasing of the problem reveals its tie to the concept of probability, to wit: What is the probability that the smaller piece will be (say) more than half the length of the larger piece? Or, what is the probability distribution of the sizes of the shorter piece?

The duality of the concepts of probability and physical entities also emerges in Whitworth's discussion (1897) of fair betting odds:

> …What sum ought you fairly give or take now, while the event is undetermined, in exchange for the assurance that you shall receive a stated sum (say $1,000) if the favourable event occur? The chance of receiving $1,000 is worth something. It is not as good as the certainty of receiving $1,000, and therefore it is worth less than $1,000. But the prospect or expectation or chance, however slight, is a commodity which may be bought and sold. It must have its price somewhere between zero and $1,000. (p. xix.)
>
> …And the ratio of the expectation to the full sum to be received is what is called the chance of the favourable event. For instance, if we say that the chance is 1/5, it is equivalent to saying that $200 is the fair price of the contingent $1,000. (p. xx.)…
>
> The fair price can sometimes be calculated mathematically from *a priori* considerations: sometimes it can be deduced from statistics, that is, from the recorded results of observation and experiment. Sometimes it can only be estimated generally, the estimate being founded on a limited knowledge or experience. If your

> expectation depends on the drawing of a ticket in a raffle, the fair price can be calculated from abstract considerations: if it depend upon your outliving another person, the fair price can be inferred from recorded statistics: if it depend upon a benefactor not revoking his will, the fair price depends upon the character of your benefactor, his habit of changing his mind, and other circumstances upon the knowledge of which you base your estimate. But if in any of these cases you determine that $300 is the sum which you ought fairly to accept for your prospect, this is equivalent to saying that your chance, whether calculated or estimated, is 3/10... (p. xx.)

It is indubitable that along with frequency data, a wide variety of other information will affect the odds at which a reasonable person will bet. If the two concepts of probability stand on a similar footing here, why should they not be on a similar footing in *all* discussion of probability? I can think of no reason that they should not be so treated.

Scholars write about the "discovery" of the concept of probability in one century or another. But is it not likely that even in pre-history, when a fisherperson was asked how long the big fish was, s/he sometimes extended her/his arms and said, "About this long, but I'm not exactly sure," and when a scout was asked how many of the enemy there were, s/he answered, "I don't know for sure...probably about fifty." The uncertainty implicit in these statements is the functional equivalent of probability statements. There simply is no need to make such heavy work of the probability concept as the philosophers and mathematicians and historians have done.

3.7 What is "chance"?

The study of probability focuses on events with randomness — that is, events about which there is uncertainty whether or not they will occur. And the uncertainty refers to your knowledge rather than to the event itself. For example, consider this physical illustration with a remote control. The remote control has a *front end* that should point at the TV that it controls, and a *back end* that will usually be pointing at me, the user of the remote control. Call the front — the *TV end*, and the back — *the sofa end* of the remote control.

I spin the remote control like a baton twirler. If I hold it at the sofa end and attempt to flip it so that it turns only half a revolution, I can be almost sure that I will correctly get the TV end and not the sofa end. And if I attempt to flip it a full revolution, again I can almost surely get the sofa end successfully. It is not a random event whether I catch the sofa end or the TV end (here ignoring those throws when I catch neither end) when doing only half a revolution or one revolution. The result is quite predictable in both these simple maneuvers so far.

When I say the result is "predictable," I mean that you would not bet with me about whether this time I'll get the TV or the sofa end. So we say that the outcome of my flip aiming at half a revolution is not "random."

When I twirl the remote control so little, I *control* (almost completely) whether the sofa end or the TV end comes down to my hand; this is the same as saying that the outcome does not occur by chance.

The terms "random" and "chance" implicitly mean that you believe that I cannot control or cannot know in advance what will happen.

Whether this twirl will be the rare time I miss, however, *should* be considered chance. Though you would not bet at even odds on my catching the sofa end versus the TV end if there is to be only a half or one full revolution, you might bet — at (say) odds of 50 to 1 — that I will make a mistake and get it wrong, or drop it. So the very same flip can be seen as random or determined depending on what aspect of it we are looking at.

Of course you would not bet *against* me about my *not* making a mistake, because the bet might *cause* me to make a mistake purposely. This "moral hazard" is a problem that emerges when a person buys life insurance and may commit suicide, or when a boxer may lose a fight purposely. The people who stake money on those events say that such an outcome is "fixed" (a very appropriate word) and not random.

Now I attempt more difficult maneuvers with the remote control. I can do $1\frac{1}{2}$ flips pretty well, and two full revolutions with some success — maybe even $2\frac{1}{2}$ flips on a good day. But when I get much beyond that, I cannot determine very well whether I'll get the sofa or the TV end. The outcome gradually becomes less and less predictable — that is, more and more random.

If I flip the remote control so that it revolves three or more times, I can hardly control the process at all, and hence I cannot predict well whether I'll get the sofa end or the TV end. With 5 revolutions I have absolutely no control over the outcome; I cannot predict the outcome better than 50-50. At that point, getting the sofa end or the TV end has become a completely random event for our purposes, just like flipping a coin high in the air. So at that point we say that "chance" controls the outcome, though that word is just a synonym for my lack of ability to control and predict the outcome. "Chance" can be thought to stand for the myriad small factors that influence the outcome.

We see the same gradual increase in randomness with increasing numbers of shuffles of cards. After one shuffle, a skilled magician can know where every card is, and after two shuffles there is still much order that s/he can work with. But after (say) five shuffles, the magician no longer has any power to predict and control, and the outcome of any draw can then be thought of as random chance.

At what point do we say that the outcome is "random" or "pure chance" as to whether my hand will grasp the TV end, the sofa end, or at some other spot? *There is no sharp boundary to this transition.* Rather, the transition is gradual; this is the crucial idea, and one that I have not seen stated before.

Whether or not we refer to the outcome as random depends upon the twirler's skill, which influences how predictable the event is. A baton twirler or juggler might be able to do ten flips

with a non-random outcome; if the twirler is an expert and the outcome is highly predictable, we say it is not random but rather is determined.

Again, this shows that the randomness is not a property of the physical event, but rather of a person's knowledge and skill.

3.8 What Do We Mean by "Random"?

We have defined "chance" and "random" as the absence of predictive power and/or explanation and/or control. Here we should not confuse the concepts of determinacy-indeterminacy and predictable-unpredictable. What matters for *decision purposes* is whether you can predict. Whether the process is "really" determinate is largely a matter of definition and labeling, an unnecessary philosophical controversy for our purposes (and perhaps for any other purpose).[3]

The remote control in the previous demonstration *becomes* unpredictable — that is, random — even though it still is subject to similar physical processes as when it is predictable. I do not deny *in principle* that these processes can be "understood," or that one could produce a machine that would — like a baton twirler — make the course of the remote control predictable for many turns. But in *practice* we cannot make the predictions — and it is the *practical reality*, rather than the principle, that matters here.

When I flip the remote control half a turn or one turn, I control (almost completely) whether it comes down at the sofa end end or the TV end, so we do not say that the outcome is chance. Much the same can be said about what happens to the predictability of drawing a given card as one increases the number of times one shuffles a deck of cards.

Consider, too, a set of fake dice that I roll. Before you know they are fake, you assume that the probabilities of various outcomes is a matter of chance. But after you know that the dice are loaded, you no longer assume that the outcome is chance. This illustrates how the probabilities you work with are influenced by your knowledge of the facts of the situation.

Admittedly, this way of thinking about probability takes some getting used to. Events may appear to be random, but in fact, we can predict them — and *vice versa*. For example, suppose a magician does a simple trick with dice such as this one:

> The magician turns her back while a spectator throws three dice on the table. He is instructed to add the faces. He then picks up any *one* die, adding the number on the *bottom* to the previous total. This same die is rolled again. The number it now

[3] The idea that our aim is to advance our work in improving our knowledge and our decisions, rather than to answer "ultimate" questions about what is "really" true is in the same spirit as some writing about quantum theory. In 1930 Ruarck and Urey wrote: "The reader who feels disappointed that the information sought in solving a dynamical problem on the quantum theory is [only] statistical … should console himself with the thought that we seldom need any information other than that which is given by the quantum theory." (1930, 622).

shows is also added to the total. The magician turns around. She calls attention to the fact that she has no way of knowing which of the three dice was used for the second roll. She picks up the dice, shakes them in her hand a moment, then correctly announces the final sum.

Method:. When the spectator rolls the dice, they get three numbers, one from each of the three dice. Call these numbers a, b and c. Then he chooses one die — it doesn't matter which, but let's say he chooses the third die, with value c. He adds the bottom of the third die to the total. Here's the trick: the total of opposite faces on a standard die always add up to 7; 1 is opposite 6, 2 is opposite 5, and 3 is opposite 4. So the total is now $a + b + 7$. Then the spectator rolls the third die again, to get a new number d. The total is now $a + b + 7 + d$. When the magician turns round she can see what a and b and d are, so to get the right final total, she just needs to add 7 (Gardner 1985, p259). Ben Sparks does a nice demonstration of the trick on Numerphile YouTube.

The point here is that, until you know the trick, you (the magician) cannot predict the final sum, so the magician and the spectator consider the result as random. If you do know the trick, you can predict the result, and it is not random. Whether something is "random" or not, depends on what you know.

Consider the distributions of heights of various groups of living things (including people). When we consider all living things taken together, the shape of the overall distribution — many individuals at the tiny end where the viruses are found, and very few individuals at the tall end where the giraffes are — is determined mostly by the distribution of species that have different mean heights. Hence we can explain the shape of that distribution, and we do not say that is determined by "chance." But with a homogeneous cohort of a single species — say, all 25-year-old human females in the U.S. — our best description of the shape of the distribution is "chance." With situations in between, the shape is partly due to identifiable factors — e.g. age — and partly due to "chance."

Or consider the case of a basketball shooter: What causes her or him to make (or not make) a basket this shot, after a string of successes? Much must be ascribed to chance variation. But what causes a given shooter to be very good or very poor relative to other players? For that explanation we can point to such factors as the amount of practice or natural talent.

Again, all this has nothing to do with whether the mechanism is "really" chance, unlike the arguments that have been raging in physics for a century. That is the point of the remote control demonstration. Our knowledge and our power to predict the outcome gradually transits from non-chance (that is, "determined") to chance ("not determined") in a gradual way even though the same sort of physical mechanism produces each throw of the remote control.

Earlier I mentioned that when we say that chance controls the outcome of the remote control flip after (say) five revolutions, we mean that there are many small forces that affect the outcome. The effect of each force is not known, and each is independent of the other. None of these forces is large enough for me (as the remote control twirler) to deal with, or else I would

deal with it and be able to improve my control and my ability to predict the outcome. This concept of many small influences — "small" meaning in practice those influences whose effects cannot be identified and allowed for — which affect the outcome and whose effects are not knowable and which are independent of each other is important in statistical inference. For example, as we will see later, when we add many unpredictable deviations together, and plot the distribution of the result, we end up with the famous and very common bell-shaped *normal distribution* — this striking result comes about because of a mathematical phenomenon called the Central Limit Theorem.[4]

3.9 Randomness from the computer

We now have the idea of *random* variation as being variation we cannot predict. For example, when we flip the remote control through many rotations, we can no longer easily predict which end will land in our hand. We can call the result of any particular flip — *random* — because we cannot predict whether the result will be TV end or sofa end.

We still know some things about the result — it will be one of two options — TV or sofa (unless we drop it). But we cannot predict which. We say the result of each flip is *random* if we cannot do anything to improve our prediction of 50% for TV (or sofa) end on the next flip.

We are not saying the result *is* random in any deep, non-deterministic sense — we are only saying we can treat the result as random, because we cannot predict it.

Now consider getting *random* numbers from the computer, where the numbers can either be 0 or 1. This is rather like tossing a fair coin, where the results are 0 and 1 rather than "heads" and "tails".

When we ask the computer for a random choice between 0 and 1, we accept it is random-enough, or random-like, if we can't do anything to predict which of 0 or 1 we will get on any one trial. We can't do better than guessing that the next value will be — say — 0 — and whichever number we guess, we will only ever have a 50% chance of being correct. We are not saying the computer is giving *truly* random numbers in the sense that they are fundamentally not deterministic, it is only giving us numbers we cannot *distinguish* from truly random numbers, because we cannot *in practice* do anything to predict them. The technical term for random numbers from the computer is therefore *pseudo-random* — meaning, like random numbers, in the sense they are effectively unpredictable. Effectively unpredictable means there is no *practical* way for you, or even a very powerful computer, to do anything to improve your prediction of the next number in the series.

[4] The Central Limit Theorem is an interesting mathematical result that proves something you can show for yourself by simulation — that if we take means of many values drawn from *any* shape of distribution, and then look at the distribution of the resulting means, it will be close to the *normal* (bell-curve) distribution. If you are interested in a technical (mathematical) explanation of this result, see the Wikipedia page on the Central Limit Theorem.

3.10 The philosophers' dispute about the concept of probability

Those who call themselves "objectivists" or "frequentists" and those who call themselves "personalists" or "Bayesians" have been arguing for hundreds or even thousands of years about the "nature" of probability. The objectivists insist (correctly) that any estimation not based on a series of observations is subject to potential bias, from which they conclude (incorrectly) that we should never think of probability that way. They are worried about the perversion of science, the substitution of arbitrary assessments for value-free data-gathering. The personalists argue (correctly) that in many situations it is not possible to obtain sufficient data to avoid considerable judgment. Indeed, if a probability is about the future, some judgment is *always* required — about which observations will be relevant, and so on. They sometimes conclude (incorrectly) that the objectivists' worries are unimportant.

As is so often the case, the various sides in the argument have different sorts of situations in mind. As we have seen, the arguments disappear if one thinks *operationally* with respect to the *purpose of the work*, rather than in terms of *properties*, as mentioned earlier.

Here is an example of the difficulty of focusing on the supposed properties of the mechanism or situation: The mathematical theorist asserts that the probability of a die falling with the "5" side up is 1/6, on the basis of the physics of equally-weighted sides. But if one rolls a particular die a million times, and it turns up "5" less than 1/6 of the time, one surely would use the observed proportion as the practical estimate. The probabilities of various outcomes with cheap dice may depend upon the number of pips drilled out on a side. In 20,000 throws of a red die and 20,000 throws of a white die, the proportions of 3's and 4's were, respectively, .159 and .146, .145 and .142 — all far below the expected proportions of .167. That is, 3's and 4's occurred about 11 percent less often that if the dice had been perfectly formed, a difference that could make a big difference in a gambling game (Bulmer 1979, 18).

It is reasonable to think of both the *engineering* method (the theoretical approach) and the *empirical* method (experimentation and data collection) as two alternative ways to estimate a probability. The two methods use different processes and different proxies for the probability you wish to estimate. One must adduce additional knowledge to decide which method to use in any given situation. It is sensible to use the empirical method when data are available. (But use both together whenever possible.)

In view of the inevitably subjective nature of probability estimates, you may prefer to talk about "degrees of belief" instead of probabilities. That's fine, just as long as it is understood that we operate with degrees of belief in exactly the same way as we operate with probabilities. The two terms are working synonyms.

Most important: One cannot sensibly talk about probabilities in the abstract, without reference to some set of facts. The topic then loses its meaning, and invites confusion and argument. This also is a reason why a general formalization of the probability concept does not make sense.

3.11 The relationship of probability to the concept of resampling

There is no all-agreed definition of the concept of the resampling method in statistics. Unlike some other writers, I prefer to apply the term to problems in *both* pure probability and statistics. This set of examples may illustrate:

1. Consider asking about the number of hits one would expect from a 0.250 (25 percent) batter in a 400 at-bat season. One would call this a problem in "probability." The sampling distribution of the batter's results can be calculated by formula or produced by Monte Carlo simulation.

2. Now consider examining the number of hits in a given batter's season, and asking how likely that number (or fewer) is to occur by chance if the batter's long-run batting average is 0.250. One would call this a problem in "statistics." But just as in example (1) above, the answer can be calculated by formula or produced by Monte Carlo simulation. And the calculation or simulation is exactly the same as used in (1).

   Here the term "resampling" might be applied to the simulation with considerable agreement among people familiar with the term, but perhaps not by all such persons.

3. Next consider an observed distribution of distances that a batter's hits travel in a season with 100 hits, with an observed mean of 150 feet per hit. One might ask how likely it is that a sample of 10 hits drawn with replacement from the observed distribution of hit lengths (with a mean of 150 feet) would have a mean greater than 160 feet, and one could easily produce an answer with repeated Monte Carlo samples. Traditionally this would be called a problem in probability.

4. Next consider that a batter gets 10 hits with a mean of 160 feet, and one wishes to estimate the probability that the sample would be produced by a distribution as specified in (3). This is a problem in statistics, and by 1996, it is common statistical practice to treat it with a resampling method. The actual simulation would, however, be identical to the work described in (3).

Because the work in (4) and (2) differ only in question (4) involving measured data and question (2) involving counted data, there seems no reason to discriminate between the two cases with respect to the term "resampling." With respect to the pairs of cases (1) and (2), and (3) and (4), there is no difference in the actual work performed, though there is a difference in the way the question is framed. I would therefore urge that the label "resampling" be applied to (1) and (3) as well as to (2) and (4), to bring out the important fact that the procedure is the same as in resampling questions in statistics.

One could easily produce examples like (1) and (2) for cases that are similar except that the drawing is without replacement.[5] And one could adduce the example of prices in different

[5] One example of drawing *without replacement* is the sampling version of Ronald Fisher's permutation test — see (Fisher 1935; Fisher 1960, chap. II, section 5).

state liquor control systems (see Section 12.15) which is similar to cases (3) and (4) except that sampling without replacement seems appropriate. Again, the analogs to cases (2) and (4) would generally be called "resampling."

The concept of resampling is defined in a more precise way in Section 8.9.

3.12 Conclusion

We define "chance" as the absence of predictive power and/or explanation and/or control.

When the remote control rotates more than three or four turns I cannot control the outcome — whether TV or sofa end — with any accuracy. That is to say, I cannot predict much better than 50-50 with more than four rotations. So we then say that the outcome is determined by "chance."

As to those persons who wish to inquire into what the situation "really" is: I hope they agree that we do not need to do so to proceed with our work. I hope all will agree that the outcome of flipping the TV gradually *becomes* unpredictable (random) though still subject to similar physical processes as when predictable. I do not deny *in principle* that these processes can be "understood," certainly one can develop a machine (or a baton twirler) that will make the outcome predictable for many turns. But this has nothing to do with whether the mechanism is "really" something one wants to say is influenced by "chance." This is the point of the demonstration with the sofa and TV ends of the remote control. The outcome traverses from non-chance (determined) to chance (not determined) in a smooth way even though the physical mechanism that produces the revolutions remains much the same over the traverse.

4 Introducing Python and the Jupyter notebook

This chapter introduces you to the technology we will use throughout the book. By *technology*, we mean two things:

- The Python programming language, along with some important add-on libraries for data analysis.
- The Jupyter notebook system for running and editing Python code in a web interface.

> **Using Python on the web and on your computer**
>
> In this chapter, we concentrate on the technology we use for the interactive version of the book. The interactive version allows you to run Python code as interactive *notebooks* in your web browser.
> Either now, or later, you should also consider running the code on your own computer — see Section 4.10.

The chapter introduces Python and its packages, and then gives an example to introduce Python and the Jupyter Notebook. If you have not used Python before, the example notebook will get you started. The example also shows how we will be using notebooks through the rest of the book.

4.1 Python and its packages

This version of the book uses the *Python*[1] programming language to implement resampling algorithms.

Python is a programming language that can be used for many tasks. It is a popular language for teaching, but has also become standard in industry and academia. It is one of the most widely used programming languages in the world, and the most popular language for data science.

[1] https://www.python.org

For many of the initial examples, we will also be using the *Numpy*[2] package for Python. A Python *package* is a library of Python code and data. Numpy is a package that makes it easier to work with sequences of data values, such as sequences of numbers. These are typical in probability and statistics.

Later, we be using the *Matplotlib*[3] package. This is the main Python package with code for producing plots, such as bar charts, histograms, and scatter plots. See the rest of the book for more details on these plots.

Still further on in the book, we will use more specialized libraries for data manipulation and analysis. *Pandas*[4] is the standard Python package for loading data files and working with data tables. *Scipy*[5] is a package that houses a wide range of numerical routines, including some simple statistical methods. The *Statsmodels*[6] package has code for many more statistical procedures. We will find ourselves comparing the results of our own resampling algorithms to those in Scipy and Statsmodels.

It is very important that Python is a *programming language* and not a set of canned routines for "doing statistics". It means that we can explore the *ideas* of probability and statistics using the *language* of Python to express those ideas. It also means that you, and we, and anyone else in the world, can write new code to share with others, so they can benefit from our work, understand it, and improve it. This book is one example; we have written the Python code in this book as clearly as we can to make it easy to follow, and to explain the underlying ideas. We hope you will help us by testing what we have done and sending us suggestions for ways we could improve. Please see **?@sec-book-as-public** for more information about how to do that.

4.2 The environment

Many of the chapters have sections with code for you to run, and experiment with. These sections contain *Jupyter notebooks*[^jupyter-nb]. Jupyter notebooks are interactive web pages that allow you to read, write and run Python code. We mark the start of each notebook in the text with a note and link heading like the one you see below. In the web edition of this book, you can click on the *Download* link in this header to download the section as a notebook. You can also click on the *Interact* link in this header to open the notebook in your web browser, using a system called JupyterLite. You can run the code, and experiment by making changes.

[2] https://numpy.org
[3] https://matplotlib.org
[4] https://pandas.pydata.org
[5] https://scipy.org
[6] https://www.statsmodels.org

> **About JupyterLite**
>
> JupyterLite[7] is a version of the Jupyter notebook and Python that will automatically download and run inside your web browser.
> When you click on the "Interact" link, it will take you to a web address that has the effect of making your browser download the JupyterLite system, along with compatible versions of Python and its standard packages . The web page that opens allows you run to the Python code in the notebook inside your browser.

In the print version of the book, we point you to the web version, to get the links.

At the end of this chapter, we explain how to run these notebooks on your own computer. In the next section you will see an example notebook; to start with, you might want to run this in your browser using the "Interact" link.

4.3 Getting started with the notebook

The next section contains a notebook called "Billie's Bill". If you are looking at the web edition, you will see links to interact with this notebook in your browser, or download it to your computer.

> **Note 3: Notebook: Billie's Bill**
>
> - Download notebook
> - Interact

The text in this notebook section assumes you have opened the page as an interactive notebook on the web, or on your own computer (see Section 4.10).

A notebook can contain blocks of text — like this one — as well as code, and the results from running the code.

Jupyter Notebooks are made up of *cells*.

Jupyter cells can contain text or code.

Notebook text can have formatting, such as links.

For example, this sentence ends with a link to the earlier second edition of this book.

If you are in the interactive notebook interface (rather than reading this in the textbook), you will see the Jupyter menu bar near the top of the page, with headings "File", "Edit" and so on.

[7] https://jupyterlite.readthedocs.io

In Jupyter, underneath the File … menu bar, by default, you may see a row of icons - the "Toolbar".

In the Jupyter toolbar, you may see icons to run the current cell, among others.

To move from one cell to the next, you can click the run icon in the toolbar, but it is more efficient to press the Shift key, and press Enter (with Shift still held down). We will write this as Shift-Enter.

In this, our first notebook, we will be using Python to solve one of those difficult and troubling problems in life — working out the bill in a restaurant.

4.4 The meal in question

Alex and Billie are at a restaurant, getting ready to order. They do not have much money, so they are calculating the expected bill before they order.

Alex is thinking of having the fish for £10.50, and Billie is leaning towards the chicken, at £9.25. First they calculate their combined bill.

Below this text you see a *code* cell. It contains the Python code to calculate the total bill. Press Shift-Enter in the cell below, to see the total..

```
10.50 + 9.25
```

```
19.75
```

The contents of the cell above is Python code. As you would predict, Python understands numbers like `10.50`, and it understands `+` between the numbers as an instruction to add the numbers.

When you press Shift-Enter, Python finds `10.50`, realizes it is a number, and stores that number somewhere in memory. It does the same thing for `9.25`, and then it runs the *addition* operation on these two numbers in memory, which gives the number 19.75.

Finally, Python sends the resulting number (19.75) back to the notebook for display. The notebook detects that Python sent back a value, and shows it to us.

This is exactly what a calculator would do.

4.5 Comments

Unlike a calculator, we can also put notes next to our calculations, to remind us what they are for. One way of doing this is to use a "comment". You have already seen comments in the previous chapter.

A comment is some text that the computer will ignore. In Python, you can make a comment by starting a line with the # (hash) character. For example, the next cell is a code cell, but when you run it, it does not show any result. In this case, that is because the computer sees the # at the beginning of the line, and then ignores the rest.

```
# This bit of text is for us to read, and the computer to ignore.
```

Many of the code cells you see will have comments in them, to explain what the code is doing.

Practice writing comments for your own code. It is a very good habit to get into. You will find that experienced programmers write many comments on their code. They do not do this to show off, but because they have a lot of experience in reading code, and they know that comments make it much easier to read and understand code.

4.6 More calculations

Let us continue with the struggle that Alex and Billie are having with their bill.

They realize that they will also need to pay a tip.

They think it would be reasonable to leave a 15% tip. Now they need to multiply their total bill by 0.15, to get the tip. The bill is about £20, so they know that the tip will be about £3.

In Python * means multiplication. This is the equivalent of the "×" key on a calculator.

What about this, for the correct calculation?

```
# The tip - with a nasty mistake.
10.50 + 9.25 * 0.15
```

```
11.8875
```

Oh dear, no, that isn't doing the right calculation.

Python follows the normal rules of *precedence* with calculations. These rules tell us to do multiplication before addition.

See https://en.wikipedia.org/wiki/Order_of_operations for more detail on the standard rules.

In the case above the rules tell Python to first calculate `9.25 * 0.15` (to get `1.3875`) and then to add the result to `10.50`, giving `11.8875`.

We need to tell Python we want it to do the *addition* and *then* the multiplication. We do this with round brackets (parentheses):

> **Note 4: Three types of brackets in Python**
>
> There are three types of brackets in Python.
> These are:
>
> - *round brackets* or *parentheses*: `()`;
> - *square brackets*: `[]`;
> - *curly brackets*: `{}`.
>
> Each type of bracket has a different meaning in Python. In the examples, play close to attention to the type of brackets we are using.

```
# The bill plus tip - mistake fixed.
(10.50 + 9.25) * 0.15
```

```
2.9625
```

The obvious next step is to calculate the bill *including the tip*.

```
# The bill, including the tip
10.50 + 9.25 + (10.50 + 9.25) * 0.15
```

```
22.7125
```

At this stage we start to feel that we are doing too much typing. Notice that we had to type out `10.50 + 9.25` twice there. That is a little boring, but it also makes it easier to make mistakes. The more we have to type, the greater the chance we have to make a mistake.

4.7 Variables

To make things simpler, we would like to be able to *store* the result of the calculation `10.50 + 9.25`, and then re-use this value, to calculate the tip.

This is the role of *variables*. A *variable* is a value with a name.

Here is a variable:

```
# The cost of Alex's meal.
a = 10.50
```

`a` is a *name* we give to the value 10.50. You can read the line above as "The variable `a` *gets the value* 10.50". We can also talk of *setting* the variable. Here we are *setting* `a` to equal 10.50.

Now, when we use `a` in code, it refers to the value we gave it. For example, we can put `a` on a line on its own, and Python will show us the *value* of `a`:

```
# The value of a
a
```

```
10.5
```

We did not have to use the name `a` — we can choose almost any name we like. For example, we could have chosen `alex_meal` instead:

```
# The cost of Alex's meal.
# alex_meal gets the value 10.50
alex_meal = 10.50
```

We often set variables like this, and then display the result, all in the same cell. We do this by first setting the variable, as above, and then, on the final line of the cell, we put the variable name on a line on its own, to ask Python to show us the value of the variable. Here we set `billie_meal` to have the value 9.25, and then show the value of `billie_meal`, all in the same cell.

```
# The cost of Billie's meal.
billie_meal = 9.25
# Show the value of billies_meal
billie_meal
```

```
9.25
```

Of course, here, we did not learn much, but we often set variable values with the results of a calculation. For example:

```
# The cost of both meals, before tip.
bill_before_tip = 10.50 + 9.25
# Show the value of both meals.
bill_before_tip
```

```
19.75
```

But wait — we can do better than typing in the calculation like this. We can use the values of our variables, instead of typing in the values again.

```
# The cost of both meals, before tip, using variables.
bill_before_tip = alex_meal + billie_meal
# Show the value of both meals.
bill_before_tip
```

```
19.75
```

We make the calculation clearer by writing the calculation this way — we are calculating the bill before the tip by adding the cost of Alex's and Billie's meal — and that's what the code looks like. But this also allows us to *change* the variable value, and recalculate. For example, say Alex decided to go for the hummus plate, at £7.75. Now we can tell Python that we want `alex_meal` to have the value 7.75 instead of 10.50:

```
# The new cost of Alex's meal.
# alex_meal gets the value 7.75
alex_meal = 7.75
# Show the value of alex_meal
alex_meal
```

```
7.75
```

Notice that `alex_meal` now has a new value. It was 10.50, but now it is 7.75. We have *reset* the value of `alex_meal`. In order to use the new value for `alex_meal`, we must *recalculate* the bill before tip with *exactly the same code as before*:

```
# The new cost of both meals, before tip.
bill_before_tip = alex_meal + billie_meal
# Show the value of both meals.
bill_before_tip
```

```
17.0
```

Notice that, now we have rerun this calculation, we have *reset* the value for `bill_before_tip` to the correct value corresponding to the new value for `alex_meal`.

All that remains is to recalculate the bill plus tip, using the new value for the variable:

```
# The cost of both meals, after tip.
bill_after_tip = bill_before_tip + bill_before_tip * 0.15
# Show the value of both meals, after tip.
bill_after_tip
```

```
19.55
```

Now we are using variables with relevant names, the calculation looks right to our eye. The code expresses the calculation as we mean it: the bill after tip is equal to the bill before the tip, plus the bill before the tip times 0.15.

4.8 And so, on

Now you have done some practice with the notebook, and with variables, you are ready for a new problem in probability and statistics, in the next chapter.

4.9 Saving your work

If you are running this notebook via the "Interact" button, you are running it using the JupyterLite system. Please bear in mind that your browser keeps all the notebooks you run in JupyterLite, in its browser *cache* — a private and temporary store that the browser maintains somewhere on your system. If you want to keep any changes you make to notebooks you have run with the "Interact" JupyterLite system, you might want to save a copy of the notebook outside the browser cache. To do this, look the pane to the left of the notebook for the name of the notebook. This name of this particular notebook is "billies_bill", and you will see the notebook file in the left pane listed as `billies_bill.ipynb`. If you want

to save a copy to your computer, first use the "File" menu, and the "Save" option, to save your notebook. This saves the notebook to your browser's private store (the *cache*). Next *right-click* on `billies_bill.ipynb` in the left pane (see Figure 4.1), and choose "Download". Save the file somewhere memorable on your computer. You can go back to the notebook by following the instructions at Section 4.10.

> **End of notebook: Billie's Bill**
>
> `billies_bill` starts at Note 3.

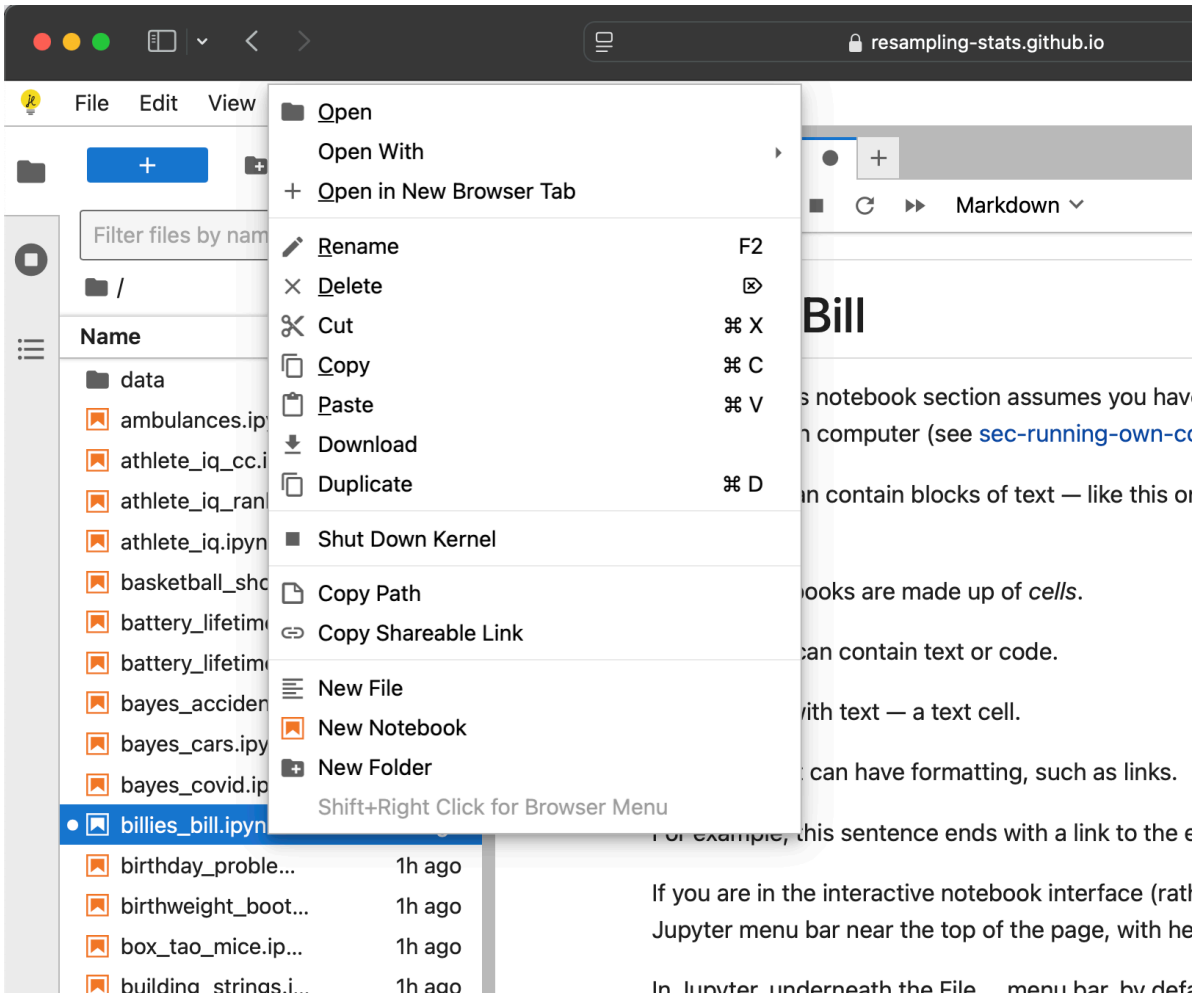


Figure 4.1: Downloading files in JupyterLite

4.10 Running the code on your own computer

Many people, including your humble authors, like to be able to run code examples on their own computers. This section explains how you can set up to run the notebooks on your own computer.

Once you have done this setup, you can use the "download" link that you will see for each notebook, to download the notebook to your machine. From there, you can open the notebook on Jupyter.

> **Note 5: Download links and data files**
>
> Most of the download links in this book will trigger a download of the notebook file. This is a file with extension `.ipynb`, that you can open with Jupyter.
> Later in the book, you will see examples where the notebook loads a data file. In that case, the download link for the notebook points to a `.zip` file containing the notebook and the data file. Unzip the `.zip` file to get the notebook and data file, and then open the resulting notebook in Jupyter.

You will need to install the Python language on your computer, and then install the following packages:

- Numpy — to work with arrays;
- Matplotlib - for plots;
- Scipy - a collection of modules for scientific computing;
- Pandas - for loading, saving and manipulating data tables;
- Statsmodels - for traditional statistical analysis;
- Jupyter - to run the Jupyter Notebook on your own computer.

One way to install Python and the packages you need, is to install Python from the Python website[8]. Then use the *Pip*[9] installer to install the packages you need.

To install the Python packages, first start a terminal application on your computer. To do this, you can use the Start key, "cmd" in Windows, or the Command key and space then "Terminal" on Mac. At the terminal prompt, type the following command:

Now you should be able to start the Jupyter notebook application. See the Jupyter documentation for how to start Jupyter. Open the notebook you downloaded for the chapter; you will now be able to run the code on your own computer, and experiment by making changes.

You can run any of the code notebooks in this textbook on your own machine by downloading the notebook, via the download link at the top of each notebook section, and then opening the resulting notebook in Jupyter.

[8] https://www.python.org
[9] https://pip.pypa.io

5 Resampling with code

Chapter 2 used simulation and resampling from tables of random numbers, dice, and coins. Making random choices in this way can make it easier to understand the process, but of course, physical methods of making random outcomes can be slow and boring.

We saw that short computer programs can do a huge number of resampling trials in a less than a second. The flexibility of a programming language makes it possible to simulate many different outcomes and tests.

Programs can build up tables of random numbers, and do basic tasks like counting the number of values in a row or taking proportions. With these simple tools, we can simulate many problems in probability and statistics.

In this chapter, we will model another problem using Python, but this chapter will add three new things.

- The problem we will work on is a little different from the ambulances problem from Chapter 2. It is a real problem about deciding whether a new cancer treatment is better than the alternatives, and it introduces the idea of making a model of the world, to ask questions about chances and probabilities.
- We will slow down a little to emphasize the steps in solving this kind of problem. First we work out how to simulate a single *trial*. Then we work out how to run many simulated trials.
- We sprinted through the code in Chapter 2, with the promise we would come back to the details. Here we go into more detail about some ideas from the code in the last chapter. These are:
  - Storing several values together in one place, with arrays.
  - Using *functions* (code recipes) to apply procedures.
  - *Comparing* numbers to other numbers.
  - *Counting* numbers that match a condition.

In the next chapter, we will talk more about *using arrays* to store results, and *for loops* to repeat a procedure many times.

5.1 Statistics and probability

We have already emphasized that *statistics* is a way of drawing conclusions about data from the real world, in the presence of random variation; *probability* is the way of reasoning about random variation. This chapter introduces our first *statistical* problem, where we use probability to draw conclusions about some important data — about a potential cure for a type of cancer. We will not make much of the distinction between probability and statistics here, but we will come back to it several times in later chapters.

5.2 A new treatment for Burkitt lymphoma

Burkitt lymphoma is an unusual cancer of the lymphatic system. The lymphatic system is a vein-like network throughout the body that is involved in the immune reaction to disease. In developed countries, with standard treatment, the cure rate for Burkitt lymphoma is about 90%.

In 2006, researchers at the US National Cancer Institute (NCI), tested a new treatment for Burkitt lymphoma (Dunleavy et al. 2006). They gave the new treatment to 17 patients, and found that all 17 patients were doing well after two years or more of follow up. By "doing well", we mean that their lymphoma had not progressed; as a short-hand, we will say that these patients were "cured", but of course, we do not know what happened to them after this follow up.

Here is where we put on our statistical hat and ask ourselves the following question — *how surprised are we that the NCI researchers saw their result of 17 out of 17 patients cured?*

At this stage you might and should ask, what could we possibly mean by "surprised"? That is a good and important question, and we will discuss that much more in the chapters to come. For now, please bear with us as we do a thought experiment.

Let us forget the 17 out of 17 result of the NCI study for a moment. Imagine that there is another hospital, called Saint Hypothetical General, just down the road from the NCI, that was also treating 17 patients with Burkitt lymphoma. Saint Hypothetical were not using the NCI treatment, they were using the standard treatment.

We already know that each patient given the standard treatment has a 90% chance of cure. Given that 90% cure rate, what is the chance that 17 out of 17 of the Hypothetical group will be cured?

You may notice that this question about the Hypothetical group is similar to the problem of the 16 ambulances in Chapter 2. In that problem, we were interested to know how likely it was that 3 or more of 16 ambulances would be out of action on any one day, given that each ambulance had a 10% chance of being out of action. Here we would like to know the chances

that all 17 patients would be cured, given that each patient has a 90% chance of being cured (and a 10% chance of relapse).

5.3 A physical model of the hypothetical hospital

As in the ambulance example, we could make a physical model of chance in this world. For example, to simulate whether a given patient is cured or not by a 90% effective treatment, we could throw a ten sided die and record the result. We could say, arbitrarily, that a result of 0 means "not cured", and all the numbers 1 through 9 mean "cured" (typical 10-sided dice have sides numbered 0 through 9).

We could roll 17 dice to simulate one "trial" in this random world. For each trial, we record the number of dice that show numbers 1 through 9 (and not 0). This will be a number between 0 and 17, and it is the number of patients "cured" in our simulated trial.

Figure 5.1 is the result of one such trial we did with a set of 17 10-sided dice we happened to have to hand:

The trial in Figure 5.1 shows are four dice with the 0 face uppermost, and the rest with numbers from 1 through 9. Therefore, there were 13 out of 17 not-zero numbers, meaning that 13 out of 17 simulated "patients" were "cured" in this simulated trial.

We could repeat this simulated trial procedure 100 times, and we would then have 100 counts of the not-zero numbers. Each of the 100 counts would be the number of patients cured in that trial. We can ask how many of these 100 counts were equal to 17. This will give us an estimate of the probability we would see 17 out of 17 patients cured, given that any one patient has a 90% chance of cure. For example, say we saw 15 out of 100 counts were equal to 17. That would give us an estimate of 15 / 100 or 0.15 or 15%, for the probability we would see 17 out of 17 patients cured.

So, if Saint Hypothetical General did see 17 out of 17 patients cured with the standard treatment, they would be a little surprised, because they would only expect to see that happen 15% of the time. But they would not be *very* surprised — 15% of the time is uncommon, but not *very* uncommon.

5.4 A trial, a run, a count and a proportion

Here we stop to emphasize the steps in the process of a random simulation.

1. *We decide what we mean by one trial.* Here one trial has the same meaning in medicine as resampling — we mean the result of treating 17 patients. One *simulated trial* is then the simulation of one set of outcomes from 17 patients.

Figure 5.1: One roll of 17 10-sided dice

2. Work out the *outcome* of interest from the trial. The outcome here is the number of patients cured.
3. We work out a way to *simulate one trial*. Here we chose to throw 17 10-sided dice, and count the number of not zero values. This is the outcome from one simulation trial.
4. We *repeat* the simulated trial procedure many times, and collect the results from each trial. Say we repeat the trial procedure 100 times; we will call this a *run* of 100 trials.
5. We *count* the number of trials with an outcome that matches the outcome we are interested in. In this case we are interested in the outcome 17 out of 17 cured, so we count the number of trials with a score of 17. Say 15 out of the run of 100 trials had an outcome of 17 cured. That is our *count*.
6. Finally we divide the count by the number of trials to get the *proportion*. From the example above, we divide 15 by 100 to 0.15 (15%). This is our estimate of the chance of seeing 17 out of 17 patients cured in any one trial. We can also call this an estimate of the *probability* that 17 out of 17 patients will be cured on any on trial.

Our next step is to work out the code for step 2: *simulate one trial*.

5.5 Simulate one trial with code

We can use the computer to do something very similar to rolling 17 10-sided dice, by asking the computer for 17 random whole numbers from 0 through 9.

> **i** Whole numbers
>
> A whole number is a number that is not negative, and does not have fractional part (does not have anything after a decimal point). 0 and 1 and 2 and 3 are whole numbers, but -1 and $\frac{3}{5}$ and 11.3 are not. The whole numbers from 0 through 9 are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

We have already discussed what we mean by *random* in Section 2.2.

We will be asking the computer to generate many random numbers. So, before we start, we again import Numpy and get its *random number generator*:

```
import numpy as np

# Ask for Numpy's default random number generator and name
# it `rnd`.  `rnd` is short for "random".
rnd = np.random.default_rng()
```

5.6 From numbers to arrays

We next need to prepare the *sequence* of numbers that we want NumPy to select from.

We have already seen the idea that Python has *values* that are individual numbers. Remember, a *variable* is a *named value*. Here we attach the name `a` to the value 1.

```
a = 1
# Show the value of "a"
a
```

```
1
```

NumPy also allows *values* that are *sequences of numbers*. NumPy calls these sequences *arrays*.

Here we make a array that contains the 10 numbers we will select from:

```
# Make an array of numbers, store with the name "some_numbers".
some_numbers = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Show the value of "some_numbers"
some_numbers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Notice that the value for `some_numbers` is an array, and that this value *contains* 10 numbers.

Put another way, `some_numbers` is now the name we can use for this collection of 10 values.

Arrays are very useful for simulations and data analysis, and we will be using these for nearly every example in this book.

5.7 Functions

Functions are another tool that we will be using everywhere, and that you seen already, although we have not introduced them until now.

You can think of functions as named *production lines*.

For example, consider the Python *function* `np.round`

```
# We load the Numpy library so we have access to the Numpy functions.
import numpy as np
```

`np.round` is the name for a simple production line, that takes in a number, and (by default) sends back the number rounded to the nearest *integer*.

> **i** What is an integer?
>
> An *integer* is a positive or negative *whole number*.
> In other words, a number is an *integer* if the number is *either* a whole number (0, 1, 2 …), *or* a negative whole number (-1, -2, -3 …). All of -208, -2, 0, 10, 105 are integers, but $\frac{3}{5}$, -10.3 and 0.2 are not.
> We will use the term *integer* fairly often, because it is a convenient way to name all the positive and negative whole numbers.

Think of a function as a named *production line*. We sent the function (production line) raw material (components) to work on. The production line does some work on the components. A finished result comes off the other end.

Therefore, think of `np.round` as the name of a production line, that takes in a *component* (in this case, any number), and does some work, and sends back the finished *result* (in this case, the number rounded to the nearest integer.

The components we send to a function are called *arguments*. The finished result the function sends back is the *return value*.

- **Arguments**: the value or values we send to a function.
- **Return value**: the values the function sends back.

See Figure 5.2 for an illustration of `np.round` as a production line.

In the next few code cells, you see examples where `np.round` takes in a not-integer number, as an *argument*, and sends back the nearest integer as the *return value*:

```
# Put in 3.2, round sends back 3.
np.round(3.2)
```

```
np.float64(3.0)
```

```
# Put in -2.7, round sends back -3.
np.round(-2.7)
```

```
np.float64(-3.0)
```

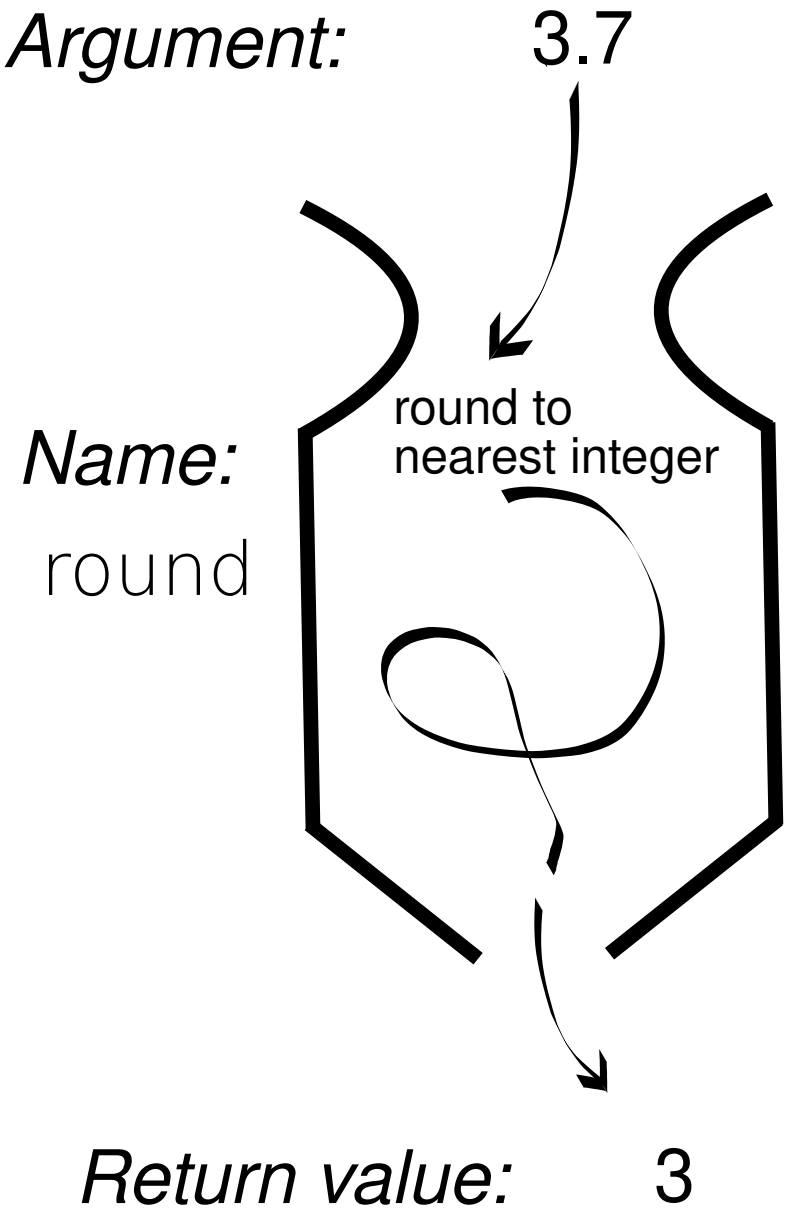


Figure 5.2: The `round` function as a production line

Like many functions, `np.round` can take more than one argument (component). You can send `round` the number of digits you want to round to, after the number of you want it to work on, like this (see Figure 5.3):

```
# Put in 3.1415, and the number of digits to round to (2).
# round sends back 3.14
np.round(3.1415, 2)
```

```
np.float64(3.14)
```

Notice that the second argument — here 2 — is *optional*. We only have to send `round` one argument: the number we want it to round. But we can *optionally* send it a second argument — the number of decimal places we want it to round to. If we don't specify the second argument, then `round` assumes we want to round to 0 decimal places, and therefore, to the nearest integer.

5.8 Functions and named arguments

In the example above, we sent `round` two arguments. `round` knows that we mean the first argument to be the number we want to round, and the second argument is the number of decimal places we want to round to. It knows which is which by the *position* of the arguments — the *first* argument is the *number* it should round, and *second* is the number of digits.

In fact, internally, the `round` function also gives these arguments *names*. It calls the number it should round — `a` — and the number of digits it should round to — `decimals`. This is useful, because it is often clearer and simpler to identify the argument we are specifying with its name, instead of just relying on its position.

If we aren't using the argument names, we call the round function as we did above:

```
# Put in 3.1415, and the number of digits to round to (2).
# round sends back 3.14
np.round(3.1415, 2)
```

```
np.float64(3.14)
```

In this call, we relied on the fact that we, the people writing the code, and you, the person reading the code, remembers that the second argument (2) means the number of decimal places it should round to. But, we can also specify the argument using its name, like this (see Figure 5.5):

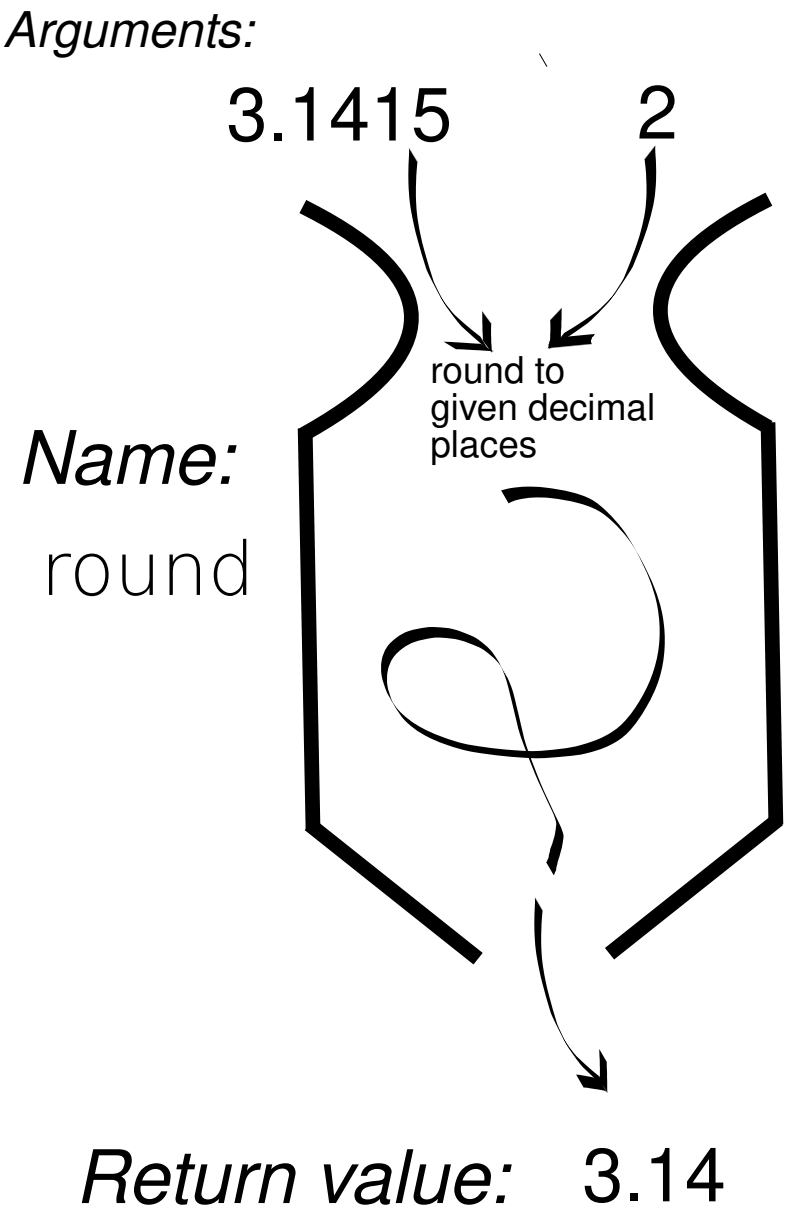


Figure 5.3: `round` with optional arguments specifying number of digits

```
# Put in 3.1415, and the number of digits to round to (2).
# Use the name of the number-of-decimals argument for clarity:
np.round(3.1415, decimals=2)
```

```
np.float64(3.14)
```

Here Python sees the *first* argument, as before, and assumes that it is the number we want to round. Then it sees the second, named argument — `decimals=2` — and knows, *from the name*, that we mean this to be the number of decimals to round to.

In fact, we could even specify *both* arguments by name, like this:

```
# Put in 3.1415, and the number of digits to round to (2).
np.round(a=3.1415, decimals=2)
```

```
np.float64(3.14)
```

We don't usually name both arguments for `round`, as we have above, because it is so obvious that the first argument is the thing we want to round, and so naming the argument does not make it any more clear what the code is doing. But — as so often in programming — whether to use the names, or let Python work out which argument is which by position, is a judgment call. The judgment you are making is about the way to write the code to be most clear for your reader, where your most important reader may be you, coming back to the code in a week or a year.

> **i** How do you know what names to use for the function arguments?
>
> You can find the names of the function arguments in the help for the function, either online, or in the notebook interface. For example, to get the help for `np.round`, including the argument names, you could make a new cell, and type `np.round?`, then execute the cell by pressing Shift-Enter. This will show the help for the function in the notebook interface.

5.9 Ranges

Now let us return to the variable `some_numbers` that we created above:

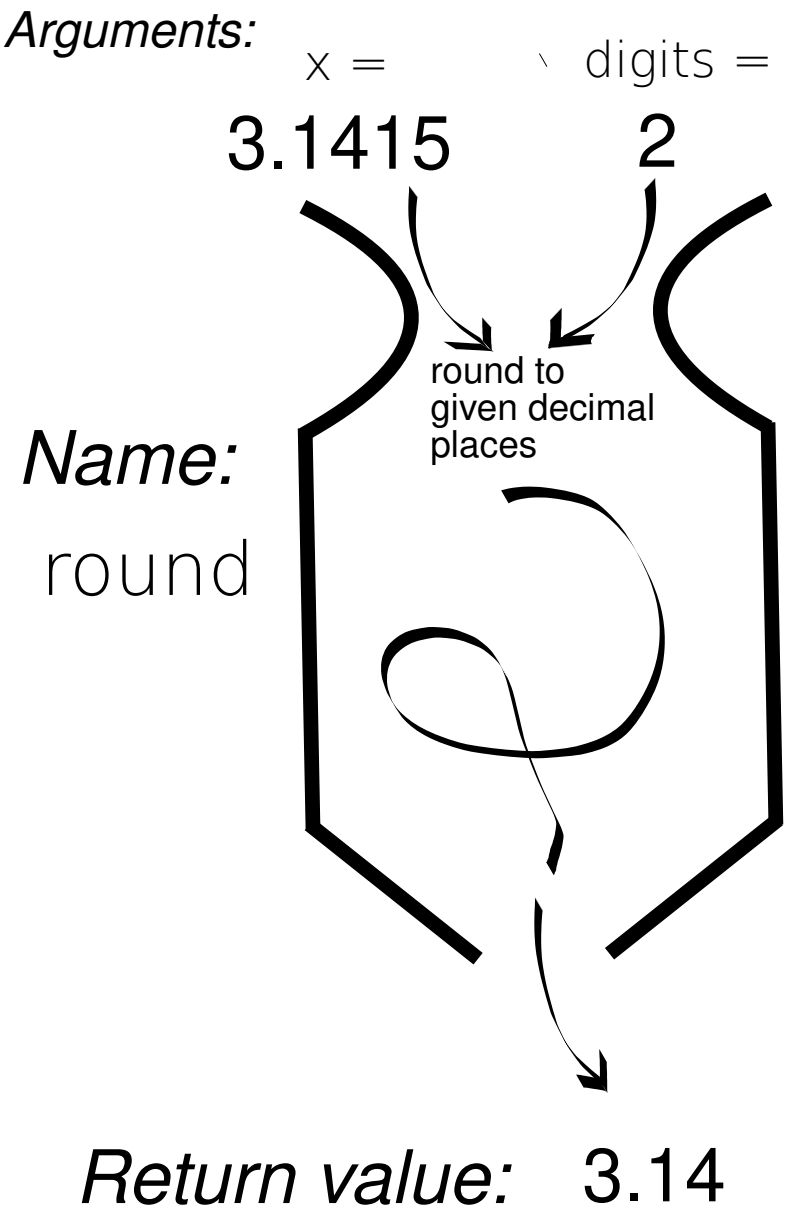


Figure 5.4: The `round` function with argument names

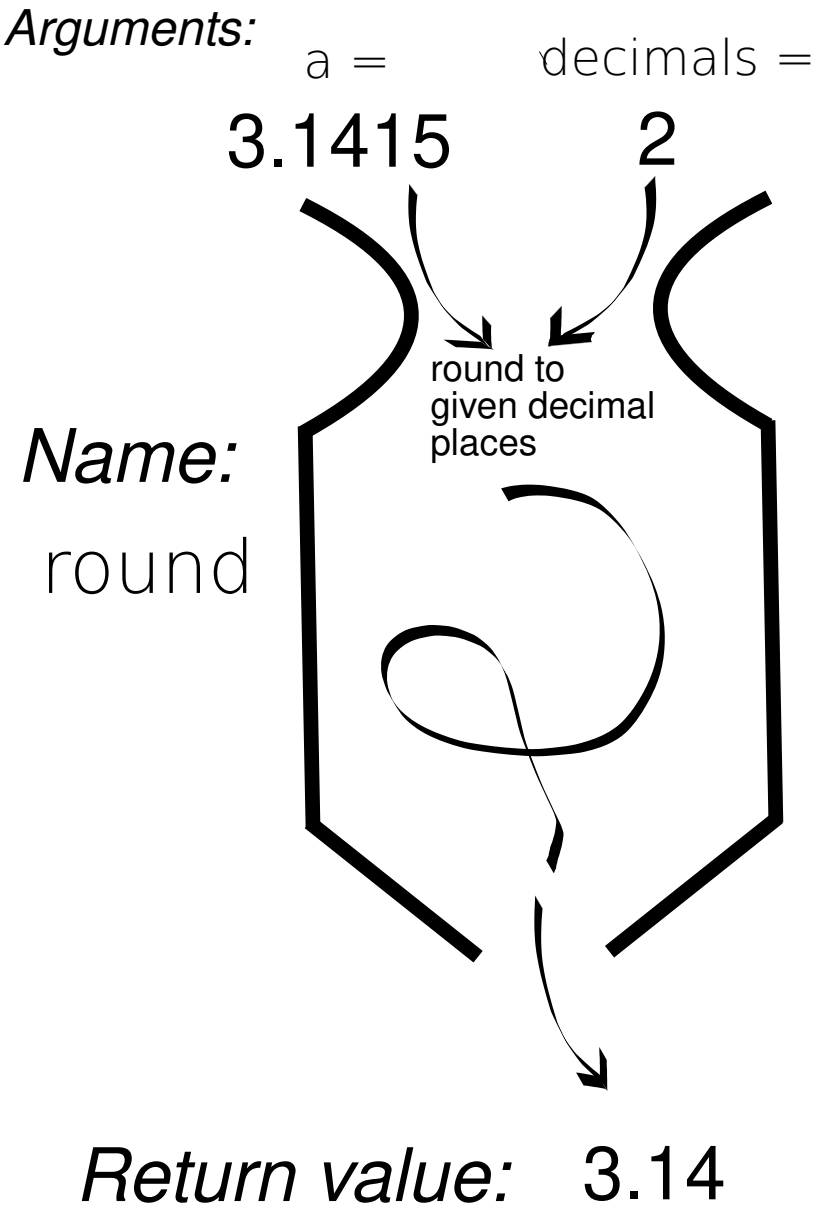


Figure 5.5: The `np.round` function with argument names

```
# Make an array of numbers, store with the name "some_numbers".
some_numbers = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Show the value of "some_numbers"
some_numbers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In fact, we often need to do this: generate a sequence or *range* of integers, such as 0 through 9.

> **Pick a number from 1 through 5**
>
> Ranges can be confusing in normal speech because it is not always clear whether they include their beginning and end. For example, if someone says "pick a number between 1 and 5", do they mean to pick from *all* of the numbers, including the first and last (any of 1 or 2 or 3 or 4 or 5)? Or do they mean only the numbers that are *between* 1 and 5 (so 2 or 3 or 4)? Or do they mean all the numbers up to, but not including 5 (so 1 or 2 or 3 or 4)?
> To avoid this confusion, we will nearly always use "from" and "through" in ranges, meaning that we do include both the start and the end number. For example, if we say "pick a number from 1 through 5" we mean one of 1 or 2 or 3 or 4 or 5.

Creating ranges of numbers is so common that Python has a standard Numpy function `np.arange` to do that.

```
# An array containing all the numbers from 0 through 9.
some_numbers = np.arange(0, 10)
some_numbers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Notice that we send `np.arange` the *arguments* 0 and 10. The first argument, here 0, is the *start* value. The second argument, here 10, is the *stop* value. Numpy (in the `arange` function) understands this to mean: *start* at 0 (the start value) and go up to *but do not include* 10 (the stop value).

You can therefore read `np.arange(0, 10)` as "the sequence of integers starting at 0, up to, but not including 10".

Like `np.round`, the arguments to `np.arange` also have names, so, we could also write:

```
# An array containing all the numbers from 0 through 9.
# Now using named arguments.
some_numbers = np.arange(start=0, stop=10)
some_numbers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

So far, we have sent `arange` two arguments, but we can also send just one argument, like this:

```
# An array containing all the numbers from 0 through 9.
some_integers = np.arange(10)
some_integers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

When we sent `arange` a single argument, like this, `arange` understands this to mean we have sent just the *stop* value, and that is should assume a *start* value of 0.

Again, if we wanted, we could send this argument by name:

```
# An array containing all the numbers from 0 through 9.
# Specify the stop value by explicit name, for clarity.
some_integers = np.arange(stop=10)
some_integers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Here are some more examples of `np.arange`:

```
# All the integers starting at 10, up to, but not including 15.
# In other words, 10 through 14.
np.arange(10, 15)
```

```
array([10, 11, 12, 13, 14])
```

```
# Here we are only sending one value (7). np.arange understands this to be
# the stop value, and assumes 0 as the start value.
# In other words, 0 through 6
np.arange(7)
```

```
array([0, 1, 2, 3, 4, 5, 6])
```

5.10 `range` in Python

So far you have seen ranges of integers using `np.arange`. The `np.` prefix refers to the fact that `np.arange` is a function from the Numpy module (library). The `a` in `arange` signals that the result `np.arange` returns is an *array*:

```
arr = np.arange(7)
# Show the result
arr
```

```
array([0, 1, 2, 3, 4, 5, 6])
```

```
# Show what type of thing this is.
type(arr)
```

```
<class 'numpy.ndarray'>
```

We do often use `np.arange` to get a range of integers in a convenient array format, but Python has another way of getting a range of integers — the `range` function.

The `range` function is very similar to `np.arange`, but it is *not* part of Numpy — it is basic function in Python — and it does *not* return an *array* of numbers, it returns something else. Here we ask for a `range` from 0 through 6 (0 up to, but not including 7):

```
# Notice no `np.` before `range`.
r = range(7)
r
```

```
range(0, 7)
```

Notice that the thing that came back is something that *represents* or *stands in for* the number 0 through 6. It is not an array, but a specific type of thing called — a `range`:

```
type(r)
```

```
<class 'range'>
```

The `range` above is a container for the numbers 0 through 6. We can get the numbers out of the container in many different ways, but one of them is to convert this container to an array, using the `np.array` function. The `np.array` function takes the thing we pass it, and makes it into an array. When we apply `np.array` to `r` above, we get the numbers that `r` contains:

```
# Get the numbers from the range `r`, convert to an array.
a_from_r = np.array(r)
# Show the result
a_from_r
```

```
array([0, 1, 2, 3, 4, 5, 6])
```

The `range` function has the same `start` and `stop` arguments that `np.arange` does, and with the same meaning:

```
# 3 up to, not including 12.
# (3 through 11)
r_2 = range(3, 12)
r_2
```

```
range(3, 12)
```

```
np.array(r_2)
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11])
```

You may reasonably ask — why do I need this `range` thing, if I have the very similar `np.arange`? The answer is — you don't *need* `range`, and you can always use `np.arange` where you would use `range`, but for reasons we will go into later (Section 6.6.3), `range` is a good option when we want to represent a sequence of numbers as input to a `for` loop. We cover `for` loops in more detail in Section 6.6.2, but for now, the only thing to remember is that `range` and `np.arange` are both ways of expressing sequential ranges of integers.

5.11 Choosing values at random

We can use the `rnd.choice` function to select a single value *at random* from the sequence of numbers in `some_integers`.

> **More on `rnd.choice`**
>
> The `rnd.choice` function will be a fundamental tool for taking many kinds of samples, and we cover it in more detail in Chapter 7.

```
# Select an integer from the choices in some_integers.
my_integer = rnd.choice(some_integers)
# Show the value that results.
my_integer
```

```
np.int64(5)
```

Like `np.round` (above), `rnd.choice` is a *function*.

> **Note 6: Functions and methods**
>
> Actually, to be precise, we should call `rnd.choice` a *method*. A method is a *function attached to a value*. In this case the function `choice` is attached to the value `rnd`. That's not an important distinction for us at the moment, so please forgive our strategic imprecision, and let us continue to say that `rnd.choice` is a function.

As you remember, a function is a named *production line*. In our case, the production line has the name `rnd.choice`.

We sent `rnd.choice`. a value to work on — an *argument*. In this case, the argument was the value of `some_integers`.

Figure 5.6 is a diagram illustrating an example run of the `rnd.choice` function (production line).

Here is the same code again, with new comments.

```
# Send the value of "some_integers" to rnd.choice
# some_integers is the *argument*.
# Put the *return* value from the function into "my_number".
my_number = rnd.choice(some_integers)
# Show the value that results.
my_number
```

```
np.int64(4)
```

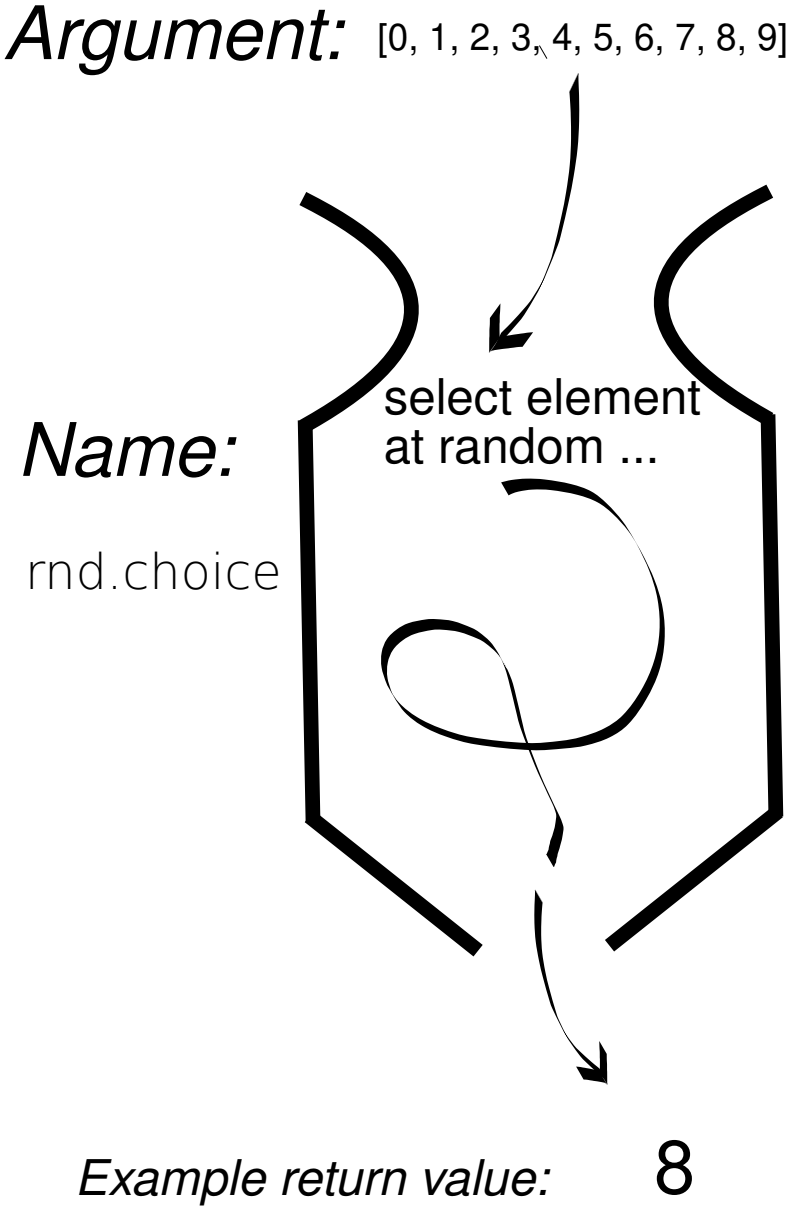


Figure 5.6: Example run of the `rnd.choice` function

5.12 Creating arrays with sampling

In the code above, we asked Python to select a single number at random — because that is what `rnd.choice` does by default.

In fact, the people who wrote `rnd.choice`, wrote it to be flexible in the work that it can do. In particular, we can tell `rnd.choice` to select *any number of values* at random, by adding a new *argument* to the function.

In our case, we would like Numpy to select 17 numbers at random from the sequence of `some_integers`.

To do this, we add an argument to the function that tells it *how many* numbers we want it to select.

```
# Get 17 values from the *some_integers* array.
# Store the 17 numbers with the name "a"
a = rnd.choice(some_integers, 17)
# Show the result.
a
```

```
array([4, 5, 9, 8, 2, 9, 1, 5, 8, 2, 1, 8, 2, 6, 6, 5, 0])
```

As you can see, the function sent back (returned) 17 numbers. Because it is sending back more than one number, the thing it sends back is an array, where the array has 17 elements.

5.12.1 `sum` — adding all the values

Bear with us for a short diversion. You will see why we made this diversion soon.

NumPy has a function `np.sum` that will add up all the numbers in an array.

You can see the contents of `a` above.

`np.sum` adds all the numbers in the array together, to give the *sum* of the array. The *sum* is just the result of adding all the values in the array. Put another way, it is the result of adding the second element to the first, then adding third element to the result, and the fourth element to the result, and so on.

```
np.sum(a)
```

```
np.int64(81)
```

5.13 Counting results

We now have the code to do the equivalent of throwing 17 ten-sided dice. This is the basis for one simulated trial in the world of Saint Hypothetical General.

Our next job is to get the code to count the number of numbers that are not zero in the array `a`. That will give us the number of patients who were cured in simulated trial.

Another way of asking this question, is to ask how many elements in `a` are greater than zero.

5.13.1 Comparison

To ask whether a number is greater than zero, we use *comparison*. Here is a *greater than zero* comparison on a single number:

```
n = 5
# Is the value of n greater than 0?
# Show the result of the comparison.
n > 0
```

```
True
```

`>` is a *comparison* — it *asks a question* about the numbers either side of it. In this case `>` is asking the question "is the value of `n` (on the left hand side) greater than 0 (on the right hand side)?" The value of `n` is 5, so the question becomes, "is 5 greater than 0?" The answer is Yes, and Python represents this Yes answer as the value `True`.

In contrast, the comparison below boils down to "is 0 greater than 0?", to which the answer is No, and Python represents this as `False`.

```
p = 0
# Is the value of p greater than 0?
# Show the result of the comparison.
p > 0
```

```
False
```

So far you have seen the results of comparison on a single number. Now say we do the same comparison on an array. For example, say we ask the question "is the value of `a` greater than 0"? Remember, `a` is an array containing 17 values. We are comparing 17 values to one value (0). What answer do you think NumPy will give? You may want to think a little about this before you read on.

As a reminder, here is the current value for `a`:

```
# Show the current value for "a"
a
```

```
array([4, 5, 9, 8, 2, 9, 1, 5, 8, 2, 1, 8, 2, 6, 6, 5, 0])
```

Now you have had some time to think, here is what happens:

```
# Is the value of "a" greater than 0
# Show the result of the comparison.
a > 0
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False])
```

There are 17 values in `a`, so the comparison to 0 means there are 17 comparisons, and 17 answers. NumPy therefore returns *an array* of 17 elements, containing these 17 answers. The first answer is the answer to the question "is the value of the *first* element of `a` greater than 0", and the second is the answer to "is the value of the *second* element of `a` greater than 0".

Let us store the result of this comparison to work on:

```
# Is the value of "a" greater than 0
# Store as another array "q".
q = a > 0
# Show the value of r
q
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False])
```

5.14 Counting `True` values with `sum`

Notice above that there is one `True` element in `q` for every element in `a` that was greater than 0. It only remains to *count* the number of `True` values in `q`, to get the count of patients in our simulated trial who were cured.

We can use the NumPy function `np.sum` to count the number of `True` elements in an array. As you can see above, `np.sum` adds up all the elements in an array, to give a single number. This will work as we want for the `q` array, because Python counts `False` as equal to 0 and `True` as equal to 1:

```
# Question: is False equal to 0?
# Answer - Yes! (True)
False == 0
```

```
True
```

```
# Question: is True equal to 1?
# Answer - Yes! (True)
True == 1
```

```
True
```

Therefore, the function `sum`, when applied to an array of `True` and `False` values, will count the number of `True` values in the array.

To see this in action we can make a new array of `True` and `False` values, and try using `np.sum` on the new array.

```
# An array containing three True values and two False values.
trues_and_falses = np.array([True, False, True, True, False])
# Show the new array.
trues_and_falses
```

```
array([ True, False,  True,  True, False])
```

The `sum` operation adds all the elements in the array. Because `True` counts as 1, and `False` counts as 0, adding all the elements in `trues_and_falses` is the same as adding up the values $1 + 0 + 1 + 1 + 0$, to give 3.

We can apply the same operation on `q` to count the number of `True` values.

```
# Count the number of True values in "q"
# This is the same as the number of values in "a" that are greater than 0.
b = np.sum(q)
# Show the result
b
```

```
np.int64(16)
```

5.15 The procedure for one simulated trial

We now have the whole procedure for one simulated trial. We can put the whole procedure in one cell:

```
# Procedure for one simulated trial

# Get 17 values from the *some_integers* array.
# Store the 17 numbers with the name "a"
a = rnd.choice(some_integers, 17)
# Is the value of "a" greater than 0
q = a > 0
# Count the number of True values in "q"
b = np.sum(q)
# Show the result of this simulated trial.
b
```

```
np.int64(17)
```

5.16 Repeating the trial

Now we know how to do one simulated trial, we could just keep running the cell above, and writing down the result each time. Once we had run the cell 100 times, we would have 100 counts. Then we could look at the 100 counts to see how many were equal to 17 (all 17 simulated patients cured on that trial). At least that would be much faster than rolling 17 dice 100 times, but we would also like the computer to automate the process of repeating the trial, and keeping track of the counts.

Please forgive us as we race ahead again, as we did in the last chapter. As in the last chapter, we will use a *results* array called `z` to store the count for each trial. As in the last chapter, we will use a `for` loop to repeat the trial procedure many times. As in the last chapter, we will not explain the counts array of the `for` loop in any detail, because we are going to cover those in the next chapter.

Let us now imagine that we want to do 100 simulated trials at Saint Hypothetical General. This will give us 100 counts. We will want to store the count for each trial.

To do this, we make an array called `z` to hold the 100 counts. We have called the array `z`, but we could have called it anything we liked, such as `counts` or `results` or `cecilia`.

```
# An array to hold the 100 count values.
# Later, we will fill this in with real count values from simulated trials.
z = np.zeros(100)
```

Next we use a `for` loop to *repeat the single trial procedure.*

Notice that the single trial procedure, inside this `for` loop, is the same as the single trial procedure above — the only two differences are:

- The trial procedure is inside the loop, and
- We are storing the count for each trial as we go.

We will go into more detail on how this works in the next chapter.

```
# Procedure for 100 simulated trials.

# An array to store the counts for each trial.
z = np.zeros(100)

# Repeat the trial procedure 100 times.
for i in np.arange(100):
    # Get 17 values from the *some_integers* array.
    # Store the 17 numbers with the name "a".
    a = rnd.choice(some_integers, 17)
    # Is the value of "a" greater than 0.
    q = a > 0
    # Count the number of True values in "q".
    b = np.sum(q)
    # Store the result at the next position in the "z" array.
    z[i] = b
    # Now go back and do the next trial until finished.
# Show the result of all 100 trials.
z
```

```
array([16., 15., 15., 16., 16., 12., 15., 11., 16., 13., 12., 16., 15.,
       16., 15., 16., 14., 15., 14., 15., 15., 15., 14., 15., 17., 15.,
       14., 15., 16., 17., 15., 17., 16., 17., 14., 16., 15., 15., 15.,
       17., 17., 13., 16., 13., 16., 14., 14., 15., 15., 15., 14., 15.,
       15., 15., 17., 16., 17., 14., 15., 14., 16., 16., 15., 15., 16.,
       15., 15., 16., 17., 15., 17., 15., 10., 15., 15., 14., 14., 13.,
       16., 14., 17., 17., 16., 14., 15., 16., 17., 14., 15., 15., 16.,
       16., 17., 16., 13., 15., 15., 14., 17., 15.])
```

Finally, we need to count how many of the trials results we stored in `z` gave a "cured" count of 17.

We can ask the question whether a single number is equal to 17 using the *double equals* comparison: `==`.

```
s = 17
# Is the value of s equal to 17?
# Show the result of the comparison.
s == 17
```

```
True
```

> Note
>
> ## 5.17 Single and double equals
>
> Notice that the *double equals* `==` means something entirely different to Python than the single equals `=`. In the code above, Python reads `s = 17` to mean "Set the variable `s` to have the value 17". In technical terms the single equals is called an *assignment* operator, because it means *assign* the value 17 to the variable `s`.
> The code `s == 17` has a completely different meaning.
> It means "give `True` if the value in `s` is equal to 17, and `False` otherwise". The `==` is a *comparison* operator — it is for comparing two values — here the value in `s` and the value 17. This comparison, like all comparisons, returns an answer that is either `True` or `False`. In our case `s` has the value 17, so the comparison becomes `17 == 17`, meaning "is 17 equal to 17?", to which the answer is "Yes", and Python sends back `True`.

We can ask this question of *all 100 counts* by asking the question: is the array `z` equal to 17, like this:

```
# Is the value of z equal to 17?
were_cured = z == 17
# Show the result of the comparison.
were_cured
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False, False,  True, False,  True, False,  True, False, False,
       False, False, False,  True,  True, False, False, False, False,
```

```
       False, False, False, False, False, False, False, False, False,
        True, False,  True, False, False, False, False, False, False,
       False, False, False, False, False,  True, False,  True, False,
       False, False, False, False, False, False, False, False,  True,
        True, False, False, False, False,  True, False, False, False,
       False, False,  True, False, False, False, False, False,  True,
       False])
```

Finally we use `sum` to count the number of `True` values in the `were_cured` array, to give the number of trials where all 17 patients were cured.

```
# Count the number of True values in "were_cured"
# This is the same as the number of values in "z" that are equal to 17.
n_all_cured = np.sum(were_cured)
# Show the result of the comparison.
n_all_cured
```

```
np.int64(15)
```

`n_all_cured` is the number of simulated trials for which all patients were cured. It only remains to get the proportion of trials for which this was true, and to do this, we divide by the number of trials.

```
# Proportion of trials where all patients were cured.
p = n_all_cured / 100
# Show the result
p
```

```
np.float64(0.15)
```

From this experiment, we see that there is roughly a one-in-six chance that all 17 patients are cured when using a 90% effective treatment.

5.18 What have we learned from Saint Hypothetical?

We started with a question about the results of the NCI trial on the new drug. The question was — was the result of their trial — 17 out of 17 patients cured — surprising.

Then, for reasons we did not explain in detail, we changed tack, and asked the same question about a hypothetical set of 17 patients getting the *standard* treatment in Saint Hypothetical General.

That Hypothetical question turns out to be fairly easy to answer, because we can use simulation to estimate the chances that 17 out of 17 patients would be cured in such a hypothetical trial, on the assumption that each patient has a 90% chance of being cured with the standard treatment.

The answer for Saint Hypothetical General was — we would be somewhat surprised, but not astonished. We only get 17 out of 17 patients cured about one time in six.

Now let us return to the NCI trial. Should the trial authors be surprised by their results? If they assumed that their new treatment was *exactly as effective* as the standard treatment, the result of the trial is a bit unusual, just by chance. It is up to us to decide whether the result is unusual enough to make us think that the actual NCI treatment might in fact have been *more effective* than the standard treatment.

You will see this move again and again as we go through the book.

- We take something that really happened — in this case the 17 out of 17 patients cured.
- Then we imagine a hypothetical world in which the results only depend on chance.
- We do simulations in that hypothetical world to see how often we get a result like the one that happened in the real world.
- If the real world result (17 out of 17) is an unusual, surprising result in the simulations from the hypothetical world, we take that as evidence that the real world result might not be due to chance alone.

We have just described the main idea in statistical inference. If that all seems strange and backwards to you, do not worry, we will go over that idea many times in this book. It is not a simple idea to grasp in one go. We hope you will find that, as you do more simulations, and think of more hypothetical worlds, the idea will start to make more sense. Later, we will start to think about asking *other questions* about probability and chance in the real world.

5.19 Conclusions

Can you see how each of the operations that the computer carries out are analogous to the operations that you yourself executed when you solved this problem using 10-sided dice? This is exactly the procedure that we will use to solve every problem in probability and statistics that we must deal with. Either we will use a device such as coins or dice, or a random number table as an analogy for the physical process we are interested in (patients being cured, in this case), or we will simulate the analogy on the computer using the Python program above.

The program above may not seem simple at first glance, but we think you will find, over the course of this book, that these programs become much simpler to understand than the older conventional approach to such problems that has routinely been taught to students for decades.

6 More resampling with code

Chapter 5 introduced a problem in probability, that was also a problem in statistics. We asked how surprised we should be at the results of a trial of a new cancer treatment regime.

Here we study another urgent problem in the real world - racial bias and the death penalty.

6.1 A question of life and death

This example comes from the excellent Berkeley introduction to data science (Ani Adhikari and Wagner 2021).

Robert Swain was a young black man who was sentenced to death in the early 60s. Swain's trial was held in Talladega County, Alabama. At the time, 26% of the eligible jurors in that county were black, but every member of Swain's jury was white. Swain and his legal team appealed to the Alabama Supreme Court, and then to the US Supreme Court, arguing that there was racial bias in the jury selection. They noted that there had been no black jurors in Talladega county since 1950, even though they made up about a quarter of the eligible pool of jurors. The US Supreme Court rejected this argument, in a 6 to 3 opinion, writing that "The overall percentage disparity has been small and reflects no studied attempt to include or exclude a specified number of Negros.".

Swain's team presented a variety of evidence on bias in jury selection, but here we will look at the obvious and apparently surprising fact that Swain's jury was entirely white. The Supreme Court decided that the "disparity" between selection of white and black jurors "has been small" — but how would they, and how would we, make a rational decision about whether this disparity *really* was "small"?

You might reasonably be worried about the result of this decision for Robert Swain. In fact his death sentence was invalidated by a later, unrelated decision and he served a long prison sentence instead. In 1986, the Supreme Court overturned the precedent set by Swain's case, in Batson v. Kentucky, 476 U.S. 79.

6.2 A small disparity and a hypothetical world

To answer the question that the Supreme Court asked, we return to the method we used in the last chapter.

Let us imagine a hypothetical world, in which each individual black or white person had an equal chance of being selected for the jury. Call this world Hypothetical County, Alabama.

Just as in 1960's Talladega County, 26% of eligible jurors in Hypothetical County are black. Hypothetical County jury selection has no bias against black people, so we expect around 26% of the jury to be black. 0.26 * 12 = 3.12, so we expect that, on average, just over 3 out of 12 jurors in a Hypothetical County jury will be black. But, if we select each juror at random from the population, that means that, sometimes, by chance, we will have fewer than 3 black jurors, and sometimes will have more than 3 black jurors. And, by chance, sometimes we will have no black jurors. But, if the jurors really are selected at random, how often would we expect this to happen — that there are no black jurors? We would like to estimate the *probability* that we will get no black jurors. If that probability is small, then we have some evidence that the disparity in selection between black and white jurors, was not "small".

What is the probability of an *all white* jury being randomly selected out of a population having 26% black people?

6.3 Designing the experiment

Before we start, we need to figure out three things:

1. What do we mean by one trial?
2. What is the outcome of interest from the trial?
3. How do we simulate one trial?

We then take **three steps** to calculate the desired probability:

1. *Repeat* the simulated trial procedure N times.
2. *Count* M, the number of trials with an outcome that matches the outcome we are interested in.
3. Calculate the *proportion*, M/N. This is an estimate of the probability in question.

For this problem, our task is made a little easier by the fact that our *trial* (in the resampling sense) is a simulated *trial* (in the legal sense). One trial requires 12 simulated jurors, each labeled by race (white or black).

The outcome we are interested in is the number of black jurors.

Now comes the harder part. How do we simulate one trial?

6.3.1 One trial

One trial requires 12 jurors, and we are interested only in the race of each juror. In Hypothetical County, where selection by race is entirely random, each juror has a 26% chance of being black.

We need a way of simulating a 26% chance.

One way of doing this is by getting a random number from 0 through 99 (inclusive). There are 100 numbers in the range 0 through 99 (inclusive).

We will arbitrarily say that the juror is white if the random number is in the range from 0 through 73. 74 of the 100 numbers are in this range, so the juror has a 74/100 = 74% chance of getting the label "white". We will say the juror is black if the random number is in the range 74 though 99. There are 26 such numbers, so the juror has a 26% chance of getting the label "black".

Next we need a way of getting a random number in the range 0 through 99. This is an easy job for the computer, but if we had to do this with a physical device, we could get a single number by throwing *two* 10-sided dice, say a blue die and a green die. The face of the blue die will be the 10s digit, and the green face will be the ones digit. So, if the blue die comes up with 8 and the green die has 4, then the random number is 84.

We could then simulate 12 jurors by repeating this process 12 times, each time writing down "white" if the number is from 0 through 74, and "black" otherwise. The trial outcome is the number of times we wrote "black" for these 12 simulated jurors.

6.3.2 Using code to simulate a trial

We use the same logic to simulate a trial with the computer. A little code makes the job easier, because we can ask Python to give us 12 random numbers from 0 through 99, and to count how many of these numbers are in the range from 75 through 99. Numbers in the range from 75 through 99 correspond to black jurors.

6.3.3 Random numbers from 0 through 99

We can now use NumPy and the random number functions from the last chapter to get 12 random numbers from 0 through 99.

```
# Import the Numpy library, rename as "np"
import numpy as np

# Ask Numpy for a random number generator.
rnd = np.random.default_rng()
```

```
# All the integers from 0 up to, but not including 100.
zero_thru_99 = np.arange(100)

# Get 12 random numbers from 0 through 99
a = rnd.choice(zero_thru_99, size=12)

# Show the result
a
```

```
array([59, 43, 45, 58, 95, 89, 23, 99, 17, 51, 85, 23])
```

6.3.3.1 Counting the jurors

We use *comparison* and `np.sum` to count how many numbers are greater than 74, and therefore, in the range from 75 through 99:

```
# How many numbers are greater than 74?
b = np.sum(a > 74)
# Show the result
b
```

```
np.int64(4)
```

6.3.3.2 A single simulated trial

We assemble the pieces from the last few sections to make a cell that simulates a single trial:

```
rnd = np.random.default_rng()
zero_thru_99 = np.arange(100)

# Get 12 random numbers from 0 through 99
a = rnd.choice(zero_thru_99, size=12)

# How many numbers are greater than 74?
b = np.sum(a > 74)

# Show the result
b
```

```
np.int64(4)
```

6.4 Three simulation steps

Now we come back to the details of how we:

1. Repeat the simulated trial many times;
2. record the results for each trial;
3. calculate the required proportion as an estimate of the probability we seek.

Repeating the trial many times is the job of the `for` loop, and we will come to that soon.

In order to record the results, we will store each trial result in an array.

6.4.1 More on arrays

Since we will be working with arrays a lot, it is worth knowing more about them.

A Numpy array is a *container* that stores many elements of the same type. You have already seen, in Chapter 2, how we can create an array from a sequence of numbers using the `np.array` function.

```
# Make an array of numbers, store with the name "some_numbers".
some_numbers = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Show the value of "some_numbers"
some_numbers
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Another way that we can create arrays is to use the `np.zeros` function to make a new array where all the elements are 0.

```
# Make a new array containing 5 zeros.
# store with the name "z".
z = np.zeros(5)
# Show the value of "z"
z
```

```
array([0., 0., 0., 0., 0.])
```

Notice the argument `5` to the `np.zeros` function. This tells the function how many zeros we want in the array that the function will return.

6.5 Array length

The are various useful things we can do with this array container. One is to ask how many elements there are in the array container. We can use the `len` function to calculate the number of elements in an array:

```
# Show the number of elements in "z"
len(z)
```

```
5
```

6.6 Indexing into arrays with integers

Another thing we can do with arrays is *set* the value for a particular element. To do this, we use square brackets following the array value, on the left hand side of the equals sign, like this:

```
# Set the value of the *first* element in the array.
z[0] = 99
# Show the new contents of the array.
z
```

```
array([99.,  0.,  0.,  0.,  0.])
```

Read the first line of code as "the element at position 0 gets a value of 99".

Notice that the position number of the first element in the array is 0, and the position number of the second element is 1. Think of the position as an *offset* from the beginning of the array. The first element is at the beginning of the array, and so it is at offset (position) 0. This can be a little difficult to get used to at first, but you will find that thinking of the positions of offsets in this way soon starts to come naturally, and later you will also find that it helps you to avoid some common mistakes when using positions for getting and setting values.

For practice, let us also set the value of the third element in the array:

```
# Set the value of the *third* element in the array.
z[2] = 99
# Show the new contents of the array.
z
```

```
array([99.,  0., 99.,  0.,  0.])
```

Read the first code line above as as "set the value at position 2 in the array to have the value 99".

We can also *get* the value of the element at a given position, using the same square-bracket notation:

```
# Get the value of the *first* element in the array.
# Store the value with name "v"
v = z[0]
# Show the value we got
v
```

```
np.float64(99.0)
```

Read the first code line here as "v gets the value at position 0 in the array".

Using square brackets to get and set element values is called *indexing* into the array.

6.6.1 Repeating trials

As a preview, let us now imagine that we want to do 50 simulated trials of Robert Swain's jury in Hypothetical County. We will want to store the count for each trial, to give 50 counts.

In order to do this, we make an array to hold the 50 counts. Call this array `z`.

```
# An array to hold the 50 count values.
z = np.zeros(50)
```

We could run a single trial to get a single simulated count. Here we just repeat the code cell you saw above. Notice that we can get a different result each time we run this code, because the numbers in `a` are *random* choices from the range 0 through 99, and different random numbers will give different counts.

```
rnd = np.random.default_rng()
zero_thru_99 = np.arange(100)
# Get 12 random numbers from 0 through 99
a = rnd.choice(zero_thru_99, size=12)
# How many numbers are greater than 74?
b = np.sum(a > 74)
# Show the result
b
```

```
np.int64(4)
```

Now we have the result of a single trial, we can store it as the first number in the `z` array:

```
# Store the single trial count as the first value in the "z" array.
z[0] = b
# Show all the values in the "z" array.
z
```

```
array([4., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Of course we could just keep doing this: run the cell corresponding to a trial, above, to get a new count, and then store it at the next position in the `z` array. For example, we could store the counts for the first three trials with:

```
# First trial
a = rnd.choice(zero_thru_99, size=12)
b = np.sum(a > 74)
# Store the result at the first position in z
# Remember, the first position is offset 0.
z[0] = b
# Second trial
a = rnd.choice(zero_thru_99, size=12)
b = np.sum(a > 74)
# Store the result at the second position in z
z[1] = b
# Third trial
a = rnd.choice(zero_thru_99, size=12)
b = np.sum(a > 74)
# Store the result at the third position in z
z[2] = b

# And so on ...
```

This would get terribly long and boring to type for 50 trials. Luckily computer code is very good at repeating the same procedure many times. For example, Python can do this using a `for` loop. You have already seen a preview of the `for` loop in Chapter 2. Here we dive into `for` loops in more depth.

6.6.2 For-loops in Python

A for-loop is a way of asking Python to:

- Take a sequence of things, one by one, and
- Do the same task on each one.

We often use this idea when we are trying to explain a repeating procedure. For example, imagine we wanted to explain what the supermarket checkout person does for the items in your shopping basket. You might say that they do this:

> For each item of shopping in your basket, they take the item off the conveyor belt, scan it, and put it on the other side of the till.

You could also break this description up into bullet points with indentation, to say the same thing:

- For each item from your shopping basket, they:
  - Take the item off the conveyor belt.
  - Scan the item.
  - Put it on the other side of the till.

Notice the logic; the checkout person is repeating the same procedure for each of a series of items.

This is the logic of the `for` loop in Python. The procedure that Python repeats is called the *body of the for loop*. In the example of the checkout person above, the repeating procedure is:

- Take the item off the conveyor belt.
- Scan the item.
- Put it on the other side of the till.

Now imagine we wanted to use Python to print out the year of birth for each of the authors for the third edition of this book:

Author	Year of birth
Julian Lincoln Simon	1932
Matthew Brett	1964
Stéfan van der Walt	1980

We want to see this output:

```
Author birth year is 1932
Author birth year is 1964
Author birth year is 1980
```

Of course, we could just ask Python to print out these exact lines, like this:

```
print('Author birth year is 1932')
```

```
Author birth year is 1932
```

```
print('Author birth year is 1964')
```

```
Author birth year is 1964
```

```
print('Author birth year is 1980')
```

```
Author birth year is 1980
```

We might instead notice that we are repeating the same procedure for each of the three birth years, and decide to do the same thing using a `for` loop:

```
author_birth_years = np.array([1932, 1964, 1980])

# For each birth year
for birth_year in author_birth_years:
    # Repeat this procedure ...
    print('Author birth year is', birth_year)
```

```
Author birth year is 1932
Author birth year is 1964
Author birth year is 1980
```

The `for` loop starts with a line where we tell it what items we want to repeat the procedure for:

```
for birth_year in author_birth_years:
```

This *initial line* of the `for` loop ends with a colon.

The next thing in the `for` loop is the procedure Python should follow for each item. Python knows that the following lines are the procedure it should repeat, because the lines are *indented*. The *indented* lines are the *body of the for loop*.

The initial line of the `for` loop above tells Python that it should take *each item* in `author_birth_years`, one by one — first 1932, then 1964, then 1980. For each of these numbers it will:

- Put the number into the variable `birth_year`, then
- Run the indented code .

Just as the person at the supermarket checkout takes each item in turn, for each iteration (repeat) of the `for` loop, `birth_year` gets a new value from the sequence in `author_birth_years`. `birth_year` is called the *loop variable*, because it is the variable that gets a new value each time we begin a new iteration of the `for` loop procedure. As for any variable in Python, we can call our loop variable anything we like. We used `birth_year` here, but we could have used `y` or `year` or some other name.

Now you know what the `for` loop is doing, you can see that the `for` loop above is equivalent to the following code:

```
birth_year = 1932  # Set the loop variable to contain the first value.
print('Author birth year is', birth_year)  # Use it.
```

```
Author birth year is 1932
```

```
birth_year = 1964  # Set the loop variable to contain the next value.
print('Author birth year is', birth_year)  # Use the second value.
```

```
Author birth year is 1964
```

```
birth_year = 1980
print('Author birth year is', birth_year)
```

```
Author birth year is 1980
```

Writing the steps in the `for` loop out like this is called *unrolling* the loop. It can be a useful exercise to do this when you come across a `for` loop, in order to work through the logic of the loop. For example, you may want to write out the unrolled equivalent of the first couple

of iterations, to see what the loop variable will be, and what will happen in the body of the loop.

We often use `for` loops with ranges (see Section 5.9). Here we use a loop to print out the numbers 0 through 3:

```
for n in np.arange(4):
    print('The loop variable n is', n)
```

```
The loop variable n is 0
The loop variable n is 1
The loop variable n is 2
The loop variable n is 3
```

Notice that the range ended at (the number before) 4, and that means we repeat the loop body 4 times. We can also use the loop variable value from the range as an *index*, to get or set the first, second, etc values from an array.

For example, maybe we would like to show the author position *and* the author year of birth.

Remember our author birth years:

```
author_birth_years
```

```
array([1932, 1964, 1980])
```

We can get (for example) the second author birth year with:

```
author_birth_years[1]
```

```
np.int64(1964)
```

Remember, for Python, the first element is position 0, so the second element is position 1.

Using the combination of looping over a range, and array indexing, we can print out the author position *and* the author birth year:

```
for n in np.arange(3):
    year = author_birth_years[n]
    print('Birth year of author position', n, 'is', year)
```

```
Birth year of author position 0 is 1932
Birth year of author position 1 is 1964
Birth year of author position 2 is 1980
```

Again, remember Python considers 0 as the first position.

Just for practice, let us unroll the three iterations through this `for` loop, to remind ourselves what the code is doing:

```
# Unrolling the for loop.
n = 0
year = author_birth_years[n]  # Will be 1932
print('Birth year of author position', n, 'is', year)
```

```
Birth year of author position 0 is 1932
```

```
n = 1
year = author_birth_years[n]  # Will be 1964
print('Birth year of author position', n, 'is', year)
```

```
Birth year of author position 1 is 1964
```

```
n = 2
year = author_birth_years[n]  # Will be 1980
print('Birth year of author position', n, 'is', year)
```

```
Birth year of author position 2 is 1980
```

6.6.3 `range` in Python `for` loops

So far we have used `np.arange` to give us the sequence of integers that we feed into the `for` loop. But — as you saw in Section 5.10 — we can also get a range of numbers from Python's `range` function. `range` is a common and useful alternative way to provide a range of numbers to a `for` loop.

You have just seen how we would use `np.arange` to send the numbers 0, 1, 2, and 3 to a `for` loop, in the example above, repeated here:

```
for n in np.arange(3):
    year = author_birth_years[n]
    print('Birth year of author position', n, 'is', year)
```

```
Birth year of author position 0 is 1932
Birth year of author position 1 is 1964
Birth year of author position 2 is 1980
```

We could also use `range` instead of `np.arange` to do the same task:

```
for n in range(3):
    year = author_birth_years[n]
    print('Birth year of author position', n, 'is', year)
```

```
Birth year of author position 0 is 1932
Birth year of author position 1 is 1964
Birth year of author position 2 is 1980
```

In fact, you will see this pattern throughout the book, where we use `for` statements like `for value in range(10000):` to ask Python to put each number in the range 0 up to (not including) 10000 into the variable `value`, and then do something in the body of the loop. Just to be clear, we could always, and almost as easily, write `for value in np.arange(10000):` to do the same task. However, we generally prefer `range` in our Python `for` loops, because it is just a little less typing (without the `np.a` of `np.arange`), and because it is a more common pattern in standard Python code.[1]

6.6.4 Putting it all together

Here is the code we worked out above, to implement a single trial:

```
rnd = np.random.default_rng()
zero_thru_99 = np.arange(100)
# Get 12 random numbers from 0 through 99
a = rnd.choice(zero_thru_99, size=12)
# How many numbers are greater than 74?
b = np.sum(a > 74)
```

[1] Actually, there is a reason why many Python programmers prefer `range` to `np.arange` in the headers for their `for` loops. `range` is a very efficient container, in that it doesn't need to take up all the memory required to create the full array, it just needs to keep track of the number to give you next. For example, consider `for i in np.arange(10000000):` — in this case Python has to make an array with 10,000,000 elements, and then, from that array, it passes each value one by one to the `for` loop. On the other hand, `for i in range(10000000):` will do the job just as well, passing the same sequence of 0 through 9,999,999 to `i`, one by one, but `range(10000000)` never has to make the whole 10,000,000 element array — it just needs to keep track of which number to give up next. Therefore `range` is very quick, and very efficient in memory. This doesn't have any great practical impact for the arrays we are using here, typically of 10,0000 elements or so, but it can be important for larger arrays.

```
# Show the result
b
```

```
np.int64(4)
```

We found that we could use arrays to store the results of these trials, and that we could use `for` loops to repeat the same procedure many times.

Now we can put these parts together to do 50 simulated trials:

```
# Procedure for 50 simulated trials.

# The Numpy random number generator.
rnd = np.random.default_rng()

# All the numbers from 0 through 99.
zero_through_99 = np.arange(100)

# An array to store the counts for each trial.
z = np.zeros(50)

# Repeat the trial procedure 50 times.
for i in np.arange(50):
    # Get 12 random numbers from 0 through 99
    a = rnd.choice(zero_through_99, size=12)
    # How many numbers are greater than 74?
    b = np.sum(a > 74)
    # Store the result at the next position in the "z" array.
    z[i] = b
    # Now go back and do the next trial until finished.
# Show the result of all 50 trials.
z
```

```
array([4., 2., 3., 3., 4., 1., 4., 2., 7., 2., 3., 1., 6., 2., 5., 5., 3.,
       1., 3., 4., 2., 2., 2., 4., 3., 4., 4., 2., 3., 3., 3., 1., 3., 1.,
       2., 3., 2., 2., 3., 3., 6., 1., 3., 3., 4., 2., 4., 3., 4., 3.])
```

Finally, we need to count how many of the trials in `z` ended up with all-white juries. These are the trials with a `z` (count) value of 0.

To do this, we can ask an array which elements match a certain condition. E.g.:

```
x = np.array([2, 1, 3, 0])
y = x < 2
# Show the result
y
```

```
array([False,  True, False,  True])
```

We now use that same technique to ask, of *each of the 50 counts*, whether the array `z` is equal to 0, like this:

```
# Is the value of z equal to 0?
all_white = z == 0
# Show the result of the comparison.
all_white
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False])
```

We need to get the number of `True` values in `all_white`, to find how many simulated trials gave all-white juries.

```
# Count the number of True values in "all_white"
# This is the same as the number of values in "z" that are equal to 0.
n_all_white = np.sum(all_white)
# Show the result of the comparison.
n_all_white
```

```
np.int64(0)
```

`n_all_white` is the number of simulated trials for which all the jury members were white. It only remains to get the proportion of trials for which this was true, and to do this, we divide by the number of trials.

```
# Proportion of trials where all jury members were white.
p = n_all_white / 50
# Show the result
p
```

```
np.float64(0.0)
```

From this initial simulation, it seems there is around a 0% chance that a jury selected randomly from the population, which was 26% black, would have no black jurors.

6.7 Many many trials

Our experiment above is only 50 simulated trials. The higher the number of trials, the more confident we can be of our estimate for `p` — the proportion of trials where we get an all-white jury.

It is no extra trouble for us to tell the computer to do a very large number of trials. For example, we might want to run 10,000 trials instead of 50. All we have to do is to run the loop 10,000 times instead of 50 times. The computer has to do more work, but it is more than up to the job.

Here is exactly the same code we ran above, but collected into one cell, and using 10,000 trials instead of 50. We have left out the comments, to make the code more compact.

```
# Full simulation procedure, with 10,000 trials.
rnd = np.random.default_rng()
zero_through_99 = np.arange(100)
# 10,000 trials.
z = np.zeros(10000)
for i in np.arange(10000):
    a = rnd.choice(zero_through_99, size=12)
    b = np.sum(a > 74)
    z[i] = b
all_white = z == 0
n_all_white = sum(all_white)
p = n_all_white / 10000
p
```

```
np.float64(0.0305)
```

We now have a new, more accurate estimate of the proportion of Hypothetical County juries that are all white. The proportion is 0.03, and so 3%.

This proportion means that, for any one jury from Hypothetical County, there is a less than one in 20 chance that the jury would be all white.

As we will see in more detail later, we might consider using the results from this experiment in Hypothetical County, to reflect on the result we saw in the real Talladega County. We might conclude, for example, that there was likely some systematic difference between Hypothetical County and Talledega County. Maybe the difference was that there was, in fact, some bias in the jury selection in Talledega county, and that the Supreme Court was wrong to reject this. You will hear more of this line of reasoning later in the book.

6.8 Conclusion

In this chapter we studied a real life-and-death question, on racial bias and the death penalty. We continued our exploration of the ways we can use probability, and resampling, to draw conclusions about real events. Along the way, we went into more detail on arrays in Python, and `for` loops; two basic tools in resampling.

In the next chapter, we will work through some more problems in probability, to show how we can use resampling, to answer questions about chance. We will add some more tools for writing code in Python, to make your programs easier to write, read, and understand.

7 Tools for samples and sampling

7.1 Introduction

Now you have some experience with Python, probabilities and resampling, it is time to introduce some useful tools for our experiments and programs.

> Note 7: Notebook: Sampling tools
>
> - Download notebook
> - Interact

7.2 Samples and labels and strings

Thus far we have used numbers such as 1 and 0 and 10 to represent the elements we are sampling from. For example, in Chapter 6, we were simulating the chance of a particular juror being black, given that 26% of the eligible jurors in the county were black. We used *integers* for that task, where we started with all the integers from 0 through 99, and asked NumPy to select values at random from those integers. When NumPy selected an integer from 0 through 25, we chose to label the resulting simulated juror as black — there are 26 integers in the range 0 through 25, so there is a 26% chance that any one integer will be in that range. If the integer was from 26 through 99, the simulated juror was white (there are 74 integers in the range 26 through 99).

Here is the process of simulating a single juror, adapted from Section 6.3.3:

```
import numpy as np
# Ask Numpy for a random number generator.
rnd = np.random.default_rng()

# All the integers from 0 up to, but not including 100.
zero_thru_99 = np.arange(100)

# Get one random numbers from 0 through 99
a = rnd.choice(zero_thru_99)
```

```
# Show the result
a
```

```
np.int64(59)
```

After that, we have to unpack our labeling of 0 through 25 as being "black" and 26 through 99 as being "white". We might do that like this:

```
this_juror_is_black = a < 26
this_juror_is_black
```

```
np.False_
```

This all works as we want it to, but it's just a little bit difficult to remember the coding (less than 26 means "black", greater than 25 means "white"). We had to use that coding because we committed ourselves to using random numbers to simulate the outcomes.

However, Python can also store bits of text, called *strings*. Values that are bits of text can be very useful because the text values can be memorable labels for the entities we are sampling from, in our simulations.

Before we get to strings, let us consider the different types of value we have seen so far.

7.3 Types of values in Python

You have already come across the idea that Python values can be integers (positive or negative whole numbers), like this:

```
v = 10
v
```

```
10
```

Here the variable `v` holds the value. We can see what type of value `v` holds by using the `type` function:

```
type(v)
```

```
<class 'int'>
```

Python can also have *floating point* values. These are values with a decimal point — numbers that do not have to be integers, but also can be any value between the integers. These floating point values are of type `float`:

```
f = 10.1
type(f)
```

```
<class 'float'>
```

7.3.1 Numpy arrays

You have also seen that Numpy contains another type, the *array*. An array is a value that contains a sequence of values. For example, here is an array of integers:

```
arr = np.array([0, 10, 99, 4])
arr
```

```
array([ 0, 10, 99,  4])
```

Notice that this value `arr` is of type `np.ndarray`:

```
type(arr)
```

```
<class 'numpy.ndarray'>
```

The array has its own internal record of what type of values it holds. This is called the array `dtype`:

```
arr.dtype
```

```
dtype('int64')
```

The array `dtype` records the type of value stored in the array. All values in the array must be of this type, and all values in the array are therefore of the same type.

The array above contains integers, but we can also make arrays containing floating point values:

```
float_arr = np.array([0.1, 10.1, 99.0, 4.3])
float_arr
```

```
array([ 0.1, 10.1, 99. ,  4.3])
```

```
float_arr.dtype
```

```
dtype('float64')
```

7.3.2 Lists

We have elided past another Python type, the *list*. In fact we have already used lists in making arrays. For example, here we make an array with four values:

```
np.array([0, 10, 99, 4])
```

```
array([ 0, 10, 99,  4])
```

We could also write the statement above in two steps:

```
my_list = [0, 10, 99, 4]
np.array(my_list)
```

```
array([ 0, 10, 99,  4])
```

In the first statement — `my_list = [0, 10, 99, 4]` — we construct a *list* — a container for the four values. Let's look at the `my_list` value:

```
my_list
```

```
[0, 10, 99, 4]
```

Notice that we do not see `array` in the display — this is not an array but a list:

```
type(my_list)
```

```
<class 'list'>
```

A list is a basic Python type. We can construct it by using the square brackets notation that you see above; we start with `[`, then we put the values we want to go in the list, separated by commas, followed by `]`. Here is another list:

```
# Creating another list.
list_2 = [5, 10, 20]
```

As you saw, we have been building arrays by building lists, and then passing the list to the `np.array` function, to create an array.

```
list_again = [100, 10, 0]
np.array(list_again)
```

```
array([100,  10,   0])
```

Of course, we can do this one line, as we have been doing up till now, by constructing the list inside the parentheses of the function. So, the following cell has just the same output as the cell above:

```
# Constructing the list inside the function brackets.
np.array([100, 10, 0])
```

```
array([100,  10,   0])
```

Lists are like arrays in that they are values that contain values, but they are unlike arrays in various ways — that we will not go into now. We often use lists to construct sequences into lists to turn them into arrays. For our purposes, and particularly for our calculations, arrays are much more useful and efficient than lists.

7.4 String values

So far, all the values you have seen in Python arrays have been numbers. Now we get on to values that are bits of text. These are called *strings*.

Here is a single Python string value:

```
s = "Resampling"
s
```

```
'Resampling'
```

What is the `type` of the new bit-of-text value `s`?

```
type(s)
```

```
<class 'str'>
```

The Python `str` value is a bit of text, and therefore consists of a sequence of characters.

As arrays are containers for other things, such as numbers, strings are containers for characters.

As we can find the number of elements in an array (Section 6.5), we can find the number of characters in a string with the `len` function:

```
# Number of characters in s
len(s)
```

```
10
```

As we can *index* into array values to get individual elements (Section 6.6), we can index into string values to get individual characters:

```
# Get the second character of the string
# Remember, Python's index positions start at 0.
second_char = s[1]
second_char
```

```
'e'
```

7.5 Strings in arrays

As we can store numbers as elements in arrays, we can also store strings as array elements.

```
# Just for clarity, make the list first.
# Lists can also contain strings.
list_of_strings = ['Julian', 'Lincoln', 'Simon']
# Then pass the list to np.array to make the array.
arr_of_strings = np.array(list_of_strings)
arr_of_strings
```

```
array(['Julian', 'Lincoln', 'Simon'], dtype='<U7')
```

```
# We can also create the list and the array in one line,
# as we have been doing up til now.
arr_of_strings = np.array(['Julian', 'Lincoln', 'Simon'])
arr_of_strings
```

```
array(['Julian', 'Lincoln', 'Simon'], dtype='<U7')
```

Notice the array `dtype`:

```
arr_of_strings.dtype
```

```
dtype('<U7')
```

The `U` in the `dtype` tells you that the elements in the array are Unicode strings (Unicode is a computer representation of text characters). The number after the `U` gives the maximum number of characters for any string in the array, here set to the length of the longest string when we created the array.

Take care with Numpy string arrays

It is easy to run into trouble with Numpy string arrays where the elements have a maximum length, as here. Remember, the `dtype` of the array tells you what type of element the array can hold. Here the `dtype` is telling you that the array can hold strings of maximum length 7 characters. Now imagine trying to put a longer string into the array — what do you think would happen?
This happens:

```
# An array of small strings.
small_strings = np.array(['six', 'one', 'two'])
small_strings.dtype
```

```
dtype('<U3')
```

```
# Set a new value for the first element (first string).
small_strings[0] = 'seven'
small_strings
```

```
array(['sev', 'one', 'two'], dtype='<U3')
```

Numpy truncates the new string to match the original maximum length.
For that reason, it is often useful to instruct Numpy that you want to use effectively infinite length strings, by specifying the array `dtype` as `object` *when you make the array*, like this:

```
# An array of small strings, but this time, tell Numpy
# that the strings should be of effectively infinite length.
small_strings_better = np.array(['six', 'one', 'two'], dtype=object)
small_strings_better
```

```
array(['six', 'one', 'two'], dtype=object)
```

Notice that the code uses a *named function argument* (Section 5.8), to specify to `np.array` that the array elements should be of type `object`. This type can store any Python value, and so, when the array is storing strings, it will use Python's own string values as elements, rather than the more efficient but more fragile Unicode strings that Numpy uses by default.

```
# Set a new value for the first element in the new array.
small_strings_better[0] = 'seven'
small_strings_better
```

```
array(['seven', 'one', 'two'], dtype=object)
```

As for any array, you can select elements with *indexing*. When you select an element with a given position (index), you get the *string* at at that position:

```
# Julian Lincoln Simon's second name.
# (Remember, Python's positions start at 0).
middle_name = arr_of_strings[1]
middle_name
```

```
np.str_('Lincoln')
```

As for numbers, we can compare strings with, for example, the `==` operator, that asks whether the two strings are equal:

```
middle_name == 'Lincoln'
```

```
True
```

7.6 Repeating elements

Now let us go back to the problem of selecting black and white jurors.

We started with the strategy of using numbers 0 through 25 to mean "black" jurors, and 26 through 99 to mean "white" jurors. We selected values at random from 0 through 99, and then worked out whether the number meant a "black" juror (was less than 26) or a "white" juror (was greater than 25).

It would be good to use strings instead of numbers to identify the potential jurors. Then we would not have to remember our coding of 0 through 25 and 26 through 99.

If only there was a way to make an array of 100 strings, where 26 of the strings were "black" and 74 were "white". Then we could select randomly from that array, and it would be immediately obvious that we had a "black" or "white" juror.

Luckily, of course, we can do that, by using the `np.repeat` function to construct the array.

Here is how that works:

```
# The values that we will repeat to fill up the larger array.
# Use a list to store the sequence of values.
juror_types = ['black', 'white']
# The number of times we want to repeat "black" and "white".
# Use a list to store the sequence of values.
repeat_nos = [26, 74]
# Repeat "black" 26 times and "white" 74 times.
# We have passed two lists here, but we could also have passed
# arrays - the Numpy repeat function converts the lists to arrays
# before it builds the repeats.
jury_pool = np.repeat(juror_types, repeat_nos)
# Show the result
jury_pool
```

```
array(['black', 'black', 'black', 'black', 'black', 'black', 'black',
       'black', 'black', 'black', 'black', 'black', 'black', 'black',
       'black', 'black', 'black', 'black', 'black', 'black', 'black',
       'black', 'black', 'black', 'black', 'black', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
```

```
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white'], dtype='<U5')
```

We can use this array of repeats of strings, to sample from. The result is easier to grasp, because we are using the string labels, instead of numbers:

```
# Select one juror at random from the black / white pool.
one_juror = rnd.choice(jury_pool)
one_juror
```

```
np.str_('white')
```

We can select our full jury of 12 jurors, and see the results in a more obvious form:

```
# Select 12 jurors at random from the black / white pool.
one_jury = rnd.choice(jury_pool, 12)
one_jury
```

```
array(['white', 'white', 'white', 'white', 'black', 'white', 'black',
       'white', 'white', 'black', 'black', 'white'], dtype='<U5')
```

Using the `size` argument to `rnd.choice`

In the code above, we have specified the *size* of the sample we want (12) with the second argument to `rnd.choice`. As you saw in Section 5.8, we can also give names to the function arguments, in this case, to make it clearer what we mean by "12" in the code above. In fact, from now on, that is what we will do; we will specify the *size* of our sample by using the *name* for the function argument to `rnd.choice` — `size` — like this:

```
# Select 12 jurors at random from the black / white pool.
# Specify the sample size using the "size" named argument.
one_jury = rnd.choice(jury_pool, size=12)
one_jury
```

```
array(['black', 'white', 'white', 'white', 'black', 'white', 'black',
       'white', 'white', 'white', 'white', 'white'], dtype='<U5')
```

We can use `==` on the array to get `True` values where the juror was "black" and `False` values otherwise:

```
are_black = one_jury == 'black'
are_black
```

```
array([ True, False, False, False,  True, False,  True, False, False,
       False, False, False])
```

Finally, we can `np.sum` to find the number of black jurors (Section 5.14):

```
# Number of black jurors in this simulated jury.
n_black = np.sum(are_black)
n_black
```

```
np.int64(3)
```

Putting that all together, this is our new procedure to select one jury and count the number of black jurors:

```
one_jury = rnd.choice(jury_pool, size=12)
are_black = one_jury == 'black'
n_black = np.sum(are_black)
n_black
```

```
np.int64(3)
```

Or we can be even more compact by putting several statements together into one line:

```
# The same as above, but on one line.
n_black = np.sum(rnd.choice(jury_pool, size=12) == 'black')
n_black
```

```
np.int64(1)
```

7.7 Resampling with and without replacement

Now let us return to the details of Robert Swain's case, that you first saw in Chapter 6.

We looked at the composition of Robert Swain's 12-person jury — but in fact, by law, that does not have to be representative of the eligible jurors. The 12-person jury is drawn from a jury *panel*, of 100 people, and this should, in turn, be drawn from the population of all eligible jurors in the county, consisting, at the time, of "all male citizens in the community over 21 who are reputed to be honest, intelligent men and are esteemed for their integrity, good character and sound judgment." So, unless there was some bias against black jurors, we might expect the 100-person jury panel to be a plausibly random sample of the eligible jurors, of whom 26% were black. See the Supreme Court case judgement for details.

In fact, in Robert Swain's trial, there were 8 black members in the 100-person jury panel. We will leave it to you to adapt the simulation from Chapter 6 to ask the question — is 8% surprising as a random sample from a population with 26% black people?

But we have a different question: given that 8 out of 100 of the jury panel were black, is it surprising that none of the 12-person jury were black? As usual, we can answer that question with simulation.

Let's think about what a single simulated jury selection would look like.

First we compile a representation of the actual jury panel, using the tools we have used above.

```
juror_types = ['black', 'white']
# in fact there were 8 black jurors and 92 white jurors.
panel_nos = [8, 92]
jury_panel = np.repeat(juror_types, panel_nos)
# Show the result
jury_panel
```

```
array(['black', 'black', 'black', 'black', 'black', 'black', 'black',
       'black', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
```

```
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white', 'white', 'white',
       'white', 'white'], dtype='<U5')
```

Now consider taking a 12-person jury at random from this panel. We select the first juror at random, so that juror has an 8 out of 100 chance of being black. But when we select the second jury member, the situation has changed slightly. We can't select the first juror again, so our panel is now 99 people. If our first juror was black, then the chances of selecting another black juror next are not 8 out of 100, but 7 out of 99 — a smaller chance. The problem is, as we shall see in more detail later, the chances of getting a black juror as the second, and third and fourth members of the jury depend on whether we selected a black juror as the first and second and third jury members. At its most extreme, imagine we had already selected eight jurors, and by some strange chance, all eight were black. Now our chances of selecting a black juror as the ninth juror are zero — there are no black jurors left to select from the panel.

In this case we are selecting jurors from the panel *without replacement*, meaning, that once we have selected a particular juror, we cannot select them again, and we do not put them back into the panel when we select our next juror.

This is the probability equivalent of the situation when you are dealing a hand of cards. Let's say someone is dealing you, and you only, a hand of five cards. You get an ace as your first card. Your chances of getting an ace as your first card were just the number of aces in the deck divided by the number of cards — four in 52 – $\frac{4}{52}$. But for your second card, the probability has changed, because there is one less ace remaining in the pack, and one less card, so your chances of getting an ace as your second card are now $\frac{3}{51}$. This is sampling without replacement — in a normal game, you can't get the same card twice. Of course, you could imagine getting a hand where you sampled *with replacement*. In that case, you'd get a card, you'd write down what it was, and you'd give the card back to the dealer, who would *replace* the card in the deck, shuffle again, and give you another card.

As you can see, the chances change if you are sampling with or without replacement, and the kind of sampling you do, will dictate how you model your chances in your simulations.

Because this distinction is so common, and so important, the machinery you have already seen in `rnd.choice` has simple ways for you to select your sampling type. You have already seen sampling *with replacement*, and it looks like this:

```
# Take a sample of 12 jurors from the panel *with replacement*
# With replacement is the default for `rnd.choice`.
strange_jury = rnd.choice(jury_panel, size=12)
strange_jury
```

```
array(['white', 'white', 'white', 'black', 'white', 'white', 'white',
       'white', 'white', 'white', 'white', 'white'], dtype='<U5')
```

This is a strange jury, because it can select any member of the jury pool more than once. Perhaps that juror would have to fill two (or more!) seats, or run quickly between them. But of course, that is not how juries are selected. They are selected *without replacement*:

```
# Take a sample of 12 jurors from the panel *without replacement*
ok_jury = rnd.choice(jury_panel, 12, replace=False)
ok_jury
```

```
array(['white', 'white', 'white', 'white', 'black', 'white', 'white',
       'white', 'white', 'white', 'white', 'white'], dtype='<U5')
```

> **Note 8: Comments at the end of lines**
>
> You have already seen comment lines. These are lines beginning with `#`, to signal to Python that the rest of the line is text for humans to read, but Python to ignore.
>
> ```
> # This is a comment.  Python ignores this line.
> ```
>
> You can also put comments at the *end of code lines*, by finishing the code part of the line, and then putting a `#`, followed by more text. Again, Python will ignore everything after the `#` as a text for humans, but not for Python.
>
> ```
> print('Hello')  # This is a comment at the end of the line.
> ```
>
> ```
> Hello
> ```

To finish the procedure for simulating a single jury selection, we count the number of black jurors:

```
n_black = np.sum(ok_jury == 'black')  # How many black jurors?
n_black
```

```
np.int64(1)
```

Now we have the procedure for one simulated trial, here is the procedure for 10000 simulated trials.

```
counts = np.zeros(10000)
for i in np.arange(10000):
    # Single trial procedure
    jury = rnd.choice(jury_panel, size=12, replace=False)
    n_black = np.sum(jury == 'black')  # How many black jurors?
    # Store the result
    counts[i] = n_black

# Number of juries with 0 black jurors.
zero_black = np.sum(counts == 0)
# Proportion
p_zero_black = zero_black / 10000
print(p_zero_black)
```

```
0.3421
```

We have found that, when there are only 8% black jurors in the jury panel, having no black jurors in the final jury happens about 34% of the time, even in this case, where the jury is selected completely at random from the jury panel.

We should look for the main source of bias in the initial selection of the jury panel, not in the selection of the jury from the panel.

i End of notebook: Sampling tools

`sampling_tools` starts at Note 7.

i With or without replacement for the original jury selection

You may have noticed in Chapter 6 that we were sampling Robert Swain's jury from the eligible pool of jurors, *with replacement*. You might reasonably ask whether we should have selected from the eligible jurors *without replacement*, given that the same juror cannot serve more than once in the same jury, and therefore, the same argument applies there as here.

The trick there was that we were selecting from a very large pool of many thousand eligible jurors, of whom 26% were black. Let's say there were 10,000 eligible jurors, of whom 2,600 were black. When selecting the first juror, there is exactly a 2,600 in 10,000 chance of getting a black juror — 26%. If we do get a black juror first, then the chance that the second juror will be black has changed slightly, 2,599 in 9,999. But these changes are very small; even if we select eleven black jurors out of eleven, when we come to the twelfth juror, we still have a 2,589 out of 9,989 chance of getting another black juror, and that works out at a 25.92% chance — hardly changed from the original 26%. So

yes, you'd be right, we really should have compiled our population of 2,600 black jurors and 7,400 white jurors, and then sampled without replacement from that population, but as the resulting sample probabilities will be very similar to the simpler sampling with replacement, we chose to try and slide that one quietly past you, in the hope you would forgive us when you realized.

7.8 Conclusion

This chapter introduced you to the idea of strings — values in Python that store bits of text. Strings are very useful as labels for the entities we are sampling from, when we do our simulations. Strings are particularly useful when we use them with arrays, and one way we often do that is to build up arrays of strings to sample from, using the `np.repeat` function.

There is a fundamental distinction between two different types of sampling — sampling *with replacement*, where we draw an element from a larger pool, then put that element back before drawing again, and sampling *without replacement*, where we remove the element from the remaining pool when we draw it into the sample. As we will see later, it is often a judgment call which of these two types of sampling is a more reasonable model of the world you are trying to simulate.

8 Probability Theory, Part 1

8.1 Introduction

Let's assume we understand the nature of the system or mechanism that produces the uncertain events in which we are interested. That is, the probability of the relevant independent *simple* events is assumed to be known, the way we assume we know the probability of a single "6" with a given die. The task is to determine the probability of various sequences or combinations of the simple events — say, three "6's" in a row with the die. These are the sorts of probability problems dealt with in this chapter.

The resampling method — or just call it simulation or Monte Carlo method, if you prefer — will be illustrated with classic examples. Typically, a single trial of the system is simulated with cards, dice, random numbers, or a computer program. Then trials are repeated again and again to estimate the frequency of occurrence of the event in which we are interested; this is the probability we seek. We can obtain as accurate an estimate of the probability as we wish by increasing the number of trials. The key task in each situation is *designing an experiment that accurately simulates the system in which we are interested.*

This chapter begins the Monte Carlo simulation work that culminates in the resampling method in statistics proper. The chapter deals with problems in probability theory — that is, situations where one wants to estimate the probability of one or more particular events when the basic structure and parameters of the system are known. In later chapters we move on to inferential statistics, where similar simulation work is known as resampling.

8.2 Definitions

A few definitions first:

- *Simple Event*: An event such as a single flip of a coin, or one draw of a single card. A simple event cannot be broken down into simpler events of a similar sort.
- *Simple Probability* (also called "primitive probability"): The probability that a simple event will occur; for example, that my favorite football team, the Washington Commanders, will win on Sunday.

During a recent season, the "experts" said that the Commanders had a 60 percent chance of winning on Opening Day; that estimate is a simple probability. We can *model* that probability by putting into a bucket six green balls to stand for wins, and four red balls to stand for losses (or we could use 60 and 40 balls, or 600 and 400). For the outcome on any given day, we draw one ball from the bucket, and record a simulated win if the ball is green, a loss if the ball is red.

So far the bucket has served only as a physical representation of our thoughts. But as we shall see shortly, this representation can help us think clearly about the process of interest to us. It can also give us information that is not yet in our thoughts.

Estimating simple probabilities wisely depends largely upon gathering evidence well. It also helps to adjust one's probability estimates skillfully to make them internally consistent. Estimating probabilities has much in common with estimating lengths, weights, skills, costs, and other subjects of measurement and judgment.

Some more definitions:

- *Composite Event*: A composite event is the combination of two or more simple events. Examples include all heads in three throws of a single coin; all heads in one throw of three coins at once; Sunday being a nice day *and* the Commanders winning; and the birth of nine females out of the next ten calves born if the chance of a female in a single birth is 0.48.
- *Compound Probability*: The probability that a composite event will occur.

The difficulty in estimating *simple* probabilities such as the chance of the Commanders winning on Sunday arises from our lack of understanding of the world around us. The difficulty of estimating *compound* probabilities such as the probability of it being a nice day Sunday *and* the Commanders winning is the weakness in our mathematical intuition interacting with our lack of understanding of the world around us. Our task in the study of probability and statistics is to overcome the weakness of our mathematical intuition by using a systematic process of simulation (or the devices of formulaic deductive theory).

Consider now a question about a compound probability: What are the chances of the Commanders winning their first *two* games if we think that *each* of those games can be modeled by our bucket containing six red and four green balls? If one drawing from the bucket represents one game, a second drawing should represent the second game (assuming we replace the first ball drawn in order to keep the chances of winning the same for the two games). If so, two drawings from the bucket should represent two games. And we can then estimate the compound probability we seek with a series of two-ball trial experiments.

More specifically, our procedure in this case — the prototype of all procedures in the resampling simulation approach to probability and statistics — is as follows:

1. Put six green ("Win") and four red ("Lose") balls in a bucket.

2. Draw a ball, record its color, and replace it (so that the probability of winning the second simulated game is the same as the first).
3. Draw another ball and record its color.
4. If both balls drawn were green record "Yes"; otherwise record "No."
5. Repeat steps 2-4 a thousand times.
6. Count the proportion of "Yes"s to the total number of "Yes"s and "No"s; the result is the probability we seek.

Much the same procedure could be used to estimate the probability of the Commanders winning (say) 3 of their next 4 games. We will return to this illustration again and we will see how it enables us to estimate many other sorts of probabilities.

- *Experiment or Experimental Trial, or Trial, or Resampling Experiment*: A simulation experiment or trial is a randomly-generated composite event which has the same characteristics as the actual composite event in which we are interested (except that in inferential statistics the resampling experiment is generated with the "benchmark" or "null" universe rather than with the "alternative" universe).
- *Parameter*: A numerical property of a universe. For example, the "true" mean (don't worry about the meaning of "true"), and the range between largest and smallest members, are two of its parameters.

8.3 Theoretical and historical methods of estimation

As introduced in Section 3.5, there are two general ways to tackle any probability problem: *theoretical-deductive* and *empirical*, each of which has two sub-types. These concepts have complicated links with the concept of "frequency series" discussed earlier.

- *Empirical Methods.* One empirical method is to look at *actual cases in nature* — for example, examine all (or a sample of) the families in Brazil that have four children and count the proportion that have three girls among them. (This is the most fundamental process in science and in information-getting generally. But in general we do not discuss it in this book and leave it to courses called "research methods." I regard that as a mistake and a shame, but so be it.) In some cases, of course, we cannot get data in such fashion because it does not exist.

  Another empirical method is to manipulate the simple elements in such fashion as to produce hypothetical experience with how the simple elements behave. This is the heart of the resampling method, as well as of physical simulations such as wind tunnels.

- *Theoretical Methods.* The most fundamental theoretical approach is to resort to first principles, working with the elements in their full deductive simplicity, and examining all possibilities. This is what we do when we use a tree diagram to calculate the probability of three girls in families of four children.

The formulaic approach is a theoretical method that aims to avoid the inconvenience of resorting to first principles, and instead uses calculation shortcuts that have been worked out in the past.

What the Book Teaches. This book teaches you the empirical method using hypothetical cases. Formulas can be misleading for most people in most situations, and should be used as a shortcut only when a person understands exactly which first principles are embodied in the formulas. But most of the time, students and practitioners resort to the formulaic approach without understanding the first principles that lie behind them — indeed, their own teachers often do not understand these first principles — and therefore they have almost no way to verify that the formula is right. Instead they use canned checklists of qualifying conditions.

8.4 Samples and universes

The terms "sample" and "universe" (or "population")[1] were used earlier without definition. But now these terms must be defined.

8.4.1 The concept of a sample

For our purposes, a "sample" is a collection of observations for which you obtain the data to be used in the problem. Almost any set of observations for which you have data constitutes a sample. (You might, or might not, choose to call a complete census a sample.)

8.5 The concept of a universe or population

For every sample there must also be a universe "behind" it. But "universe" is harder to define, partly because it is often an *imaginary* concept. A universe is the collection of things or people *that you want to say that your sample was taken from.* A universe can be finite and well defined — "all live holders of the Congressional Medal of Honor," "all presidents of major universities," "all billion-dollar corporations in the United States." Of course, these finite universes may not be easy to pin down; for instance, what is a "major university"? And these universes may contain some elements that are difficult to find; for instance, some Congressional Medal winners may have left the country, and there may not be adequate public records on some billion-dollar corporations.

Universes that are called "infinite" are harder to understand, and it is often difficult to decide which universe is appropriate for a given purpose. For example, if you are studying a sample of patients suffering from schizophrenia, what is the universe from which the sample comes?

[1] "Universe" and "population" are perfect synonyms in scientific research. We choose to use "universe" because it seems to have fewer confusing associations.

Depending on your purposes, the appropriate universe might be all patients with schizophrenia now alive, or it might be all patients who might *ever* live. The latter concept of the universe of patients with schizophrenia is *imaginary* because some of the universe does not exist. And it is *infinite* because it goes on forever.

Not everyone likes this definition of "universe." Others prefer to think of a universe, not as the collection of people or things that you *want* to say your sample was taken from, but as the collection that the sample was *actually* taken from. This latter view equates the universe to the "sampling frame" (the actual list or set of elements you sample from) which is always finite and existent. The definition of universe offered here is simply the most practical, in our opinion.

8.6 The conventions of probability

Let's review the basic conventions and rules used in the study of probability:

1. Probabilities are expressed as decimals between 0 and 1, like percentages. The weather forecaster might say that the probability of rain tomorrow is 0.2, or 0.97.
2. The probabilities of all the possible alternative outcomes in a single "trial" must add to unity. If you are prepared to say that it must either rain or not rain, with no other outcome being possible — that is, if you consider the outcomes to be *mutually exclusive* (a term that we discuss below), then one of those probabilities implies the other. That is, if you estimate that the probability of rain is 0.2 — written $P(\text{rain}) = 0.2$ — that implies that you estimate that $P(\text{no rain}) = 0.8$.

i Writing probabilities

We will now be writing some simple formulae using probability. Above we write the *probability of rain tomorrow* as $P(\text{rain})$. This probability might be 0.2, and we could write this as:

$$P(\text{rain}) = 0.2$$

We can term "rain tomorrow" an *event* — the event may occur: rain, or it may *not* occur: no rain.
We often shorten the *name* of our event — here rain — to a single letter, such as R. So, in this case, we could write $P(\text{rain}) = 0.2$ as $P(R) = 0.2$ — meaning the same thing. We tend to prefer single letters — as in $P(R)$ — to longer names — as in $P(\text{rain})$. This is because the single letters can be easier to read in these compact formulae.
Above we have written the probability of "rain tomorrow" event *not* occurring as $P(\text{no rain})$. Another way of referring to an event *not* occurring is to suffix the event name with a *caret* (^) character like this: $\hat{R}$. So read $P(\hat{R})$ as "the probability that it

will not rain", and it is just another way of writing P(no rain). We sometimes call $\hat{}R$ the *complement* of R.
We use and between two events to mean *both* events occur.
For example, say we call the event "Commanders win the game" as W. One example of a *compound event* (see above) would be the event $W$and$R$, meaning, the event where the Commanders won the game *and* it rained.

8.7 Mutually exclusive events — the addition rule

Definition: If there are just two events A and B and they are "mutually exclusive" or "disjoint," each implies the absence of the other. Green and red coats are mutually exclusive for you if (but only if) you never wear more than one coat at a time.

To state this idea formally, if A and B are mutually exclusive, then:

$$P(A \text{ and } B) = 0$$

If A is "wearing a green coat" and B is "wearing a red coat" (and you never wear two coats at the same time), then the probability that you are wearing a green coat *and* a red coat is 0: $P(A \text{ and } B) = 0$.

In that case, outcomes A and B, and hence outcome A and its own absence (written $P(\hat{}A)$), are necessarily mutually exclusive, and hence the two probabilities add to unity:

$$P(A) + P(\hat{}A) = 1$$

The sales of your store in a given year cannot be both above and below \$1 million. Therefore if $P(\text{sales} > \$1 \text{ million}) = 0.2$, $P(\text{sales} <= \$1 \text{ million}) = 0.8$.

This "complements" rule is useful as a consistency check on your estimates of probabilities. If you say that the probability of rain is 0.2, then you should check that you think that the probability of no rain is 0.8; if not, reconsider both the estimates. The same for the probabilities of your team winning and losing its next game.

8.8 Joint probabilities

Let's return now to the Commanders. We said earlier that our best guess of the probability that the Commanders will win the first game is 0.6. Let's complicate the matter a bit and say that the probability of the Commanders winning depends upon the weather; on a nice day we estimate a 0.65 chance of winning, on a nasty (rainy or snowy) day a chance of 0.55. It is obvious that we then want to know the chance of a nice day, and we estimate a probability of 0.7. Let's now ask the probability that both will happen — *it will be a nice day and the Commanders will win.* Before getting on with the process of estimation itself, let's tarry a moment to discuss the probability estimates. Where do we get the notion that the probability of a nice day next Sunday is 0.7? We might have done so by checking the records of the past 50 years, and finding 35 nice days on that date. If we assume that the weather has not changed over that period (an assumption that some might not think reasonable, and the wisdom of which must be the outcome of some non-objective judgment), our probability estimate of a nice day would then be $35/50 = 0.7$.

Two points to notice here: 1) The source of this estimate is an objective "frequency series." And 2) the data come to us as the records of 50 days, of which 35 were nice. We would do best to stick with exactly those numbers rather than convert them into a single number — 70 percent. Percentages have a way of being confusing. (When his point score goes up from 2 to 3, my racquetball partner is fond of saying that he has made a "fifty percent increase"; that's just one of the confusions with percentages.) And converting to a percent loses information: We no longer know how many observations the percent is based upon, whereas 35/50 keeps that information.

Now, what about the estimate that the Commanders have a 0.65 chance of winning on a nice day — where does that come from? Unlike the weather situation, there is no long series of stable data to provide that information about the probability of winning. Instead, we *construct* an estimate using whatever information or "hunch" we have. The information might include the Commanders' record earlier in this season, injuries that have occurred, what the "experts" in the newspapers say, the gambling odds, and so on. The result certainly is not "objective," or the result of a stable frequency series. But we treat the 0.65 probability in quite the same way as we treat the .7 estimate of a nice day. In the case of winning, however, we produce an estimate expressed directly as a percent.

If we are shaky about the estimate of winning — as indeed we ought to be, because so much judgment and guesswork inevitably goes into it — we might proceed as follows: Take hold of a bucket and two bags of balls, green and red. Put into the bucket some number of green balls — say 10. Now add enough red balls to express your judgment that the *ratio* is the ratio of expected wins to losses on a nice day, adding or subtracting green balls as necessary to get the ratio you want. If you end up with 13 green and 7 red balls, then you are "modeling" a probability of 0.65, as stated above. If you end up with a different ratio of balls, then you have learned from this experiment with your own mind processes that you think that the probability of a win on a nice day is something other than 0.65.

Don't put away the bucket. We will be using it again shortly. And keep in mind how we have just been using it, because our use later will be somewhat different though directly related.

One good way to begin the process of producing a compound estimate is by portraying the available data in a "tree diagram" like Figure 8.1. The tree diagram shows the possible events in the order in which they might occur. A tree diagram is extremely valuable whether you will continue with either simulation or the formulaic method.

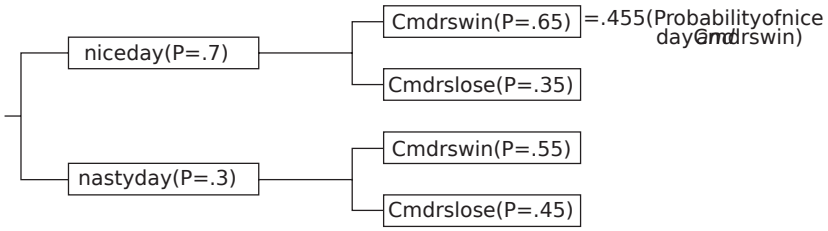


Figure 8.1: Tree diagram

8.9 The Monte Carlo simulation method (resampling)

The steps we follow to simulate an answer to the compound probability question are as follows:

1. Put seven blue balls (for "nice day") and three yellow balls ("not nice") into a bucket labeled A.
2. Put 65 green balls (for "win") and 35 red balls ("lose") into a bucket labeled B. This bucket represents the chance that the Commanders will when it is a nice day.
3. Draw one ball from bucket A. If it is blue, carry on to the next step; otherwise record "no" and stop.
4. If you have drawn a blue ball from bucket A, now draw a ball from bucket B, and if it is green, record "yes" on a score sheet; otherwise write "no."
5. Repeat steps 3-4 perhaps 10000 times.
6. Count the number of "yes" trials.
7. Compute the probability you seek as (number of "yeses"/ 10000). (This is the same as (number of "yeses"/ (number of "yeses" + number of "noes")

Actually doing the above series of steps by hand is useful to build your intuition about probability and simulation methods. But the procedure can also be simulated with a computer. We will use Python to do this in a moment.

8.10 If statements in Python

Before we get to the simulation, we need another feature of Python, called a *conditional* or *if* statement.

Here we have rewritten step 4 above, but using indentation to emphasize the idea:

```
If you have drawn a blue ball from bucket A:
    Draw a ball from bucket B
    if the ball is green:
        record "yes"
    otherwise:
        record "no".
```

Notice the structure. The first line is the *header* of the `if` statement. It has a *condition* — this is why `if` statements are often called *conditional* statements. The condition here is "you have drawn a blue ball from bucket A". If this condition is met — it is True that you have drawn a blue ball from bucket A *then* we go on to do the stuff that is indented. Otherwise we do not do any of the stuff that is indented.

The indented stuff above is the *body* of the `if` statement. It is the stuff we do `if` the *conditional* at the top is True.

Now let's see how we would write that in Python.

Let's make bucket A. Remember, this is the *weather* bucket. It has seven blue balls (for 70% fine days) and 3 yellow balls (for 30% rainy days). See Section 7.6 for the `np.repeat` way of repeating elements multiple times.

> **Note 9: Notebook: Fine day and win**
>
> - Download notebook
> - Interact

```
# Load the Numpy array library.
import numpy as np

# Make a random number generator
rnd = np.random.default_rng()
```

```
# blue means "nice day", yellow means "not nice".
bucket_A = np.repeat(['blue', 'yellow'], [7, 3])
bucket_A
```

```
array(['blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'yellow',
       'yellow', 'yellow'], dtype='<U6')
```

Now let us draw a ball at random from bucket_A:

```
a_ball = rnd.choice(bucket_A)
a_ball
```

```
np.str_('blue')
```

How we run our first `if` statement. Running this code will display "The ball was blue" if the ball was blue, otherwise it will not display anything:

```
if a_ball == 'blue':
    print('The ball was blue')
```

```
The ball was blue
```

Notice that the header line has `if`, followed by the conditional expression (question) `a_ball == 'blue'`. The header line finishes with a colon `:`. The *body* of the `if` statement is one or more *indented* lines. Here there is only one line: `print('The ball was blue')`. Python only runs the body of the if statement if the *condition* is `True`.[2]

To confirm we see "The ball was blue" if `a_ball` is `'blue'` and nothing otherwise, we can set `a_ball` and re-run the code:

```
# Set value of a_ball so we know what it is.
a_ball = 'blue'
```

```
if a_ball == 'blue':
    # The conditional statement is True in this case, so the body does run.
    print('The ball was blue')
```

```
The ball was blue
```

[2] In this case, the result of the conditional expression is in fact either `True` or `False`. Python is more liberal on what it allows in the conditional expression; it will take whatever the result is, and then force the result into either `True` or `False`, in fact, by wrapping the result with the `bool` function, that takes anything as input, and returns either `True` or `False`. Therefore, we could refer to the result of the conditional expression as something "truthy" — that is - something that comes back as `True` or `False` from the `bool` function. In the case here, that does not arise, because the result is in fact either exactly `True` or exactly `False`.

```
a_ball = 'yellow'
```

```
if a_ball == 'blue':
    # The conditional statement is False, so the body does not run.
    print('The ball was blue')
```

We can add an `else` clause to the `if` statement. Remember the *body* of the `if` statement runs if the *conditional expression* (here `a_ball == 'blue'`) is `True`. The `else` clause runs if the conditional statement is `False`. This may be clearer with an example:

```
a_ball = 'blue'
```

```
if a_ball == 'blue':
    # The conditional expression is True in this case, so the body runs.
    print('The ball was blue')
else:
    # The conditional expression was True, so the else clause does not run.
    print('The ball was not blue')
```

```
The ball was blue
```

Notice that the `else` clause of the `if` statement starts with a header line — `else` — followed by a colon `:`. It then has its own indented *body* of indented code. The body of the `else` clause only runs if the initial conditional expression is *not* `True`.

```
a_ball = 'yellow'
```

```
if a_ball == 'blue':
    # The conditional expression was False, so the body does not run.
    print('The ball was blue')
else:
    # but the else clause does run.
    print('The ball was not blue')
```

```
The ball was not blue
```

With this machinery, we can now implement the full logic of step 4 above:

```
If you have drawn a blue ball from bucket A:
    Draw a ball from bucket B
    if the ball is green:
        record "yes"
    otherwise:
        record "no".
```

Here is bucket B. Remember green means "win" (65% of the time) and red means "lose" (35% of the time). We could call this the "Commanders win when it is a nice day" bucket:

```
bucket_B = np.repeat(['green', 'red'], [65, 35])
```

The full logic for step 4 is:

```
# By default, say we have no result.
result = 'No result'
a_ball = rnd.choice(bucket_A)
# If you have drawn a blue ball from bucket A: (then run indented code)
if a_ball == 'blue':
    # Draw a ball at random from bucket B
    b_ball = rnd.choice(bucket_B)
    # if the ball is green: (then run the indented code)
    if b_ball == 'green':
        # record "yes"
        result = 'yes'
    # otherwise (not green):
    else:
        # record "no".
        result = 'no'
# Show what we got in this case.
result
```

```
'yes'
```

Now we have everything we need to run many trials with the same logic.

```
# The result of each trial.
# To start with, say we have no result for all the trials.
z = np.repeat(['No result'], 10000)

# Repeat trial procedure 10000 times
```

```
for i in range(10000):
    # draw one "ball" for the weather, store in "a_ball"
    # blue is "nice day", yellow is "not nice"
    a_ball = rnd.choice(bucket_A)
    if a_ball == 'blue':  # nice day
        # if no rain, check on game outcome
        # green is "win" (give nice day), red is "lose" (given nice day).
        b_ball = rnd.choice(bucket_B)
        if b_ball == 'green':  # Commanders win
            # Record result.
            z[i] = 'yes'
        else:
            z[i] = 'no'
    # End of trial, go back to the beginning until done.

# Count of the number of times we got "yes".
k = np.sum(z == 'yes')
# Show the proportion of *both* fine day *and* wins
kk = k / 10000
kk
```

```
np.float64(0.4603)
```

The above procedure gives us the probability that it will be a nice day and the Commanders will win — about 46%.

> **i** End of notebook: Fine day and win
>
> `fine_win` starts at Note 9.

Let's say that we think that the Commanders have a 0.55 (55%) chance of winning on a not-nice day. With the aid of a bucket with a different composition — one made by substituting 55 green and 45 yellow balls in Step 4 — a similar procedure yields the chance that it will be a *nasty* day and the Commanders will win. With a similar substitution and procedure we could also estimate the probabilities that it will be a nasty day and the Commanders will lose, and a nice day and the Commanders will lose. The sum of these probabilities should come close to unity, because the sum includes all the possible outcomes. But it will not *exactly* equal unity because of what we call "sampling variation" or "sampling error."

Please notice that each trial of the procedure begins with the same numbers of balls in the buckets as the previous trial. That is, you must replace the balls you draw after each trial in order that the probabilities remain the same from trial to trial. Later we will discuss the general concept of replacement versus non-replacement more fully.

8.11 The deductive formulaic method

It also is possible to get an answer with formulaic methods to the question about a nice day and the Commanders winning. The following discussion of nice-day-Commanders-win handled by formula is a prototype of the formulaic deductive method for handling other problems.

Return now to the tree diagram (Figure 8.1) above. We can read from the tree diagram that 70 percent of the time it will be nice, and of that 70 percent of the time, 65 percent of the games will be wins. That is, $0.65 * 0.7 = 0.455 =$ the probability of a nice day and a win. That is the answer we seek. The method seems easy, but it also is easy to get confused and obtain the wrong answer.

8.12 Multiplication rule

We can generalize what we have just done. The foregoing formula exemplifies what is known as the "multiplication rule":

$$P(\text{nice day and win}) = P(\text{nice day}) * P(\text{winning} \mid \text{nice day})$$

where the vertical line in $P(\text{winning} \mid \text{nice day})$ means "conditional upon" or "given that." That is, the vertical line indicates a "conditional probability," a concept we must consider in a minute.

The multiplication rule is a formula that produces the probability of the *combination (juncture) of two or more events*. More discussion of it will follow below.

8.13 Conditional and unconditional probabilities

Two kinds of probability statements — *conditional* and *unconditional* — must now be distinguished.

It is the appropriate concept when many factors, all small relative to each other rather than one force having an overwhelming influence, affect the outcome.

A *conditional* probability is formally written $P(\text{Commanders win} \mid \text{rain}) = 0.65$, and it is read "The probability that the Commanders will win if (given that) it rains is 0.65." It is the appropriate concept when there is one (or more) major event of interest in decision contexts.

Let's use another football example to explain conditional and unconditional probabilities. In the year this was being written, the University of Maryland had an unpromising football team. Someone may nevertheless ask what chance the team had of winning the post season game at the bowl to which only the best team in the University of Maryland's league is sent. One

may say that *if* by some miracle the University of Maryland does get to the bowl, its chance would be a bit less than 50- 50 — say, 0.40. That is, the probability of its winning, *conditional* on getting to the bowl is 0.40. But the chance of its getting to the bowl at all is very low, perhaps 0.01. If so, the unconditional probability of winning at the bowl is the probability of its getting there multiplied by the probability of winning *if* it gets there; that is, 0.01 x 0.40 = 0.004. (It would be even better to say that .004 is the probability of winning conditional only on having a team, there being a league, and so on, all of which seem almost sure things.) Every probability is conditional on many things — that war does not break out, that the sun continues to rise, and so on. But if all those unspecified conditions are very sure, and can be taken for granted, we talk of the probability as unconditional.

A conditional probability is a statement that the probability of an event is such-and-such *if* something else is so-and-so; it is the "if" that makes a probability statement conditional. True, in *some* sense all probability statements are conditional; for example, the probability of an even-numbered spade is 6/52 *if* the deck is a poker deck and not necessarily if it is a pinochle deck or Tarot deck. But we ignore such conditions for most purposes.

Most of the use of the concept of probability in the social sciences is conditional probability. All hypothesis-testing statistics (discussed starting in Chapter 20) are conditional probabilities.

Here is the typical conditional-probability question used in social-science statistics: What is the probability of obtaining this sample S (by chance) *if* the sample were taken from universe A? For example, what is the probability of getting a sample of five children with I.Q.s over 100 *by chance* in a sample randomly chosen from the universe of children whose average I.Q. is 100?

One way to obtain such conditional-probability statements is by examination of the results generated by universes like the conditional universe. For example, assume that we are considering a universe of children where the average I.Q. is 100.

Write down "over 100" and "under 100" respectively on many slips of paper, put them into a hat, draw five slips several times, and see how often the first five slips drawn are all over 100. This is the resampling (Monte Carlo simulation) method of estimating probabilities.

Another way to obtain such conditional-probability statements is formulaic calculation. For example, if half the slips in the hat have numbers under 100 and half over 100, the probability of getting five in a row above 100 is 0.03125 — that is, 0.5^5, or 0.5 x 0.5 x 0.5 x 0.5 x 0.5, using the multiplication rule introduced above. But if you are not absolutely sure you know the proper mathematical formula, you are more likely to come up with a sound answer with the simulation method.

Let's illustrate the concept of conditional probability with four cards — two aces and two 3's (or two black and two red). What is the probability of an ace? Obviously, 0.5. If you first draw an ace, what is the probability of an ace now? That is, what is the probability of an ace *conditional on* having drawn one already? Obviously not 0.5.

This change in the conditional probabilities is the basis of mathematician Edward Thorp's famous system of card-counting to beat the casinos at blackjack (Twenty One).

Casinos can defeat card counting by using many decks at once so that conditional probabilities change more slowly, and are not very different than unconditional probabilities. Looking ahead, we will see that sampling with replacement, and sampling without replacement from a huge universe, are much the same in practice, so we can substitute one for the other at our convenience.

Let's further illustrate the concept of conditional probability with a puzzle (from Gardner 2001, 288). "… shuffle a packet of four cards — two red, two black — and deal them face down in a row. Two cards are picked at random, say by placing a penny on each. What is the probability that those two cards are the same color?"

1. Play the game with the cards 100 times, and estimate the probability sought.

OR

1. Put slips with the numbers "1," "1," "2," and "2" in a hat, or in an array named `N` on a computer.
2. Shuffle the slips of paper by shaking the hat or shuffling the array (of which more below).
3. Take two slips of paper from the hat or from `N`, to get two numbers.
4. Call the first number you selected `A` and the second `B`.
5. Are `A` and `B` the same? If so, record "Yes" otherwise "No".
6. Repeat (2-5) 10000 times, and count the proportion of "Yes" results. That proportion equals the probability we seek to estimate.

Before we proceed to do this procedure in Python, we need a command to *shuffle* an array.

8.14 Shuffling with `rnd.permuted`

In the recipe above, the array `N` has four values:

```
# Numbers representing the slips in the hat.
N = np.array([1, 1, 2, 2])
```

For the physical simulation, we specified that we would shuffle the slips of paper with these numbers, meaning that we would jumble them up into a random order. When we have done this, we will select two slips — say the first two — from the shuffled slips.

As we will be discussing more in various places, this shuffle-then-draw procedure is also called *resampling without replacement*. The *without replacement* idea refers to the fact that, after shuffling, we take a first virtual slip of paper from the shuffled array, and then a second — but we do not replace the first slip of paper into the shuffled array before drawing the second. For

example, say I drew a "1" from `N` for the first value. If I am sampling *without replacement* then, when I draw the next value, the candidates I am choosing from are now "1", "2" and "2", because I have removed the "1" I got as the first value. If I had instead been sampling *with replacement*, then I would put back the "1" I had drawn, and would draw the second sample from the full set of "1", "1", "2", "2".

You can use `rnd.permuted` to shuffle an array into a random order.

Like `rnd.choice`, `rnd.permuted` is a function (actually, a method) of `rnd`, that takes an array as input, and produces a version of the array, where the elements are in random order.

```
# The array N, shuffled into a random order.
shuffled = rnd.permuted(N)
# The "slips" are now in random order.
shuffled
```

```
array([2, 2, 1, 1])
```

See Section 11.4 for some more discussion of shuffling and sampling without replacement.

8.15 Code answers to the cards and pennies problem

> **Note 10:** Notebook: Cards and pennies
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
# Numbers representing the slips in the hat.
N = np.array([1, 1, 2, 2])

# An array in which we will store the result of each trial.
z = np.repeat(['No result yet'], 10000)

for i in range(10000):
    # Shuffle the numbers in N into a random order.
    shuffled = rnd.permuted(N)
```

```
    A = shuffled[0]  # The first slip from the shuffled array.
    B = shuffled[1]  # The second slip from the shuffled array.

    # Set the result of this trial.
    if A == B:
        z[i] = 'Yes'
    else:
        z[i] = 'No'

# How many times did we see "Yes"?
k = np.sum(z == 'Yes')

# The proportion.
kk = k / 10000

print(kk)
```

```
0.337
```

Now let's play the game differently, first picking one card and *putting it back and shuffling* before picking a second card. What are the results now? You can try it with the cards, but here is another program, similar to the last, to run that variation.

```
# The cards / pennies game - but replacing the slip and re-shuffling, before
# drawing again.

# An array in which we will store the result of each trial.
z = np.repeat(['No result yet'], 10000)

for i in range(10000):
    # Shuffle the numbers in N into a random order.
    first_shuffle = rnd.permuted(N)
    # Draw a slip of paper.
    A = first_shuffle[0]  # The first slip.

    # Shuffle again (with all the slips).
    second_shuffle = rnd.permuted(N)
    # Draw a slip of paper.
    B = second_shuffle[0]  # The second slip.

    # Set the result of this trial.
```

```
    if A == B:
        z[i] = 'Yes'
    else:
        z[i] = 'No'

# How many times did we see "Yes"?
k = np.sum(z == 'Yes')

# The proportion.
kk = k / 10000

print(kk)
```

```
0.5072
```

> **i** End of notebook: Cards and pennies
>
> `cards_pennies` starts at Note 10.

Why do you get different results in the two cases? Let's ask the question differently: What is the probability of first picking a black card? Clearly, it is 50-50, or 0.5. Now, if you first pick a black card, what is the probability in the first game above of getting a second black card? There are two red and one black cards left, so now p = 1/3.

But in the second game, what is the probability of picking a second black card if the first one you pick is black? It is still 0.5 because we are *sampling with replacement.*

The probability of picking a second black card *conditional on picking a first black card* in the first game is 1/3, and it is different from the unconditional probability of picking a black card first. But in the second game the probability of the second black card conditional on first picking a black card is the same as the probability of the first black card.

So the reason you lose money if you play the first game at even odds against a carnival game operator is because the conditional probability is different than the original probability.

And an illustrative joke: The best way to avoid there being a live bomb aboard your plane flight is to take an inoperative bomb aboard with you; the probability of one bomb is very low, and by the multiplication rule, *the probability of two bombs is very very low.* Two hundred years ago the same joke was told about the midshipman who, during a battle, stuck his head through a hole in the ship's side that had just been made by an enemy cannon ball because he had heard that the probability of two cannonballs striking in the same place was one in a million.

What's wrong with the logic in the joke? The probability of there being a bomb aboard already, *conditional on* your bringing a bomb aboard, is the same as the conditional probability if you do *not* bring a bomb aboard. Hence you change nothing by bringing a bomb aboard, and do not reduce the probability of an explosion.

8.16 The Commanders again, plus leaving the game early

Let's carry exactly the same process one tiny step further. Assume that if the Commanders win, there is a 0.3 chance you will leave the game early. Now let us ask the probability of a nice day, the Commanders winning, and you leaving early. You should be able to see that this probability can be estimated with three buckets instead of two. Or it can be computed with the multiplication rule as 0.65 * 0.7 * 0.3 = 0.1365 (about 0.14) — the probability of a nice day and a win and you leave early.

The book shows you the formal method — the multiplication rule, in this case — for several reasons: 1) Simulation is weak with very low probabilities, e.g. P(50 heads in 50 throws). But — a big but — *statistics* and probability is seldom concerned with very small probabilities. Even for games like poker, the orders of magnitude of 5 aces in a wild game with joker, or of a royal flush, matter little. 2) The multiplication rule is wonderfully handy and convenient for quick calculations in a variety of circumstances. A back-of-the-envelope calculation can be quicker than a simulation. And it can also be useful in situations where the probability you will calculate will be very small, in which case simulation can require considerable computer time to be accurate. (We will shortly see this point illustrated in the case of estimating the rate of transmission of AIDS by surgeons.) 3) It is useful to know the theory so that you are able to talk to others, or if you go on to other courses in the mathematics of probability and statistics.

The multiplication rule also has the drawback of sometimes being confusing, however. If you are in the slightest doubt about whether the circumstances are correct for applying it, you will be safer to perform a simulation as we did earlier with the Commanders, though in practice you are likely to simulate with the aid of a computer program, as we shall see shortly. So use the multiplication rule only when there is no possibility of confusion. Usually that means using it only when the events under consideration are independent.

Notice that the same multiplication rule gives us the probability of *any particular sequence* of hits and misses — say, a miss, then a hit, then a hit if the probability of a single miss is 2/3. Among the 2/3 of the trials with misses on the first shot, 1/3 will next have a hit, so 2/3 x 1/3 equals the probability of a miss then a hit. Of those 2/9 of the trials, 1/3 will then have a hit, or 2/3 x 1/3 x 1/3 = 2/27 equals the probability of the sequence miss-hit-hit.

The multiplication rule is very useful in everyday life. It fits closely to a great many situations such as "What is the chance that it will rain (.3) and that (if it does rain) the plane will not fly (.8)?" Hence the probability of your not leaving the airport today is 0.3 x 0.8 = 0.24.

9 Probability Theory Part I (continued)

9.1 The special case of independence

A key concept in probability and statistics is that of the independence of two events in which we are interested. Two events are said to be "independent" when one of them does not have any apparent relationship to the other. If I flip a coin that I know from other evidence is a fair coin, and I get a head, the chance of then getting *another* head is still 50-50 (one *in* two, or one *to* one.) And, if I flip a coin ten times and get heads the first nine times, the probability of getting a head on the tenth flip is *still* 50-50. Hence the concept of independence is characterized by the phrase "The coin has no memory." (Actually the matter is a bit more complicated. If you had previously flipped the coin many times and knew it to be a fair coin, then the odds would still be 50-50, even after nine heads. But, if you had never seen the coin before, the run of nine heads might reasonably make you doubt that the coin was a fair one.)

In the Washington Commanders example above, we needed a different set of buckets to estimate the probability of a *nice* day plus a win, and of a *nasty* day plus a win. But what if the Commanders' chances of winning are the same whether the day is nice or nasty? If so, we say that the chance of winning is *independent* of the kind of day. That is, in this special case,

$$P(\text{win} \mid \text{nice day}) = P(\text{win} \mid \text{nasty day}) \text{ and } P(\text{nice day and win})$$

$$= P(\text{nice day}) * P(\text{winning} \mid \text{nice day})$$

$$= P(\text{nice day}) * P(\text{winning})$$

> **Note**
>
> See section Section 8.13 for an explanation of this notation.

In this case we need only one set of two buckets to make all the estimates.

Independence means that the elements are drawn from 2 or more *separate sets of possibilities*. **That is,** $P(A|B) = P(A|\hat{B}) = P(A)$ **and vice versa**.

In other words, if the occurrence of the first event does not change this probability that the second event will occur, then the events are independent.

Another way to put the matter: Events A and B are said to be independent of each other if *knowing* whether A occurs does not change the probability that B will occur, and vice versa. If knowing whether A does occur alters the probability of B occurring, then A and B are dependent.

If two events are independent, the multiplication rule simplifies to $P(A \text{ and } B) = P(A) * P(B)$. I'll repeat once more: **This rule is simply a mathematical shortcut, and one can make the desired estimate by simulation.**

Also again, if two events are *not* independent — that is, if $P(A|B)$ is not equal to $P(A)$ because $P(A)$ is dependent upon the occurrence of B, then the formula to be used now is, $P(A \text{ and } B) = P(A|B) * P(B)$, which is sufficiently confusing that you are probably better off with a simulation.

What about if each of the probabilities is dependent on the other outcome? There is no easy formulaic method to deal with such a situation.

People commonly make the mistake of treating independent events as non-independent, perhaps from superstitious belief. After a long run of blacks, roulette gamblers say that the wheel is "due" to come up red. And sportswriters make a living out of interpreting various sequences of athletic events that occur by chance, and they talk of teams that are "due" to win because of the "Law of Averages." For example, if Barry Bonds goes to bat four times without a hit, all of us (including trained statisticians who really know better) *feel* that he is "due" to get a hit and that the probability of his doing so is very high — higher that is, than his season's average. The so-called "Law of Averages" implies no such thing, of course.

Events are often dependent in subtle ways. A boy may telephone one of several girls chosen at random. But, if he calls the same girl *again* (or if he does *not* call her again), the second event is not likely to be independent of the first. And the probability of his calling her is *different* after he has gone out with her once than before he went out with her.

As noted in the section above, events A and B are said to be independent of each other if the *conditional probabilities of A and B remain the same.* And the conditional probabilities remain the same if sampling is conducted *with replacement.*

Let's now re-consider the multiplication rule with the special but important case of independence.

9.1.1 Example: Four Events in a Row — The Multiplication Rule

Assume that we want to know the probability of four successful archery shots in a row, where the probability of a success on a given shot is .25.

Instead of simulating the process with resampling trials we can, if we wish, arrive at the answer with the "multiplication rule." This rule says that the probability that *all* of a given number of *independent events* (the successful shots) will occur (four out of four in this case) is the *product* of their individual probabilities — in this case, 1/4 x 1/4 x 1/4 x 1/4 = 1/256. If in doubt about whether the multiplication rule holds in any given case, however, you may check by resampling simulation. For the case of four daughters in a row, assuming that the probability of a girl is .5, the probability is 1/2 x 1/2 x 1/2 x 1/2 = 1/16.

Better yet, we'd use the more exact probability of getting a girl: 100/206, and multiply out the result as $(100/206)^4$. An important point here, however: we have estimated the probability of a *particular* family having four daughters as 1 in 16 — that is, odds of 15 to 1. But note well: This is a very different idea from stating that the odds are 15 to 1 against *some* family's having four daughters in a row. In fact, as many families will have four girls in a row as will have boy-girl-boy-girl in that order or girl-boy-girl-boy or *any other series* of four children. The chances against any particular series is the same — 1 in 16 — and one-sixteenth of all four-children families will have each of these series, on average. This means that if your next-door neighbor has four daughters, you cannot say how much "out of the ordinary" the event is. It is easy to slip into unsound thinking about this matter.

Why do we *multiply* the probabilities of the independent simple events to learn the probability that they will occur jointly (the composite event)? Let us consider this in the context of three basketball shots each with 1/3 probability of hitting.

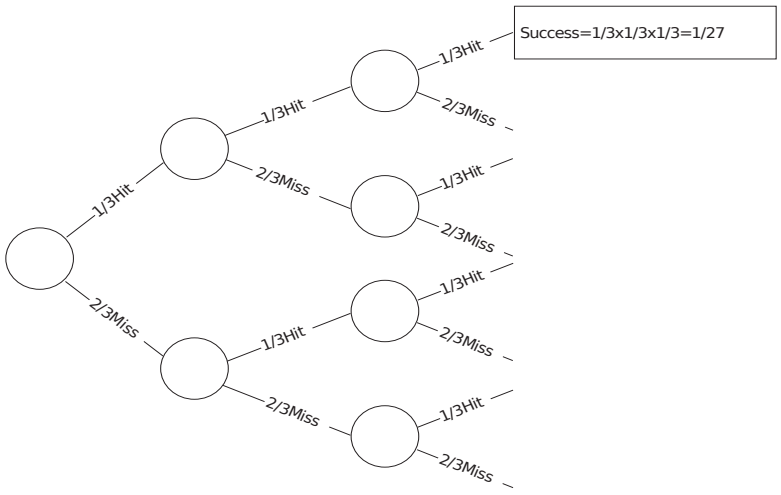


Figure 9.1: Tree Diagram for 3 Basketball Shots, Probability of a Hit is 1/3

Figure 9.1 is a tree diagram showing a set of sequential simple events where each event is conditional upon a prior simple event. Hence every probability after the first is a *conditional* probability.

In Figure 9.1, follow the top path first. On approximately one-third of the occasions, the first shot will hit. *Among that third* of the first shots, roughly a third will *again* hit on the second shot, that is, 1/3 of 1/3 or 1/3 x 1/3 = 1/9. The top path makes it clear that in 1/3 x 1/3 = 1/9 of the trials, *two* hits in a row will occur. Then, of the 1/9 of the total trials in which two hits in a row occur, about 1/3 will go on to a *third* hit, or 1/3 x 1/3 x 1/3 = 1/27. Remember that we are dealing here with independent events; regardless of whether the player made his first two shots, the probability is still 1 in 3 on the third shot.

9.2 The addition of probabilities

Back to the Washington Redskins again. You ponder more deeply the possibility of a nasty day, and you estimate with more discrimination that the probability of snow is .1 and of rain it is .2 (with .7 of a nice day). Now you wonder: What is the probability of a rainy day *or* a nice day?

To find this probability by simulation:

1. Put 7 blue balls (nice day), 1 black ball (snowy day) and 2 gray balls (rainy day) into a bucket. You want to know the probability of a blue *or* a gray ball. To find this probability:
2. Draw one ball and record "yes" if its color is blue or gray, "no" otherwise.
3. Repeat step 1 perhaps 200 times.
4. Find the proportion of "yes" trials.

This procedure certainly will do the job. And simulation may be unavoidable when the situation gets more complex. But in this simple case, you are likely to see that you can compute the probability by *adding* the .7 probability of a nice day and the .2 probability of a rainy day to get the desired probability. This procedure of formulaic deductive probability theory is called the *addition rule*. ## The addition rule

The addition rule applies to *mutually exclusive* outcomes — that is, the case where if one outcome occurs, the other(s) cannot occur; one event implies the absence of the other when events are mutually exclusive. Green and red coats are mutually exclusive if you never wear more than one coat at a time. If there are only two possible mutually-exclusive outcomes, the outcomes are *complementary*. It may be helpful to note that mutual exclusivity equals total dependence; if one outcome occurs, the other cannot. Hence we write formally that

$$\text{If} P(A \text{ and } B) = 0 \text{ then}$$

$$P(A \text{ or } B) = P(A) + P(B)$$

An outcome and its *absence* are mutually exclusive, and their probabilities add to unity.

$$P(A) + P(\hat{}A) = 1$$

Examples include a) rain and no rain, and b) if $P(\text{sales} > \$1 \text{ million}) = 0.2$, then $P(\text{sales} <= \$1 \text{ million}) = 0.8$.

As with the multiplication rule, the addition rule can be a useful shortcut. The answer can always be obtained by simulation, too.

We have so far implicitly assumed that a rainy day and a snowy day are mutually exclusive. But that need not be so; both rain and snow can occur on the same day; if we take this possibility into account, we cannot then use the addition rule.

Consider the case in which seven days in ten are nice, one day is rainy, one day is snowy, *and* one day is *both* rainy and snowy. What is the chance that it will be either nice or snowy? The procedure is just as before, except that some rainy days are included because they are also snowy.

When A and B are *not* mutually exclusive — when it is possible that the day might be both rainy and snowy, or you might wear both red and green coats on the same day, we write (in the latter case) P(red and green coats) > 0, and the appropriate formula is

$$P(\text{red or green}) = P(\text{red}) + P(\text{green}) - P(\text{red and green})`$$

In this case as in much of probability theory, the simulation for the case in which the events are not mutually exclusive is no more complex than when they are mutually exclusive; indeed, if you simulate you never even need to know the concept of mutual exclusivity or inquire whether that is your situation. In contrast, the appropriate formula for non-exclusivity is more complex, and if one uses formulas one must inquire into the characteristics of the situation and decide which formula to apply depending upon the classification; if you classify wrongly and therefore apply the wrong formula, the result is a wrong answer.

To repeat, the addition rule only works when the probabilities you are adding are *mutually exclusive* — that is, when the two cannot occur together.

The multiplication and addition rules are as different from each other as mortar and bricks; both, however, are needed to build walls. The multiplication rule pertains to a single outcome composed of two or more elements (e.g. weather, and win-or-lose), whereas the addition rule pertains to two or more possible outcomes for one element. Drawing from a card deck (with replacement) provides an analogy: the addition rule is like one draw with two or more possible cards of interest, whereas the multiplication rule is like two or more cards being drawn with one particular "hand" being of interest.

9.3 Theoretical devices for the study of probability

It may help you to understand the simulation approach to estimating composite probabilities demonstrated in this book if you also understand the deductive formulaic approach. So we'll say a bit about it here.

The most fundamental concept in theoretical probability is the list of events that may occur, together with the probability of each one (often arranged so as to be equal probabilities). This is the concept that Galileo employed in his great fundamental work in theoretical probability about four hundred years ago when a gambler asked Galileo about the chances of getting a nine rather than a ten in a game of three dice (though others such as Cardano had tackled the subject earlier).[1]

Galileo wrote down all the possibilities in a tree form, a refinement for mapping out the sample space.

Galileo simply displayed the events themselves — such as "2," "4," and "4," making up a total of 10, a specific event arrived at in a specific way. Several different events can lead to a 10 with three dice. If we now consider each of these events, we arrive at the concept of the *ways* that a total of 10 can arise. We ask the number of ways that an outcome can and cannot occur. (See the paragraph above). This is equivalent both operationally and linguistically to the *paths* in (say) the quincunx device or Pascal's Triangle which we shall discuss shortly.

A *tree* is the most basic display of the paths in a given situation. Each branch of the tree — a unique path from the start on the left-hand side to the endpoint on the right-hand side — contains the *sequence of all the elements* that make up that event, in the order in which they occur. The right-hand ends of the branches constitute a list of the outcomes. That list includes all possible permutations — that is, it distinguishes among outcomes by the orders in which the particular die outcomes occur.

[1] Here is another example of the confusion on such matters, this one written by Charles Cotton (part-author of The Compleat Angler) in 1674:

> Now six and eight one would think should admit of no difference in advantage with seven, but if you will rightly consider the case, and be so vain to make trial thereof, you will find a great advantage in seven over six and eight. How can that be you will say, hath not six, seven and eight equal chances? For example, in six, quarter deuce and two treys; in eight, six deuce, cinque trey, and two quarters; and hath not seven three as aforesaid? It is confest; but pray consider the disadvantage in the doublets, two treys and two quarters, and you will find that six deuce is sooner thrown than two quarters, and so consequently, cinque Ace or quarter deuce sooner than two treys: I saw an old rook once take up a young fellow in a tavern, upon this very score: the bargain was made that the rook should have seven always and the young gentleman six, and throw continually; agreed to play they went, the rook got the first day ten pound, the next day the like sum; and so for six days together losing in all threescore pounds; notwithstanding the gentleman, I am confident, had square dice, and threw them always himself.

Cited in (Bulmer 1979, 20–21)

9.4 The Concept of Sample Space

The formulaic approach begins with the idea of *sample space*, which is the set of all possible outcomes of the "experiment" or other situation that interests us. Here is a formal definition from Goldberg (1986, 46):

> A *sample space S* associated with a real or conceptual experiment is a set such that (1) each element of S denotes an outcome of the experiment, and (2) any performance of the experiment results in an outcome that corresponds to one and only one element of S.

Because the sum of the probabilities for all the possible outcomes in a given experimental trial is unity, the sum of all the events in the sample space (S) = 1.

Early on, people came up with the idea of estimating probabilities by arraying the possibilities for, and those against, the event occurring. For example, the coin could fall in three ways — head, tail, or on its side. They then speedily added the qualification that the possibilities in the list must have an equal chance, to distinguish the coin falling on its side from the other possibilities (so ignore it). Or, if it is impossible to make the probabilities equal, make special allowance for inequality. Working directly with the sample space is the *method of first principles*. The idea of a list was refined to the idea of sample space, and "for" and "against" were refined to the "success" and "failure" elements among the total elements.

The concept of sample space raises again the issue of how to estimate the simple probabilities. While we usually can estimate the probabilities accurately in gambling games because we ourselves construct the games and therefore control the probabilities that they produce, we have much less knowledge of the structures that underlie the important problems in life — in science, business, the stock market, medicine, sports, and so on. We therefore must wrestle with the issue of what probabilities we should include in our theoretical sample space, or in our experiments. Often we proceed by choosing as an analogy a physical "model" whose properties we know and which we consider to be appropriate — such as a gambling game with coins, dice, cards. This model becomes our idealized setup. But this step makes crystal-clear that judgment is heavily involved in the process, because choosing the analogy requires judgment.

A *Venn diagram* is another device for displaying the elements that make up an event. But unlike a tree diagram, it does not show the sequence of those elements; rather, it shows the *extent of overlap among various classes of elements*. A Venn diagram expresses by *areas* (especially rectangular Venn diagrams) the numbers at the end of the branches in a tree.

Pascal's Triangle is still another device. It aggregates the last permutation branches in the tree into *combinations* — that is, without distinguishing by order. It shows analytically (by tracing them) the various paths that lead to various combinations.

The study of the mathematics of probability is the study of calculational shortcuts to do what tree diagrams do. If you don't care about the shortcuts, then you don't need the formal mathematics--though it may improve your mathematical insight (or it may not). The resampling method dispenses not only with the shortcuts but also with the entire counting of points in the sample space.

10 Two puzzles and more tools

10.1 Introduction

In the next chapter we will deal with some more involved problems in *probability*, as a preparation for *statistics*, where we use reasoning from probability to draw conclusions about a world like our own, where variation often appears to be more or less random.

Before we get down to the business of complex probabilistic problems in the next few chapters, let's consider a couple of peculiar puzzles. These puzzles allow us to introduce some more of the key tools in Python for Monte Carlo resampling, and show the power of such simulation to help solve, and then reason about, problems in probability.

But, before we get to the puzzles, we will need some more elements of Python.

10.2 Selecting elements from arrays with slicing

> Note 11: Notebook: Selecting elements by slicing
>
> - Download notebook
> - Interact

As you saw in Section 6.6, we do *indexing* when we put square brackets following a value that is a container, such as an array. Inside the square brackets we put another value to specify which elements we want to fetch from the container.

We will use the `some_numbers` array as our container for indexing:

```
import numpy as np

some_numbers = np.array([3, 1, 4, 1, 5, 9, 2, 6])
```

In the indexing expression below, we have the array `some_numbers`, followed by the value `3` inside the square brackets, telling Python that we want the fourth value from the `some_numbers` container.

```
# Indexing with an integer inside the square brackets.
some_numbers[3]
```

```
np.int64(1)
```

In the example above, we put an integer (a whole number) inside the square brackets, to specify the position of the element we want to fetch from the container.

We can also put something called a *slice* inside the square brackets.

A slice specifies a *range* of elements to fetch from the container.

We can form a *slice* with an integer, followed by a colon (`:`), followed by another integer. The first integer specifies the *start* position; this is the position of the first element we want. There follows a colon. Read the colon as "up to, but not including". Finally, we have an integer that gives the *stop* position. The slice, thus specified, asks Python to give us all the elements from (including) the *start* position, up to, *but not including* the *stop* position.

For example, here we index with a slice having *start* of 1 (offset 1 from the start, the position of the second element). The *stop* is 5, meaning we should go up to, but not include the element at position 5 (the sixth element). The result is another array, that has the elements of `some_numbers` from positions 1 through 4:

```
# Indexing with a slice (an expression including a colon).
some_numbers[1:5]
```

```
array([1, 4, 1, 5])
```

In fact, we can omit the value before the colon (the *start* value), and Python will assume we mean 0. This indexing expression fetches the elements at position 0 through 3:

```
# Indexing with a slice that omits the first (start) value.  Python assumes 0.
some_numbers[:4]
```

```
array([3, 1, 4, 1])
```

The expression above then means "get all the elements up to (not including) position 4", or equivalently, "get the first four elements of the array".

We can also omit the stop value. Python assumes we mean one past the last position in the array. This is also the `len` of the array. By taking this stop value, the slice selects the elements starting at the *start* element, through to (including) the last element of the array:

```
# Indexing with a slice that omits the last (stop) value.  Python assumes
# we mean one past the last position in the array.
some_numbers[3:]
```

```
array([1, 5, 9, 2, 6])
```

The slice in the example above therefore means "get all the elements from position 3 to the end of the array".

Actually, we can even omit the start *and* the stop values, leaving just the colon. As you might expect, Python assumes 0 as the start, and one past the end as the stop, so the colon on its own means "return all the elements in the array". As usual with slicing, we get a new array with the chosen elements.

```
# Just the colun, meaning "all the elements in the array".
some_numbers[:]
```

```
array([3, 1, 4, 1, 5, 9, 2, 6])
```

> End of notebook: Selecting elements by slicing
>
> `selecting_by_slicing` starts at Note 11.

10.3 The treasure fleet recovered

As promised, we have now arrived at the first of the probability puzzles.

This is a classic problem in probability:[1]

[1] The treasure fleet problem is a restatement of a problem that Joseph Bertrand posed early in the 19th century.) Here is a variation from (Goldberg 1986, 99):

> Three identical boxes each contain two coins. In one box both are pennies, in the second both are nickels, and in the third there is one penny and one nickel.
>
> A man chooses a box at random and takes out a coin. If the coin is a penny, what is the probability that the other coin in the box is also a penny?

A Spanish treasure fleet of three ships was sunk at sea off Mexico. One ship had a chest of gold forward and another aft, another ship had a chest of gold forward and a chest of silver aft, while a third ship had a chest of silver forward and another chest of silver aft. Divers just found one of the ships and a chest of gold in it, but they don't know whether it was from forward or aft. They are now taking bets about whether the other chest found on the same ship will contain silver or gold. What are fair odds?

These are the logical steps one may distinguish in arriving at a correct answer with deductive logic (portrayed in Figure 10.1).

1. Postulate three ships — Ship I with two gold chests (G-G), ship II with one gold and one silver chest (G-S), and ship III with S-S. (Choosing notation might well be considered one or more additional steps.)
2. Assert equal probabilities of each ship being found.
3. Step 2 implies equal probabilities of being found for each of the six chests.
4. Fact: Diver finds a chest of gold.
5. Step 4 implies that S-S ship III was not found; hence remove it from subsequent analysis.
6. Three possibilities: 6a) Diver found chest I-Ga, 6b) diver found I-Gb, 6c) diver found II-Gc.

   From step 2, the cases a, b, and c in step 6 have equal probabilities.
7. If possibility 6a is the case, then the other chest is I-Gb; the comparable statements for cases 6b and 6c are I-Ga and II-S.
8. From steps 6 and 7: From equal probabilities of the three cases, and no other possible outcome, $P(6a) = 1/3$, $P(6b) = 1/3$, $P(6c) = 1/3$.
9. So $P(G) = P(6a) + P(6b) = 1/3 + 1/3 = 2/3$.

See Figure 10.1.

The following simulation arrives at the correct answer.

1. Write "Gold" on three pieces of paper and "Silver" on three pieces of paper. These represent the chests.
2. Get three buckets each with two pieces of paper. Each bucket represents a ship, each piece of paper represents a chest in that ship. One bucket has two pieces of paper with "Gold" written on them; one has pieces of paper with "Gold" and "Silver", and one has "Silver" and "Silver".
3. Choose a bucket at random, to represent choosing a ship at random.
4. Shuffle the pieces of paper in the bucket and pick one, to represent choosing the first chest from that ship at random.

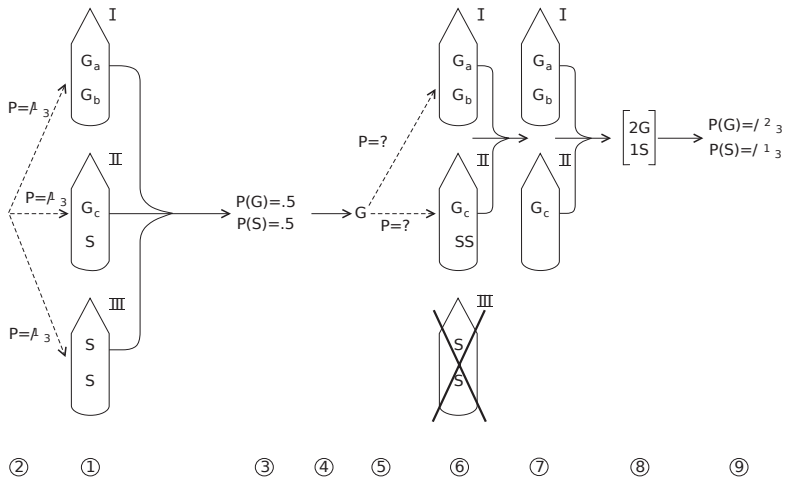


Figure 10.1: Ships with Gold and Silver

5. If the piece of paper says "Silver", the first chest we found in this ship was silver, and we stop the trial and make no further record. If "Gold", continue.
6. Get the second piece of paper from the bucket, representing the second chest on the chosen ship. Record whether this was "Silver" or "Gold" on the scoreboard.
7. Repeat steps (3 - 6) many times, and calculate the proportion of "Gold"s on the scoreboard. (The answer should be about $\frac{2}{3}$.)

Here is a notebook simulation with Python:

> **Note 12:** Notebook: Ships with gold and silver
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
# The 3 buckets.  Each bucket represents a ship.  Each has two chests.
bucket1 = ['Gold', 'Gold']  # Chests in first ship.
bucket2 = ['Gold',  'Silver']  # Chests in second ship.
bucket3 = ['Silver', 'Silver']  # Chests in third ship.
```

```
# For each trial, we will have one of three states:
#
```

```
# 1. When opening the first chest, it did not contain gold.
#    We will reject these trials, since they do not match our
#    experiment description.
# 2. Gold was found in the first and the second chest.
# 3. Gold was found in the first, but silver in the second chest.
#
# We need a placeholder value for all trials, and will make that
# "No gold in chest 1, chest 2 never opened".
second_chests = np.repeat(['No gold in chest 1, chest 2 never opened'], 10000)

for i in range(10000):
    # Select a ship at random from the three ships.
    ship_no = rnd.choice([1, 2, 3])
    # Get the chests from this ship (represented by a bucket).
    if ship_no == 1:
        bucket = bucket1
    if ship_no == 2:
        bucket = bucket2
    if ship_no == 3:
        bucket = bucket3

    # We shuffle the order of the chests in this ship, to simulate
    # the fact that we don't know which of the two chests we have
    # found first, forward or aft.
    shuffled = rnd.permuted(bucket)

    if shuffled[0] == 'Gold':  # We found a gold chest first.
        # Store whether the Second chest was silver or gold.
        second_chests[i] = shuffled[1]

    # End loop, go back to beginning.

# Number of times we found gold in the second chest.
n_golds = np.sum(second_chests == 'Gold')
# Number of times we found silver in the second chest.
n_silvers = np.sum(second_chests == 'Silver')
# As a ratio of golds to all second chests (where the first was gold).
print(n_golds / (n_golds + n_silvers))
```

```
0.6625368731563421
```

> **End of notebook: Ships with gold and silver**
>
> `gold_silver_ships` starts at Note 12.

In the code above, we have first chosen the ship number at random, and then used a set of `if ...` statements to get the pair of chests corresponding to the given ship. There are simpler and more elegant ways of writing this code, but they would need some Python features that we haven't covered yet.[2]

10.4 Back to Boolean arrays

The code above implements the procedure we might well use if we were simulating the problem physically. We do a trial, and we record the result. We do this on a piece of paper if we are doing a physical simulation, and in the `second_chests` array in code.

Finally we tally up the results. If we are doing a physical simulation, we go back over the all the trial results and counting up the "Gold" and "Silver" outcomes. In code we use the comparisons `== 'Gold'` and `== 'Silver'` to find the trials of interest, and then count them up with `np.sum`.

[2] Although we have used multiple `if` statements to set the chests for each ship in the code for the ship problem, it would have been more elegant to store the definitions of ships and chests as a list of lists. We haven't covered these, but to give you a taste of what that would look like, for a single trial.

```
import numpy as np
rnd = np.random.default_rng()

# We define the ships and their respective chests, using a list of lists.
# Remember, list has elements, where the elements can be any type of value.
# In this case, the elements in the list are themselves - lists.
ship_chests = [  # Create list, with lists as elements.
    ['Gold', 'Gold'],  # List for ship 1; element 0 of the list of lists.
    ['Gold', 'Silver'],  # List for ship 2; element 1 of the list of lists.
    ['Silver', 'Silver']  # Ship 3; element 2.
]
# Choose a ship at random.
ship_no = rnd.choice([0, 1, 2])

# Get the corresponding chests for this ship.
chests = ship_chests[ship_no]
chests
```

```
['Gold', 'Silver']
```

Boolean (logical) arrays are a fundamental tool in Python, and we will use them in nearly all our simulations.

Here is a remind of how those arrays work.

First, let's slice out the first 10 values of the `second_chests` trial-by-trial results tally from the simulation above:

```
# Get values at positions 0 through 9 (up to, but not including position 10)
first_10_chests = second_chests[:10]
first_10_chests
```

```
array(['Silver', 'No gold in chest 1, chest 2 never opened',
       'No gold in chest 1, chest 2 never opened', 'Gold', 'Gold',
       'No gold in chest 1, chest 2 never opened', 'Gold', 'Gold',
       'No gold in chest 1, chest 2 never opened', 'Gold'], dtype='<U40')
```

Before we started the simulation, we set `second_chests` to contain 10,000 strings, where each string was "No gold in chest 1, chest 2 never opened". In the simulation, we check whether there was gold in the first chest, and, if not, we don't change the value in `second_chest`, and the value remains as "No gold in chest 1, chest 2 never opened".

Only if there was gold in the first chest, do we go on to check whether the second chest contains silver or gold. Therefore, we only set a new value in `second_chests` where there was gold in the first chest.

Now let's show the effect of running a comparison on `first_10_chests`:

```
were_gold = (first_10_chests == 'Gold')
were_gold
```

```
array([False, False, False,  True,  True, False,  True,  True, False,
        True])
```

> **Parentheses and Boolean comparisons**
>
> Notice the round brackets (parentheses) around `(first_10_chests == 'Gold')`. In this particular case, we would get the same result without the parentheses, so the paretheses are optional— although see below for an example where the they are not optional. In general, you will see we put parentheses around all expressions that generate Boolean arrays, and we recommend you do too. It is good habit to get into, to make it clear that this is an expression that generates a value.

The `== 'Gold'` *comparison* is asking a question. It is asking that question of an array, and the array contains multiple values. NumPy treats this comparison as asking the question *of each element in the array*. We get an answer for the question *for each element*. The answer for position 0 is `True` if the element at position 0 is equal to `'Gold'` and `False` otherwise, and so on, for positions 1, 2 and so on. We started with 10 strings. After the comparison `== 'Gold'` we have 10 Boolean values, where a Boolean value can either be `True` or `False`.

Now we have an array with `True` for the "Gold" results and `False` otherwise, we can count the number of "Gold" results by using `np.sum` on the array. As you remember (Section 5.14) `np.sum` counts `True` as 1 and `False` as 0, so the sum of the Boolean array is just the number of `True` values in the array — the count that we need.

```
# The number of True values - so the number of "Gold" chests.
np.sum(were_gold)
```

```
np.int64(5)
```

10.5 Boolean arrays and another take on the ships problem

If we are doing a physical simulation, we usually want to finish up all the work for the trial during the trial, so we have one outcome from the trial. This makes it easier to tally up the results in the end.

We have no such constraint when we are using code, so it is sometimes easier to record several results from the trial, and do the final combinations and tallies at the end. We will show you what we mean with a slight variation on the two-ships code you saw above.

> **Note 13: Notebook: Another approach to ships with gold and silver**
>
> - Download notebook
> - Interact

Notice that the first part of the code is identical to the first approach to this problem. There are two key differences — see the comments for an explanation.

```
import numpy as np
rnd = np.random.default_rng()
```

```
# The 3 buckets, each representing two chests on a ship.
# As before.
bucket1 = ['Gold', 'Gold']  # Chests in first ship.
bucket2 = ['Gold',  'Silver']  # Chests in second ship.
bucket3 = ['Silver', 'Silver']  # Chests in third ship.
```

```
# Here is where the difference starts.  We are now going to fill in
# the result for the first chest _and_ the result for the second chest.
#
# Later we will fill in all these values, so the string we put here
# does not matter.

# Whether the first chest was Gold or Silver.
first_chests = np.repeat(['To be announced'], 10000)
# Whether the second chest was Gold or Silver.
second_chests = np.repeat(['To be announced'], 10000)

for i in range(10000):
    # Select a ship at random from the three ships.
    # As before.
    ship_no = rnd.choice([1, 2, 3])
    # Get the chests from this ship.
    # As before.
    if ship_no == 1:
        bucket = bucket1
    if ship_no == 2:
        bucket = bucket2
    if ship_no == 3:
        bucket = bucket3

    # As before.
    shuffled = rnd.permuted(bucket)

    # Here is the big difference - we store the result for the first and second
    # chests.
    first_chests[i] = shuffled[0]
    second_chests[i] = shuffled[1]

# End loop, go back to beginning.

# We will do the calculation we need in the next cell.  For now
# just display the first 10 values.
```

```
ten_first_chests = first_chests[:10]
print('The first 10 values of "first_chests:', ten_first_chests)
```

```
The first 10 values of "first_chests: ['Gold' 'Silver' 'Silver' 'Gold' 'Gold' 'Silver' 'Gold
 'Gold']
```

```
ten_second_chests = second_chests[:10]
print('The first 10 values of "second_chests', ten_second_chests)
```

```
The first 10 values of "second_chests ['Silver' 'Gold' 'Silver' 'Gold' 'Gold' 'Silver' 'Gold
 'Gold']
```

In this variant, we recorded the type of first chest for each trial ("Gold" or "Silver"), and the type of second chest of the second chest ("Gold" or "Silver").

We would like to count the number of times there was "Gold" in the first chest *and* "Gold" in the second.

10.6 Combining Boolean arrays

We can do the count we need by *combining* the Boolean arrays with the `&` operator. `&` combines Boolean arrays with a *logical and*. *Logical and* is a rule for combining two Boolean values, where the rule is: the result is `True` if the first value is `True` *and* the second value if `True`.

Here we use the `&` *operator* to combine some Boolean values on the left and right of the operator:

```
True & True   # Both are True, so result is True
```

```
True
```

```
True & False   # At least one of the values is False, so result is False
```

```
False
```

```
False & True   # At least one of the values is False, so result is False
```

```
False
```

```
False & False   # At least one (in fact both) are False, result is False.
```

```
False
```

> **Note 14: `&` and `and` in Python**
>
> In fact Python has another operation to apply this *logical and* operation to values — the `and` operator:
>
> ```
> print(True and True)
> ```
>
> ```
> True
> ```
>
> ```
> print(True and False)
> ```
>
> ```
> False
> ```
>
> ```
> print(False and True)
> ```
>
> ```
> False
> ```
>
> ```
> print(False and False)
> ```
>
> ```
> False
> ```
>
> You will see this `and` operator often in Python code, but it does not work well when combining Numpy *arrays*, so we will use the similar `&` operator, that does work on arrays.

Above you saw that the `==` operator (as in `== 'Gold'`), when applied to arrays, asks the question of every element in the array.

First make the Boolean arrays.

```
ten_first_gold = (ten_first_chests == 'Gold')
print("Ten first == 'Gold'", ten_first_gold)
```

```
Ten first == 'Gold' [ True False False  True  True False  True  True False  True]
```

```
ten_second_gold = (ten_second_chests == 'Gold')
print("Ten second == 'Gold'", ten_second_gold)
```

```
Ten second == 'Gold' [False  True False  True  True False  True  True False  True]
```

Now let us use `&` to combine Boolean arrays:

```
ten_both = (ten_first_gold & ten_second_gold)
ten_both
```

```
array([False, False, False,  True,  True, False,  True,  True, False,
        True])
```

Notice that Python does the comparison *elementwise* — element by element.

You saw that when we did `second_chests == 'Gold'` this had the effect of asking the `== 'Gold'` question of *each element*, so there will be one answer per element in `second_chests`. In that case there was an array to the *left* of `==` and a single value to the *right*. We were comparing an array to a value.

Here we are asking the `&` question of `ten_first_gold` and `ten_second_gold`. Here there is an array to the *left* and an array to the *right*. We are asking the `&` question 10 times, but the first question we are asking is:

```
# First question, giving first element of result.
(ten_first_gold[0] & ten_second_gold[0])
```

```
np.False_
```

The second question is:

```
# Second question, giving second element of result.
(ten_first_gold[1] & ten_second_gold[1])
```

```
np.False_
```

and so on. We have ten elements on *each side*, and 10 answers, giving an array (`ten_both`) of 10 elements. Each element in `ten_both` is the answer to the `&` question for the elements at the corresponding positions in `ten_first_gold` and `ten_second_gold`.

We could also create the Boolean arrays and do the `&` operation all in one step, like this:

```
ten_both = (ten_first_chests == 'Gold') & (ten_second_chests == 'Gold')
ten_both
```

```
array([False, False, False,  True,  True, False,  True,  True, False,
        True])
```

> **Parentheses, arrays and comparisons**
>
> Again you will notice the round brackets (parentheses) around `(ten_first_chests == 'Gold')` and `(ten_second_chests == 'Gold')`. Above, you saw us recommend you always use paretheses around Boolean expressions like this. The parentheses make the code easier to read — but be careful — in this case, we actually *need* the parentheses to make Python do what we want; see the footnote for more detail.[3]

Remember, we wanted the answer to the question: how many trials had "Gold" in the first chest *and* "Gold" in the second. We can answer that question for the first 10 trials with `np.sum`:

```
n_ten_both = np.sum(ten_both)
n_ten_both
```

```
np.int64(5)
```

We can answer the same question for *all* the trials, in the same way:

[3] We warned that we need parentheses around our `&` expressions to get the result we want. We would add the parentheses in any case, as good practice, but here we also *need* the parentheses in `(ten_first_chests == 'Gold') & (ten_second_chests == 'Gold')`. Remember *operator precedence*; for example, the multiply operator `*` has *higher precedence* than the operator `+`, so `3 + 5 * 2` is equal to `3 + (5 * 2)` = 13. If we want to do addition before multiplication, we use parentheses to tell Python the order it should use: `(3 + 5) * 2` = 16.

The same applies for the two operators `==` and `&` here. In fact `&` has a higher precedence than `==`. This means that, if we write the expression without parentheses — `ten_first_chests == 'Gold' & ten_second_chests == 'Gold'` — because of operator precedence, Python takes this to mean `ten_first_chests == ('Gold' & ten_second_chests) == 'Gold'`. Python does not know what to do with `'Gold' & ten_second_chests` and generates an error of form `'bitwise_and' not supported for the input types`. The error tells you that Python does not know how to apply `&` (`'bitwise_and'`) to the string `'Gold'` and the array `ten_second_chests`.

This is the same error you would get for running the code `'Gold' & ten_second_chests` on its own.

The point to take away is, that when you are using `&` to combine Boolean arrays in Python, remember operator precedence, and, when in doubt, put parentheses around the expressions on either side of `&`, as here.

```
first_gold = (first_chests == 'Gold')
second_gold = (second_chests == 'Gold')
n_both_gold = np.sum(first_gold & second_gold)
n_both_gold
```

```
np.int64(3369)
```

We could also do the same calculation all in one line:

```
# Notice the parentheses - we need these - see above.
n_both_gold = np.sum((first_chests == 'Gold') & (second_chests == 'Gold'))
n_both_gold
```

```
np.int64(3369)
```

We can then count all the ships where the first chest was gold:

```
n_first_gold = np.sum(first_chests == 'Gold')
n_first_gold
```

```
np.int64(5085)
```

The final calculation is the proportion of second chests that are gold, given the first chest was also gold:

```
p_g_given_g = n_both_gold / n_first_gold
p_g_given_g
```

```
np.float64(0.6625368731563421)
```

Of course we won't get exactly the same results from the two simulations, in the same way that we won't get exactly the same results from any two runs of the same simulation, because of the random values we are using. But the logic for the two simulations are the same, and we are doing many trials (10,000), so the results will be very similar.

> **i** End of notebook: Another approach to ships with gold and silver
>
> `gold_silver_booleans` starts at Note 13.

10.7 The Monty Hall problem

The Monty Hall Problem is a puzzle in probability that is famous for its deceptive simplicity. It has its own long Wikipedia page: https://en.wikipedia.org/wiki/Monty_Hall_problem.

Here is the problem in the form it is best known; a letter to the columnist Marilyn vos Savant, published in Parade Magazine (1990):

> Suppose you're on a game show, and you're given the choice of three doors. Behind one door is a car, behind the others, goats. You pick a door, say #1, and the host, who knows what's behind the doors, opens another door, say #3, which has a goat. He says to you, "Do you want to pick door #2?" Is it to your advantage to switch your choice of doors?

In fact the first person to propose (and solve) this problem was Steve Selvin, a professor of public health at the University of California, Berkeley (Selvin 1975).

Most people, including at least one of us, your humble authors, quickly come to the wrong conclusion. The most common but incorrect answer is that it will make no difference if you switch doors or stay with your original choice. The obvious intuition is that, after Monty opens his door, there are two doors that might have the car behind them, and therefore, there is a 50% chance it will be behind any one of the two. It turns out that answer is wrong; you will double your chances of winning by switching doors. Did you get the answer right?

If you got the answer wrong, you are in excellent company. As you can see from the commentary in Savant (1990), many mathematicians wrote to Parade magazine to assert that the (correct) solution was wrong. Paul Erdős was one of the most famous mathematicians of the 20th century; he could not be convinced of the correct solution until he had seen a computer simulation (Vazsonyi 1999), of the type we will do below.

To simulate a trial of this problem, we need to select a door at random to house the car, and another door at random, to be the door the contestant chooses. We number the doors 1, 2 and 3. Now we need two random choices from the options 1, 2 or 3, one for the door with the car, the other for the contestant door. To chose a door for the car, we could throw a die, and chose door 1 if the die shows 1 or 4, door 2 if the die shows 2 or 5, and door 3 for 3 or 6. Then we throw the die again to chose the contestant door.

But throwing dice is a little boring; we have to find the die, then throw it many times, and record the results. Instead we can ask the computer to chose the doors at random.

For this simulation, let us do 25 trials. We ask the computer to create two sets of 25 random numbers from 1 through 3. The first set is the door with the car behind it ("Car door"). The second set have the door that the contestant chose at random ("Our door"). We put these in a table, and make some new, empty columns to fill in later. The first new column is "Monty opens". In due course, we will use this column to record the door that Monty Hall will open on this trial. The last two columns express the outcome. The first is "Stay wins". This has "Yes"

if we win on this trial by sticking to our original choice of door, and "No" otherwise. The last column is "Switch wins". This has "Yes" if we win by switching doors, and "No" otherwise. See table Table 10.1).

Table 10.1: 25 simulations of the Monty Hall problem

	Car door	Our door	Monty opens	Stay wins	Switch wins
1	3	3			
2	3	1			
3	1	3			
4	1	1			
5	2	3			
6	2	1			
7	2	2			
8	1	3			
9	1	2			
10	3	1			
11	2	2			
12	3	2			
13	2	2			
14	3	1			
15	1	2			
16	2	1			
17	3	3			
18	3	2			
19	1	1			
20	3	2			
21	2	2			
22	3	1			
23	3	1			
24	1	1			
25	2	3			

In the first trial in Table 10.1), the computer selected door 3 for car, and door 3 for the contestant. Now Monty must open a door, and he cannot open our door (door 3) so he has the choice of opening door 1 or door 2; he chooses randomly, and opens door 2. On this trial, we win if we stay with our original choice, and we lose if we change to the remaining door, door 1.

Now we go the second trial. The computer chose door 3 for the car, and door 1 for our choice. Monty cannot choose our door (door 1) or the door with the car behind it (door 3), so he must open door 2. Now if we stay with our original choice, we lose, but if we switch, we win.

You may want to print out table Table 10.1, and fill out the blank columns, to work through the logic.

After doing a few more trials, and some reflection, you may see that there are two different situations here: the situation when our *initial guess was right*, and the situation where our *initial guess was wrong*. When our initial guess was right, we win by staying with our original choice, but when it was wrong, we always win by switching. The chance of our *initial guess* being correct is 1/3 (one door out of three). So the chances of winning by staying are 1/3, and the chances of winning by switching are 2/3. But remember, you don't need to follow this logic to get the right answer. As you will see below, the resampling simulation shows us that the Switch strategy wins.

Table 10.2 is a version of table Table 10.1 for which we have filled in the blank columns using the logic above.

Table 10.2: 25 simulations of the Monty Hall problem, filled out

	Car door	Our door	Monty opens	Stay wins	Switch wins
1	3	3	1	Yes	No
2	3	1	2	No	Yes
3	1	3	2	No	Yes
4	1	1	2	Yes	No
5	2	3	1	No	Yes
6	2	1	3	No	Yes
7	2	2	3	Yes	No
8	1	3	2	No	Yes
9	1	2	3	No	Yes
10	3	1	2	No	Yes
11	2	2	1	Yes	No
12	3	2	1	No	Yes
13	2	2	1	Yes	No
14	3	1	2	No	Yes
15	1	2	3	No	Yes
16	2	1	3	No	Yes
17	3	3	2	Yes	No
18	3	2	1	No	Yes
19	1	1	2	Yes	No
20	3	2	1	No	Yes
21	2	2	1	Yes	No
22	3	1	2	No	Yes
23	3	1	2	No	Yes
24	1	1	2	Yes	No
25	2	3	1	No	Yes

The proportion of times "Stay" wins in these 25 trials is 0.36. The proportion of times "Switch" wins is 0.64; the Switch strategy wins about twice as often as the Stay strategy.

10.8 Monty Hall with Python

Now you have seen what the results might look like for a physical simulation, you can exercise some of your newly-strengthened Python muscles to do the simulation with code.

> **Note 15: Notebook: The Monty Hall problem**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

The Monty Hall problem has a slightly complicated structure, so we will start by looking at the procedure for one trial. When we have that clear, we will put that procedure into a `for` loop for the simulation.

Let's start with some variables. Let's call the door I choose `my_door`.

We choose that door at random from a sequence of all possible doors. Call the doors 1, 2 and 3 from left to right.

```
# List of doors to chose from.
doors = [1, 2, 3]

# We choose one door at random.
my_door = rnd.choice(doors)

# Show the result
my_door
```

```
np.int64(2)
```

We choose one of the doors to be the door with the car behind it:

```
# One door at random has the car behind it.
car_door = rnd.choice(doors)

# Show the result
car_door
```

```
np.int64(2)
```

Now we need to decide which door Monty will open.

By our set up, Monty cannot open our door (`my_door`). By the set up, he has not opened (and cannot open) the door with the car behind it (`car_door`).

`my_door` and `car_door` might be the same.

So, to get Monty's choices, we want to take all doors (`doors`) and remove `my_door` and `car_door`. That leaves the door or doors Monty can open.

Here are the doors Monty cannot open. Remember, a third of the time `my_door` and `car_door` will be the same, so we will include the same door twice, as doors Monty can't open.

```
cant_open = [my_door, car_door]
cant_open
```

```
[np.int64(2), np.int64(2)]
```

We want to find the remaining doors from `doors` after removing the doors named in `cant_open`.

NumPy has a good function for this, called `np.setdiff1d`. It calculates the *set difference* between two sequences, such as arrays.

The set difference between two sequences is the members that *are* in the first sequence, but are *not* in the second sequence. Here are a few examples of this set difference function in NumPy.

Notice that we are using *lists* as the input (first and second) sequences here. We can use lists or arrays or any other type of sequence in Python. (See Section 7.3.2 for an introduction to lists).

Numpy functions like `np.setdiff1d` always *return* an array.

```
# Members in [1, 2, 3] that are *not* in [1]
# 1, 2, 3, removing 1, if present.
np.setdiff1d([1, 2, 3], [1])
```

```
array([2, 3])
```

```
# Members in [1, 2, 3] that are *not* in [2, 3]
# 1, 2, 3, removing 2 and 3, if present.
np.setdiff1d([1, 2, 3], [2, 3])
```

```
array([1])
```

```
# Members in [1, 2, 3] that are *not* in [2, 2]
# 1, 2, 3, removing 2 and 2 again, if present.
np.setdiff1d([1, 2, 3], [2, 2])
```

```
array([1, 3])
```

This logic allows us to choose the doors Monty can open:

```
montys_choices = np.setdiff1d(doors, [my_door, car_door])
montys_choices
```

```
array([1, 3])
```

Notice that `montys_choices` will only have one element left when `my_door` and `car_door` were different, but it will have two elements if `my_door` and `car_door` were the same.

Let's play out those two cases:

```
my_door = 1  # For example.
car_door = 2  # For example.
# Monty can only choose door 3 now.
montys_choices = np.setdiff1d(doors, [my_door, car_door])
montys_choices
```

```
array([3])
```

```
my_door = 1  # For example.
car_door = 1  # For example.
# Monty can choose either door 2 or door 3.
montys_choices = np.setdiff1d(doors, [my_door, car_door])
montys_choices
```

```
array([2, 3])
```

If Monty can only choose one door, we'll take that. Otherwise we'll chose a door at random from the two doors available.

```
if len(montys_choices) == 1:  # Only one door available.
    montys_door = montys_choices[0]  # Take the first (of 1!).
else:  # Two doors to choose from:
    # Choose at random.
    montys_door = rnd.choice(montys_choices)
montys_door
```

```
np.int64(2)
```

In fact, we can avoid that `if len(` check for the number of doors, because `rnd.choice` will also work on a sequence of length 1 — in that case, it always returns the single element in the sequence, like this:

```
# rnd.choice on sequence with single element - always returns that element.
rnd.choice([2])
```

```
np.int64(2)
```

That means we can simplify the code above to:

```
# Choose single door left to choose, or door at random if two.
montys_door = rnd.choice(montys_choices)
montys_door
```

```
np.int64(3)
```

Now we know Monty's door, we can identify the other door, by removing our door, and Monty's door, from the available options:

```
remaining_doors = np.setdiff1d(doors, [my_door, montys_door])
# There is only one remaining door, take that.
other_door = remaining_doors[0]
other_door
```

```
np.int64(2)
```

The logic above gives us the full procedure for one trial.

```
my_door = rnd.choice(doors)
car_door = rnd.choice(doors)
# Which door will Monty open?
montys_choices  = np.setdiff1d(doors, [my_door, car_door])
# Choose single door left to choose, or door at random if two.
montys_door = rnd.choice(montys_choices)
# Now find the door we'll open if we switch.
remaining_doors = np.setdiff1d(doors, [my_door, montys_door])
# There is only one door left.
other_door = remaining_doors[0]
# Calculate the result of this trial.
if my_door == car_door:
    stay_wins = True
if other_door == car_door:
    switch_wins = True
```

All that remains is to put that trial procedure into a loop, and collect the results as we repeat the procedure many times.

```
# Arrays to store the results for each trial.
stay_wins = np.repeat([False], 10000)
switch_wins = np.repeat([False], 10000)

# A list of doors to chose from.
doors = [1, 2, 3]

for i in range(10000):
    # You will recognize the below as the single-trial procedure above.
    my_door = rnd.choice(doors)
    car_door = rnd.choice(doors)
    # Which door will Monty open?
    montys_choices  = np.setdiff1d(doors, [my_door, car_door])
    # Choose single door left to choose, or door at random if two.
    montys_door = rnd.choice(montys_choices)
    # Now find the door we'll open if we switch.
    remaining_doors = np.setdiff1d(doors, [my_door, montys_door])
    # There is only one door left.
    other_door = remaining_doors[0]
```

```
    # Calculate the result of this trial.
    if my_door == car_door:
        stay_wins[i] = True
    if other_door == car_door:
        switch_wins[i] = True

p_for_stay = np.sum(stay_wins) / 10000
p_for_switch = np.sum(switch_wins) / 10000

print('p for stay:', p_for_stay)
```

```
p for stay: 0.3326
```

```
print('p for switch:', p_for_switch)
```

```
p for switch: 0.6674
```

We can also follow the same strategy as we used for the second implementation of the two-ships problem (Section 10.5).

Here, as in the second two-ships implementation, we do not calculate the trial results (`stay_wins`, `switch_wins`) in each trial. Instead, we store the *doors* for each trial, and then use Boolean arrays to calculate the results for all trials, at the end.

```
# Instead of storing the trial results, we store the doors for each trial.
my_doors = np.zeros(10000)
car_doors = np.zeros(10000)
other_doors = np.zeros(10000)

doors = [1, 2, 3]

for i in range(10000):
    my_door = rnd.choice(doors)
    car_door = rnd.choice(doors)
    # Which door will Monty open?
    montys_choices  = np.setdiff1d(doors, [my_door, car_door])
    # Choose single door left to choose, or door at random if two.
    montys_door = rnd.choice(montys_choices)
    # Now find the door we'll open if we switch.
    remaining_doors = np.setdiff1d(doors, [my_door, montys_door])
    # There is only one door left.
```

```
    other_door = remaining_doors[0]

    # Store the doors we chose.
    my_doors[i] = my_door
    car_doors[i] = car_door
    other_doors[i] = other_door

# Now - at the end of all the trials, we use Boolean arrays to calculate the
# results.
stay_wins = my_doors == car_doors
switch_wins = other_doors == car_doors

p_for_stay = np.sum(stay_wins) / 10000
p_for_switch = np.sum(switch_wins) / 10000

print('p for stay:', p_for_stay)
```

```
p for stay: 0.3374
```

```
print('p for switch:', p_for_switch)
```

```
p for switch: 0.6626
```

10.8.1 Insight from the Monty Hall simulation

The code simulation gives us an estimate of the right answer, but it also forces us to set out the exact mechanics of the problem. For example, by looking at the code, we see that we can calculate "stay_wins" with this code alone:

```
# Just choose my door and the car door for each trial.
my_doors = np.zeros(10000)
car_doors = np.zeros(10000)
doors = [1, 2, 3]

for i in range(10000):
    my_doors[i] = rnd.choice(doors)
    car_doors[i] = rnd.choice(doors)

# Calculate whether I won by staying.
stay_wins = my_doors == car_doors
```

```
p_for_stay = np.sum(stay_wins) / 10000

print('p for stay:', p_for_stay)
```

```
p for stay: 0.3244
```

This calculation, on its own, tells us the answer, but it also points to another insight — whatever Monty does with the doors, it doesn't change the probability that our *initial guess* is right, and that must be 1 in 3 (0.333). If the probability of `stay_win` is 1 in 3, and we only have one other door to switch to, the probability of winning after switching must be 2 in 3 (0.666).

10.8.2 Simulation and a variant of Monty Hall

You have seen that you can avoid the silly mistakes that many of us make with probability — by asking the computer to tell you the result *before* you start to reason from first principles.

As an example, consider the following variant of the Monty Hall problem.

The set up to the problem has us choosing a door (`my_door` above), and then Monty opens one of the other two doors.

Sometimes (in fact, 2/3 of the time) there is a car behind one of Monty's doors. We've obliged Monty to open the *other* door, and his choice is forced.

When his choice was not forced, we had Monty choose the door at random.

For example, let us say we chose door 1.

Let us say that the car is also under door 1.

Monty has the option of choosing door 2 or door 3, and he chooses randomly between them.

```
my_door = 1  # We chose door 1 at random.
car_door = 1  # This trial, by chance, the car door is 1.
# Monty is left with doors 2 and 3 to choose from.
montys_choices  = np.setdiff1d(doors, [my_door, car_door])
# He chooses randomly.
montys_door = rnd.choice(montys_choices)
# Show the result
montys_door
```

```
np.int64(2)
```

Now — let us say we happen to know that Monty is rather lazy, and he will always choose the left-most (lower-numbered) door of the two options.

In the previous example, Monty had the option of choosing door 2 and 3. In this new scenario, we know that he will always choose door 2 (the left-most door).

```
my_door = 1  # We chose door 1 at random.
car_door = 1  # This trial, by chance, the car door is 1.
# Monty is left with doors 2 and 3 to choose from.
montys_choices  = np.setdiff1d(doors, [my_door, car_door])
# He chooses the left-most door, always.
montys_door = montys_choices[0]
# Show the result
montys_door
```

```
np.int64(2)
```

It feels as if we have more information about where the car is, when we know this. Consider the situation where we have chosen door 1, and Monty opens door 3. We know that he would have preferred to open door 2, if he was allowed. We therefore know he wasn't allowed to open door 2, and that means the car is definitely under door 2.

```
my_door = 1  # We chose door 1 at random.
car_door = 2  # This trial, by chance, the car door under door 2.
# Monty is left with door 3 only to choose from.
montys_choices  = np.setdiff1d(doors, [my_door, car_door])
# He chooses the left-most door, always.  But in this case, the left-most
# available door is 3 (he can't choose 2, it is the car_door).
# Notice the doors were in order, so the left-most door is the first door
# in the array.
montys_door = montys_choices[0]
# Show the result
montys_door
```

```
np.int64(3)
```

To take that into account, we might try a different strategy. We will stick to our own choice if Monty has chosen the left-most of the two doors he had available to him, because he might have chosen that door because there was a car underneath the other door, or because there was a car under neither, but he preferred the left door. But, if Monty chooses the right-most of the two-doors available to him, we will switch from our own choice to the other (unopened) door, because we can be sure that the car is under the other (unopened) door.

Call this the "switch if Monty chooses right door" strategy, or "switch if right" for short.

Can you see quickly whether this will be better than the "always stay" strategy? Will it be better than the "always switch" strategy? Take a moment to think it through, and write down your answers.

If you can quickly see the answer to both questions — well done — but, are you sure you are right?

We can test by simulation.

For our test of the "switch is right" strategy, we can tell if one door is to the right of another door by comparison; higher numbers mean further to the right: 2 is right of 1, and 3 is right of 2.

```
# Door 3 is right of door 1.
3 > 1
```

```
True
```

```
# A test of the switch-if-right strategy.
# The car doors.
car_doors = np.zeros(10000)
# The door we chose using the strategy.
strategy_doors = np.zeros(10000)

doors = [1, 2, 3]

for i in range(10000):
    my_door = rnd.choice(doors)
    car_door = rnd.choice(doors)
    # Which door will Monty open?
    montys_choices  = np.setdiff1d(doors, [my_door, car_door])
    # Choose Monty's door from the remaining options.
    # This time, he always prefers the left door.
    montys_door = montys_choices[0]
    # Now find the door we'll open if we switch.
    remaining_doors = np.setdiff1d(doors, [my_door, montys_door])
    # There is only one door remaining - but is Monty's door
    # to the right of this one?  Then Monty had to shift.
    other_door = remaining_doors[0]
    if montys_door > other_door:
        # Monty's door was the right-hand door, the car is under the other one.
        strategy_doors[i] = other_door
```

```
    else:  # We stick with the door we first thought of.
        strategy_doors[i] = my_door
    # Store the car door for this trial.
    car_doors[i] = car_door

strategy_wins = strategy_doors == car_doors

p_for_strategy = np.sum(strategy_wins) / 10000

print('p for strategy:', p_for_strategy)
```

```
p for strategy: 0.6641
```

We find that the "switch-if-right" has around the same chance of success as the "always-switch" strategy — of about 66.6%, or 2 in 3. Were your initial answers right? Now you've seen the result, can you see why it should be so? It may not be obvious — the Monty Hall problem is deceptively difficult. But our case here is that the simulation first gives you an estimate of the correct answer, and then, gives you a good basis for thinking more about the problem. That is:

- simulation is useful for estimation and
- simulation is useful for reflection.

> ℹ End of notebook: The Monty Hall problem
>
> `monty_hall` starts at Note 15.

10.9 Why use simulation?

Doing these simulations has two large benefits. First, it gives us the right answer, saving us from making a mistake. Second, the process of simulation forces us to think about how the problem works. This can give us better understanding, and make it easier to reason about the solution.

We will soon see that these same advantages also apply to reasoning about statistics.

11 Probability Theory, Part 2: Compound Probability

11.1 Introduction

In this chapter we will deal with what are usually called "probability problems" rather than the "statistical inference problems" discussed in later chapters. The difference is that for probability problems we begin with a knowledge of the properties of the universe with which we are working. (See Section 8.9 on the definition of resampling.)

We start with some basic problems in probability. To make sure we do know the properties of the universe we are working with, we start with poker, and a pack of cards. Working with some poker problems, we rediscover the fundamental distinction between sampling *with* and *without* replacement.

11.2 Introducing a poker problem: one pair (two of a kind)

What is the chance that the first five cards chosen from a deck of 52 (bridge/poker) cards will contain two (and only two) cards of the same denomination (two 3's for example)? (Please forgive the rather sterile unrealistic problems in this and the other chapters on probability. They reflect the literature in the field for 300 years. We'll get more realistic in the statistics chapters.)

We shall estimate the odds the way that gamblers have estimated gambling odds for thousands of years. First, check that the deck is a standard deck and is not missing any cards. (Overlooking such small but crucial matters often leads to errors in science.) Shuffle thoroughly until you are satisfied that the cards are randomly distributed. (It is surprisingly hard to shuffle well.) Then deal five cards, and mark down whether the hand does or does not contain a pair of the same denomination.

At this point, we must decide whether three of a kind, four of a kind or two pairs meet our criterion for a pair. Since our criterion is "two and only two," we decide *not* to count them.

Then replace the five cards in the deck, shuffle, and deal again. Again mark down whether the hand contains one pair of the same denomination. Do this many times. Then count the number of hands with one pair, and figure the proportion (as a percentage) of all hands.

Table 11.1 has the results of 25 hands of this procedure.

Table 11.1: Results of 25 hands for the problem "one pair"

Hand	Card 1	Card 2	Card 3	Card 4	Card 5	One pair?
1	King	King	Queen	10	6	Yes
2	8	Ace	4	10	3	No
3	4	5	Ace	Queen	10	No
4	3	Ace	5	3	Jack	Yes
5	6	King	6	3	3	No
6	Queen	7	Jack	5	8	No
7	9	4	9	Jack	5	Yes
8	3	3	3	5	5	Yes
9	Queen	4	Queen	6	4	No
10	Queen	3	7	7	8	Yes
11	8	9	7	8	Ace	Yes
12	Ace	9	4	2	Ace	Yes
13	4	3	Ace	9	5	No
14	10	7	8	King	4	No
15	Queen	8	Queen	8	5	No
16	King	10	Jack	10	10	No
17	Queen	Queen	Ace	King	7	Yes
18	5	6	Ace	4	6	Yes
19	3	5	2	King	9	No
20	8	Jack	7	10	3	No
21	5	4	Jack	2	King	No
22	5	4	Jack	King	2	No
23	King	King	6	2	5	Yes
24	8	9	6	Ace	5	No
25	Ace	7	4	9	9	Yes
% Yes						44%

In this series of 25 experiments, 44 percent of the hands contained one pair, and therefore 0.44 is our estimate (for the time being) of the probability that one pair will turn up in a poker hand. But we must notice that this estimate is based on only 25 hands, and therefore might well be fairly far off the mark (as we shall soon see).

This experimental "resampling" estimation does not require a deck of cards. For example, one might create a 52-sided die, one side for each card in the deck, and roll it five times to get a "hand." But note one important part of the procedure: No single "card" is allowed to come up twice in the same set of five spins, just as no single card can turn up twice or more in the same

hand. If the same "card" did turn up twice or more in a dice experiment, one could pretend that the roll had never taken place; this procedure is necessary to make the dice experiment analogous to the actual card-dealing situation under investigation. Otherwise, the results will be slightly in error. This type of sampling is "sampling without replacement," because each card is *not replaced* in the deck prior to dealing the next card (that is, prior to the end of the hand).

11.3 A first approach to the one-pair problem with code

We could also approach this problem using random numbers from the computer to simulate the values.

Let us first make some numbers from which to sample. We want to simulate a deck of playing cards analogous to the real cards we used previously. We don't need to simulate all the features of a deck, but only the features that matter for the problem at hand. In our case, the feature that matters is the face value. We require a deck with four "1"s, four "2"s, etc., up to four "13"s, where 1 is an Ace, and 13 is a King. The suits don't matter for our present purposes.

We first first make an array to represent the face values in one suit.

```
# Card values 1 through 13 (1 up to, not including 14).
one_suit = np.arange(1, 14)
one_suit
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

We have the face values for one suit, but we need the face values for whole deck of cards — four suits. We do this by making a new array that consists of four repeats of `one_suit`:

```
# Repeat the one_suit array four times
deck = np.repeat(one_suit, 4)
deck
```

```
array([ 1,  1,  1,  1,  2,  2,  2,  2,  3,  3,  3,  3,  4,  4,  4,  4,  5,
        5,  5,  5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  8,  9,  9,
        9,  9, 10, 10, 10, 10, 11, 11, 11, 11, 12, 12, 12, 12, 13, 13, 13,
       13])
```

11.4 Shuffling the deck with Python

At this point we have a complete deck in the variable `deck` . But that "deck" is ordered by value, first ones (Aces) then 2s and so on. If we do not shuffle the deck, the results will be predictable. Therefore, we would like to select five of these "cards" (52 values) at random. There are two ways of doing this. The first is to use the 'rnd.choice'](.python} tool in the familiar way, to choose 5 values at random from this strictly ordered deck. We want to draw these cards *without replacement* (of which more later). *Without replacement* means that once we have drawn a particular value, we cannot draw that value a second time — just as you cannot get the same card twice in a hand when the dealer deals you a hand of five cards.

So far, each of our uses of `rnd.choice` has done sampling *with replacement*, where you *can* get the same item more than once in a particular sample. Here we need *without replacement*. `rnd.choice` has an argument you can send, called `replace`, to tell it whether to replace values when drawing the sample. We have not used that argument so far, because the default is `True` — sampling *with replacement*. Here we need to use the argument — `replace=False` — to get sampling *without replacement*.

```
# One hand, sampling from the deck without replacement.
hand = rnd.choice(deck, size=5, replace=False)
hand
```

```
array([ 9,  4, 11,  9, 13])
```

The above is one way to get a random hand of five cards from the deck. Another way is to use the `rnd.permuted` function to shuffle the whole `deck` of 52 "cards" into a random order, just as a dealer would shuffle the deck before dealing. Then we could take — for example — the first five cards from the shuffled deck to give a random hand. See Section 8.14 for more on `rnd.permuted`.

```
# Shuffle the whole 52 card deck.
shuffled = rnd.permuted(deck)
# The "cards" are now in random order.
shuffled
```

```
array([12, 13,  2,  9,  6,  7,  7,  7, 11, 13,  2,  8,  6,  9,  4,  1,  5,
       12, 11,  9,  1,  2,  4,  2,  3,  3, 11,  6,  4, 11,  8,  7, 13,  8,
       12,  5,  4,  5,  9,  8,  5,  6,  3,  1,  1, 12,  3, 13, 10, 10, 10,
       10])
```

Now we can get our `hand` by taking the first five cards from the `deck`:

```
# Select the first five "cards" from the shuffled deck.
hand = shuffled[:5]
hand
```

```
array([12, 13,  2,  9,  6])
```

You have seen that we can use one of two procedures to a get random sample of five cards from `deck`, drawn without replacement:

1. Using `rnd.choice` with `size=5` and `replace=False` to take the random sample directly from `deck`, or
2. shuffling the entire `deck` and then taking the first five "cards" from the result of the shuffle.

Either is a valid way of getting five cards at random from the `deck`. It's up to us which to choose — we slightly prefer to shuffle and take the first five, because it is more like the physical procedure of shuffling the deck and dealing, but which you prefer, is up to you.

11.4.1 A first-pass computer solution to the one-pair problem

Choosing the shuffle deal way, the cell to generate one hand is:

```
shuffled = rnd.permuted(deck)
hand = shuffled[:5]
hand
```

```
array([ 7,  4, 12,  1,  2])
```

Without doing anything further, we could run this cell many times, and each time, we could note down whether the particular `hand` had exactly one pair or not.

Table 11.2 has the result of running that procedure 25 times:

Table 11.2: Results of 25 hands using random numbers

Hand	Card 1	Card 2	Card 3	Card 4	Card 5	One pair?
1	10	5	7	12	12	Yes
2	6	9	2	6	8	Yes
3	11	8	9	6	1	No
4	8	10	2	11	12	No

Hand	Card 1	Card 2	Card 3	Card 4	Card 5	One pair?
5	1	10	11	8	5	No
6	8	10	3	9	5	No
7	10	9	13	1	9	Yes
8	13	4	3	11	5	No
9	7	1	4	13	6	No
10	11	5	11	8	4	Yes
11	7	10	7	13	9	Yes
12	2	11	4	7	8	No
13	12	1	3	10	2	No
14	10	2	11	8	1	No
15	1	6	12	12	5	Yes
16	4	8	7	8	6	Yes
17	7	10	9	4	4	Yes
18	3	4	11	11	12	Yes
19	10	12	2	13	1	No
20	9	6	4	13	4	Yes
21	7	3	3	9	7	No
22	13	4	10	5	8	No
23	13	2	9	8	8	Yes
24	5	12	7	11	8	No
25	7	5	8	10	7	Yes
% Yes						48%

11.5 Finding exactly one pair using code

Thus far we have had to look ourselves at the set of cards, or at the numbers, and decide if there was exactly one pair. We would like the computer to do this for us. Let us stay with the numbers we generated above by dealing the random `hand` from the `deck` of numbers. To find pairs, we will go through the following procedure:

- For each possible value (1 through 13), count the number of times each value has occurred in `hand`. Call the result of this calculation — `repeat_nos`.
- Select `repeat_nos` values equal to 2;
- Count the number of "2" values in `repeat_nos`. This the number of pairs, and excludes three of a kind or four a kind.
- If the number of pairs is exactly one, label the `hand` as "Yes", otherwise label it as "No".

11.6 Finding number of repeats using `np.bincount`

Consider the following 5-card "hand" of values:

```
hand = np.array([5, 7, 5, 4, 7])
```

This hand represents a pair of 5s and a pair of 7s.

We want to detect the number of repeats for each possible card value, 1 through 13. Let's say we are looking for 5s. We can detect which of the values are equal to 5 by making a Boolean array, where there is `True` for a value equal to 5, and `False` otherwise:

```
is_5 = (hand == 5)
is_5
```

```
array([ True, False,  True, False, False])
```

We can then count the number of 5s with:

```
np.sum(is_5)
```

```
np.int64(2)
```

In one cell:

```
number_of_5s = np.sum(hand == 5)
number_of_5s
```

```
np.int64(2)
```

We could do this laborious task for every possible card value (1 through 13):

```
number_of_1s = np.sum(hand == 1)  # Number of aces in hand
number_of_2s = np.sum(hand == 2)  # Number of 2s in hand
number_of_3s = np.sum(hand == 3)
number_of_4s = np.sum(hand == 4)
number_of_5s = np.sum(hand == 5)
number_of_6s = np.sum(hand == 6)
number_of_7s = np.sum(hand == 7)
```

```
number_of_8s = np.sum(hand == 8)
number_of_9s = np.sum(hand == 9)
number_of_10s = np.sum(hand == 10)
number_of_11s = np.sum(hand == 11)
number_of_12s = np.sum(hand == 12)
number_of_13s = np.sum(hand == 13)  # Number of Kings in hand.
```

Above, we store the result for each card in a separate variable; this is inconvenient, because we would have to go through each variable checking for a pair (a value of 2). It would be more convenient to store these results in an array. One way to do that would be to store the result for card value 1 at position (index, offset) 1, the result for value 2 at position 2, and so on, like this:

```
# Make array length 14.  We don't use position (offset) 0, and the last
# position (offset) in this array will be 13.
repeat_nos = np.zeros(14)
repeat_nos[1] = np.sum(hand == 1)  # Number of aces in hand
repeat_nos[2] = np.sum(hand == 2)  # Number of 2s in hand
repeat_nos[3] = np.sum(hand == 3)
repeat_nos[4] = np.sum(hand == 4)
repeat_nos[5] = np.sum(hand == 5)
repeat_nos[6] = np.sum(hand == 6)
repeat_nos[7] = np.sum(hand == 7)
repeat_nos[8] = np.sum(hand == 8)
repeat_nos[9] = np.sum(hand == 9)
repeat_nos[10] = np.sum(hand == 10)
repeat_nos[11] = np.sum(hand == 11)
repeat_nos[12] = np.sum(hand == 12)
repeat_nos[13] = np.sum(hand == 13)  # Number of Kings in hand.
# Show the result
repeat_nos
```

```
array([0., 0., 0., 0., 1., 2., 0., 2., 0., 0., 0., 0., 0., 0.])
```

You may recognize all this repetitive typing as a good sign we could use a `for` loop to do the work — er — for us.

```
repeat_nos = np.zeros(14)
for i in range(14):  # Set i to be first 0, then 1, ... through 13.
    repeat_nos[i] = np.sum(hand == i)
# Show the result
repeat_nos
```

```
array([0., 0., 0., 0., 1., 2., 0., 2., 0., 0., 0., 0., 0., 0.])
```

Notice that we started our loop by checking for values equal to 0, and then values equal to 1 and so on. By our definition of the deck, no card can have value 0, so the first time through this loop, we will always get a count of 0. We could have saved ourselves a tiny amount of computing time if we had missed out that pointless step of checking 0, by using `for i in range(1, 14):` instead. In this case, we think the code is a little bit neater to read if we leave in the default start at 0, at a tiny cost in wasted computer effort.

In our particular `hand`, after we have done the count for 7s, we will always get 0 for card values 8, 9 … 13, because 7 was the highest card (maximum value) for our particular `hand`. As you might expect, there is a a Numpy function `np.max` that will quickly tell us the maximum value in the hand:

```
np.max(hand)
```

```
np.int64(7)
```

We can use `np.max` to make our loop more efficient, by stopping our checks when we've reached the maximum value, like this:

```
max_value = np.max(hand)
# Only make an array large enough to house counts for the max value.
repeat_nos = np.zeros(max_value + 1)
for i in range(max_value + 1):  # Set i to 0, then 1 ... through max_value
    repeat_nos[i] = np.sum(hand == i)
# Show the result
repeat_nos
```

```
array([0., 0., 0., 0., 1., 2., 0., 2.])
```

In fact, this is exactly what the function `np.bincount` does, so we can use that function instead of our loop, to do the same job:

```
repeat_nos = np.bincount(hand)
repeat_nos
```

```
array([0, 0, 0, 0, 1, 2, 0, 2])
```

11.7 Looking for hands with exactly one pair

Now we have `repeat_nos`, we can proceed with the rest of the steps above.

We can count the number of cards that have exactly two repeats:

```
(repeat_nos == 2)
```

```
array([False, False, False, False, False,  True, False,  True])
```

```
n_pairs = np.sum(repeat_nos == 2)
# Show the result
n_pairs
```

```
np.int64(2)
```

The hand is of interest to us only if the number of pairs is exactly 1:

```
# Check whether there is exactly one pair in this hand.
n_pairs == 1
```

```
np.False_
```

We now have the machinery to use Python for all the logic in simulating multiple hands, and checking for exactly one pair.

Let's do that, and use Python to do the full job of dealing many hands and finding pairs in each one. We repeat the procedure above using a `for` loop. The `for` loop commands the program to do ten thousand repeats of the statements in the "loop" (indented statements).

In the *body* of the loop (the part that gets repeated for each trial) we:

- Shuffle the `deck`.
- Deal ourselves a new `hand`.
- Calculate the `repeat_nos` *for this new hand*.
- Calculate the number of pairs from `repeat_nos`; store this as `n_pairs`.
- Put `n_pairs` for this repetition into the correct place in the scoring array `z`.

With that we end a single trial, and go back to the beginning, until we have done this 10000 times.

When those 10000 repetitions are over, the computer moves on to count (`sum`) the number of "1's" in the score-keeping array `z`, each "1" indicating a hand with exactly one pair. We store this count at location `k`. We divide `k` by 10000 to get the *proportion* of hands that had one pair, and we `print` the result of `k` to the screen.

> **Note 16: Notebook: One pair**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
# Create a bucket (vector) called a with four "1's," four "2's," four "3's,"
# etc., to represent a deck of cards
one_suit = np.arange(1, 14)
one_suit
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

```
# Repeat values for one suit four times to make a 52 card deck of values.
deck = np.repeat(one_suit, 4)
deck
```

```
array([ 1,  1,  1,  1,  2,  2,  2,  2,  3,  3,  3,  3,  4,  4,  4,  4,  5,
        5,  5,  5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  8,  9,  9,
        9,  9, 10, 10, 10, 10, 11, 11, 11, 11, 12, 12, 12, 12, 13, 13, 13,
       13])
```

```
# Array to store result of each trial.
z = np.zeros(10000)

# Repeat the following steps 10000 times
for i in range(10000):
    # Shuffle the deck
    shuffled = rnd.permuted(deck)

    # Take the first five cards to make a hand.
```

```
    hand = shuffled[:5]

    # How many pairs?
    # Counts for each card rank.
    repeat_nos = np.bincount(hand)
    n_pairs = np.sum(repeat_nos == 2)

    # Keep score of # of pairs
    z[i] = n_pairs

    # End loop, go back and repeat

# How often was there 1 pair?
k = np.sum(z == 1)

# Convert to proportion.
kk = k / 10000

# Show the result.
print(kk)
```

```
0.4191
```

> **i** End of notebook: One pair
>
> `one_pair` starts at Note 16.

In one run of the program, the result in `kk` was 0.419, so our estimate would be that the probability of a single pair is 0.419.

How accurate are these resampling estimates? The accuracy depends on the *number of hands* we deal — the more hands, the greater the accuracy. If we were to examine millions of hands, 42 percent would contain a pair each; that is, the chance of getting a pair in the long run is 42 percent. It turns out the estimate of 48 percent based on 25 hands in Table 11.1 is fairly close to the long-run estimate, though whether or not it is close *enough* depends on one's needs of course. If you need great accuracy, deal many more hands.

A note on the `decks`, `hands`, `repeat_noss` in the above program, etc.: These "variables" are called "array"s in Python. An *array* is an array (sequence) of elements that gets filled with numbers as Python conducts its operations.

To help keep things straight (though the program does not require it), we often use `z` to name the array that collects all the trial results, and `k` to denote our overall summary results. Or you could call it something like `scoreboard` — it's up to you.

How many trials (hands) should be made for the estimate? There is no easy answer.[1] One useful device is to run several (perhaps ten) equal sized sets of trials, and then examine whether the proportion of pairs found in the entire group of trials is very different from the proportions found in the various subgroup sets. If the proportions of pairs in the various subgroups differ greatly from one another or from the overall proportion, then keep running additional larger subgroups of trials until the variation from one subgroup to another is sufficiently small for your purposes. While such a procedure would be impractical using a deck of cards or any other physical means, it requires little effort with the computer and Python.

11.8 Two more tntroductory poker problems

Which is more likely, a poker hand with two pairs, or a hand with three of a kind? This is a *comparison* problem, rather than a problem in *absolute* estimation as was the previous example.

In a series of 100 "hands" that were "dealt" using random numbers, four hands contained two pairs, and two hands contained three of a kind. Is it safe to say, on the basis of these 100 hands, that hands with two pairs are more frequent than hands with three of a kind? To check, we deal another 300 hands. Among them we see fifteen hands with two pairs (3.75 percent) and eight hands with three of a kind (2 percent), for a total of nineteen to ten. Although the difference is not enormous, it is reasonably clear-cut. Another 400 hands might be advisable, but we shall not bother.

Earlier I obtained forty-four hands with *one* pair each out of 100 hands, which makes it quite plain that *one* pair is more frequent than *either* two pairs or three-of-a-kind. Obviously, we need *more* hands to compare the odds in favor of two pairs with the odds in favor of three-of-a-kind than to compare those for one pair with those for either two pairs or three-of-a-kind. Why? Because the difference in odds between one pair, and either two pairs or three-of-a-kind, is much greater than the difference in odds between two pairs and three-of-a-kind. This observation leads to a general rule: The closer the odds between two events, the *more trials* are needed to determine which has the higher odds.

Again it is interesting to compare the odds with the formulaic mathematical computations, which are 1 in 21 (4.75 percent) for a hand containing two pairs and 1 in 47 (2.1 percent) for

[1] One simple rule-of-thumb is to quadruple the original number. The reason for quadrupling is that four times as many iterations (trials) of this resampling procedure give *twice* as much accuracy (as measured by the standard deviation, the most frequent measurement of accuracy). That is, the error decreases with the square root of the number of iterations. If you see that you need *much* more accuracy, then *immediately* increase the number of iterations even more than four times — perhaps ten or a hundred times.

a hand containing three-of-a-kind — not too far from the estimates of .0375 and .02 derived from simulation.

To handle the problem with the aid of the computer, we simply need to estimate the proportion of hands having triplicates and the proportion of hands with two pairs, and compare those estimates.

To estimate the hands with three-of-a-kind, we can use a notebook just like "One Pair" earlier, except using `repeat_nos == 3` to search for triplicates instead of duplicates. The program, then, is:

> **Note 17: Notebook: Three of a kind**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
# Create a bucket (vector) called a with four "1's," four "2's," four "3's,"
# etc., to represent a deck of cards
one_suit = np.arange(1, 14)
# Repeat values for one suit four times to make a 52 card deck of values.
deck = np.repeat(one_suit, 4)
```

```
triples_per_trial = np.zeros(10000)

# Repeat the following steps 10000 times
for i in range(10000):
    # Shuffle the deck
    shuffled = rnd.permuted(deck)

    # Take the first five cards.
    hand = shuffled[:5]

    # How many triples?
    repeat_nos = np.bincount(hand)
    n_triples = np.sum(repeat_nos == 3)

    # Keep score of # of triples
    triples_per_trial[i] = n_triples

    # End loop, go back and repeat
```

```
# How often was there 1 pair?
n_triples = np.sum(triples_per_trial == 1)

# Convert to proportion
print(n_triples / 10000)
```

```
0.0272
```

> **i** End of notebook: Three of a kind
>
> `three_of_a_kind` starts at Note 17.

To estimate the probability of getting a two-pair hand, we revert to the original program (counting pairs), except that we examine all the results in the score-keeping vector `z` for hands in which we had *two* pairs, instead of *one*. ::: {.notebook name="two_pairs" title="Two pairs"}

```
import numpy as np
rnd = np.random.default_rng()

one_suit = np.arange(1, 14)
deck = np.repeat(one_suit, 4)
```

```
pairs_per_trial = np.zeros(10000)

# Repeat the following steps 10000 times
for i in range(10000):
    # Shuffle the deck
    shuffled = rnd.permuted(deck)

    # Take the first five cards.
    hand = shuffled[:5]

    # How many pairs?
    # Counts for each card rank.
    repeat_nos = np.bincount(hand)
    n_pairs = np.sum(repeat_nos == 2)

    # Keep score of # of pairs
    pairs_per_trial[i] = n_pairs
```

```
    # End loop, go back and repeat

# How often were there 2 pairs?
n_two_pairs = np.sum(pairs_per_trial == 2)

# Convert to proportion
print(n_two_pairs / 10000)
```

```
0.0487
```

:::

For efficiency (though efficiency really is not important here because the computer performs its operations so cheaply) we could develop both estimates in a single program by simply generating 10000 hands, and count the number with three-of-a-kind and the number with two pairs.

Before we leave the poker problems, we note a difficulty with Monte Carlo simulation. The probability of a royal flush is so low (about one in half a million) that it would take much computer time to compute. On the other hand, considerable inaccuracy is of little matter. Should one care whether the probability of a royal flush is 1/100,000 or 1/500,000?

11.9 The concepts of replacement and non-replacement

In the poker example above, we *did not replace* the first card we drew. If we were to replace the card, it would leave the probability the same before the second pick as before the first pick. That is, the conditional probability remains the same. *If we replace, conditions do not change.* But if we do not replace the item drawn, the probability changes from one moment to the next. (Perhaps refresh your mind with the examples in the discussion of conditional probability including Section 9.1.1)

If we sample with replacement, the sample drawings remain *independent* of each other — a topic addressed in Section 9.1.

In many cases, a key decision in modeling the situation in which we are interested is whether to sample with or without replacement. The choice must depend on the characteristics of the situation.

There is a close connection between the lack of finiteness of the concept of universe in a given situation, and sampling with replacement. That is, when the universe (population) we have in mind is not small, or has no conceptual bounds at all, then the probability of each successive

observation remains the same, and this is modeled by sampling with replacement. ("Not finite" is a less expansive term than "infinite," though one might regard them as synonymous.)

Chapter 12 discusses problems whose appropriate concept of a universe is finite, whereas Chapter 13 discusses problems whose appropriate concept of a universe is not finite. This general procedure will be discussed several times, with examples included.

12 Probability Theory, Part 3

This chapter discusses problems whose appropriate concept of a universe is not finite, whereas Chapter 13 discusses problems whose appropriate concept of a universe is finite.

How can a universe be infinite yet known? Consider, for example, the possible flips with a given coin; the number is not limited in any meaningful sense, yet we understand the properties of the coin and the probabilities of a head and a tail.

12.1 Example: The Birthday Problem

This examples illustrates the probability of duplication in a multi-outcome sample from an infinite universe.

As an indication of the power *and* simplicity of resampling methods, consider this famous examination question used in probability courses: What is the probability that two or more people among a roomful of (say) twenty-five people will have the same birthday? To obtain an answer we need simply examine the first twenty-five numbers from the random-number table that fall between "001" and "365" (the number of days in the year), record whether or not there is a duplication among the twenty-five, and repeat the process often enough to obtain a reasonably stable probability estimate.

Pose the question to a mathematical friend of yours, then watch her or him sweat for a while, and afterwards compare your answer to hers/his. I think you will find the correct answer very surprising. It is not unheard of for people who know how this problem works to take advantage of their knowledge by making and winning big bets on it. (See how a bit of knowledge of probability can immediately be profitable to you by avoiding such unfortunate occurrences?)

More specifically, these steps answer the question for the case of twenty-five people in the room:

- **Step 1.** Let three-digit random numbers 1-365 stand for the 365 days in the year. (Ignore leap year for simplicity.)
- **Step 2.** Examine for duplication among the first twenty-five random numbers chosen "001-365." (Triplicates or higher-order repeats are counted as duplicates here.) If there is one or more duplicate, record "yes." Otherwise record "no."

- **Step 3.** Repeat perhaps a thousand times, and calculate the proportion of a duplicate birthday among twenty-five people.

You would probably use the computer to generate the initial random numbers.

Now try the program written as follows.

> Note 18: Notebook: The Birthday Problem
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
n_with_same_birthday = np.zeros(10000)

days_of_year = np.arange(1, 366)  # 1 through 365

# Do 10000 trials (experiments)
for i in range(10000):
    # Generate 25 numbers randomly between "1" and "365" put them in a.
    a = rnd.choice(days_of_year, size=25)

    # Looking in a, count the number of multiples and put the result in
    # b. We request multiples > 1 because we are interested in any multiple,
    # whether it is a duplicate, triplicate, etc. Had we been interested only
    # in duplicates, we would have put in np.sum(counts == 2).
    counts = np.bincount(a)
    n_duplicates = np.sum(counts > 1)

    # Score the result of each trial to our store
    n_with_same_birthday[i] = n_duplicates

    # End the loop for the trial, go back and repeat the trial until all 10000
    # are complete, then proceed.

# Determine how many trials had at least one multiple.
k = np.sum(n_with_same_birthday)

# Convert to a proportion.
kk = k / 10000
```

```
# Print the result.
print(kk)
```

```
0.7799
```

> End of notebook: The Birthday Problem
>
> `birthday_problem` starts at Note 18.

We have dealt with this example in a rather intuitive and unsystematic fashion. From here on, we will work in a more systematic, step-by-step manner. And from here on the problems form an orderly sequence of the classical types of problems in probability theory (Chapter 12 and Chapter 13), and inferential statistics (Chapter 20 to Chapter 28.)

12.2 Example: Three Daughters Among Four Children

This problem illustrates a problem with two outcomes (Binomial[1]) and sampling with Replacement Among Equally Likely Outcomes.

What is the probability that exactly three of the four children in a four-child family will be daughters?[2]

The first step is to state that the approximate probability that a single birth will produce a daughter is 50-50 (1 in 2). This estimate is not strictly correct, because there are roughly 106 male children born to each 100 female children. But the approximation is close enough for most purposes, and the 50-50 split simplifies the job considerably. (Such "false" approximations are part of the everyday work of the scientist. The appropriate question is not whether or not a statement is "only" an approximation, but whether or not it is a *good enough* approximation for your purposes.)

The probability that a fair coin will turn up heads is .50 or 50-50, close to the probability of having a daughter. Therefore, flip a coin in groups of four flips, and count how often three of the flips produce *heads*. (You must decide in *advance* whether three heads means three girls or three boys.) It is as simple as that.

[1] Conventional labels such as "binomial" are used here for general background and as guideposts to orient the student of conventional statistics. You do not need to know these labels to understand the resampling approach; one of the advantages of resampling is that it avoids errors resulting from incorrect pigeonholing of problems.

[2] Here, by "daughter" or "girl", we just mean the assigned gender of the child at birth. If we wanted to be more sophisticated, and work out the proportion of children who identify as female, we would need some statistics on the current likelihood of changing gender identification by age, the children's ages, and so on.

In resampling estimation it is of the highest importance to work in a careful, step-by-step fashion — to write down the steps in the estimation, and then to do the experiments just as described in the steps. Here are a set of steps that will lead to a correct answer about the probability of getting three daughters among four children:

- **Step 1.** Using coins, let "heads" equal "girl" and "tails" equal "boy."
- **Step 2.** Throw four coins.
- **Step 3.** Examine whether the four coins fall with exactly three heads up. If so, write "yes" on a record sheet; otherwise write "no."
- **Step 4.** Repeat step 2 perhaps two hundred times.
- **Step 5.** Count the proportion "yes." This proportion is an estimate of the probability of obtaining exactly 3 daughters in 4 children.

The first few experimental trials might appear in the record sheet as follows (Table 12.1):

Table 12.1: Example trials from the three-girls problem

Number of Heads	Yes or No
1	No
0	No
3	Yes
2	No
1	No
2	No
...	...
...	...
...	...

The probability of getting three daughters in four births could also be found with a deck of cards, a random number table, a die, or with Python. For example, half the cards in a deck are black, so the probability of getting a black card ("daughter") from a full deck is 1 in 2. Therefore, deal a card, record "daughter" or "son," *replace* the card, shuffle, deal again, and so forth for 200 sets of four cards. Then count the proportion of groups of four cards in which you got four daughters.

> **Note 19: Notebook: Three Girls**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
girl_counts = np.zeros(10000)

# Do 10000 trials
for i in range(10000):

    # Select 'girl' or 'boy' at random, four times.
    children = rnd.choice(['girl', 'boy'], size=4)

    # Count the number of girls and put the result in b.
    b = np.sum(children == 'girl')

    # Keep track of each trial result in z.
    girl_counts[i] = b

    # End this trial, repeat the experiment until 10000 trials are complete,
    # then proceed.

# Count the number of experiments where we got exactly 3 girls, and put this
# result in k.
n_three_girls = np.sum(girl_counts == 3)

# Convert to a proportion.
three_girls_prop = n_three_girls / 10000

# Print the results.
print(three_girls_prop)
```

```
0.2502
```

> **i** End of notebook: Three Girls
>
> `three_girls` starts at Note 19.

Notice that the procedure outlined in the steps above would have been different (though almost identical) if we asked about the probability of *three or more* daughters rather than *exactly three* daughters among four children. For *three or more* daughters we would have scored "yes" on our score-keeping pad for *either* three or four heads, rather than for just three

heads. Likewise, in the computer solution we would have used the statement `n_three_girls = np.sum(girl_counts >= 3)` .

It is important that, in this case, in contrast to what we did in the example from Section 11.2 (the introductory poker example), the card is *replaced* each time so that each card is dealt from a full deck. This method is known as *sampling with replacement*. One samples with replacement whenever the successive events are *independent* ; in this case we assume that the chance of having a daughter remains the same (1 girl in 2 births) no matter what sex the previous births were[3]. But, if the first card dealt is black and would *not* be replaced, the chance of the second card being black would no longer be 26 in 52 (.50), but rather 25 in 51 (.49), if the first *three* cards are black and would not be replaced, the chances of the fourth card's being black would sink to 23 in 49 (.47).

To push the illustration further, consider what would happen if we used a deck of only six cards, half (3 of 6) black and half (3 of 6) red, instead of a deck of 52 cards. If the chosen card is replaced each time, the 6-card deck produces the same results as a 52-card deck; in fact, a two-card deck would do as well. But, if the sampling is done *without* replacement, it is *impossible* to obtain 4 "daughters" with the 6-card deck because there are only 3 "daughters" in the deck. To repeat, then, whenever you want to estimate the probability of some series of events where each event is independent of the other, you must sample *with replacement*. ## Variations of the daughters problem

In later chapters we will frequently refer to a problem which is identical in basic structure to the problem of three girls in four children — the probability of getting 9 females in ten calf births if the probability of a female birth is (say) .5 — when we set this problem in the context of the possibility that a genetic engineering practice is effective in increasing the proportion of females (desirable for the production of milk).

So far we have assumed the simple case where we have an array of values that we are sampling from, and we are selecting each of these values into the sample with equal probability.

For example, we started with the simple assumption that a child is just as likely to be born a boy as a girl. Our input is:

```
input_values = ['girl', 'boy']
```

By default, `rnd.choice` will draw the input values with equal probability. Here, we draw a sample (`children`) of four values from the input, where *each value* in `children` has an equal chance of being "girl" or "boy".

[3]This assumption is slightly contrary to scientific fact. A better example would be: What is the probability that four mothers delivering successively in a hospital will all have daughters? But that example has other difficulties — which is the way science always is.

```
children = rnd.choice(input_values, size=4)
children
```

```
array(['boy', 'boy', 'boy', 'girl'], dtype='<U4')
```

That is, `rnd.choice` gives each element in `input_values` an equal chance of being selected as the next element in `children`.

That is fine if we have some simple probability to simulate, like 0.5. But now let us imagine we want to get more precise. We happen to know that any given birth is just slightly more likely to be a boy than a girl.[4]. For example, the proportion of boys born in the UK is 0.513. Hence the proportion of girls is 1-0.513 = 0.487.

12.3 `rnd.choice` and the `p` argument

We could replicate this probability of 0.487 for 'girl' in the output sample by making an input array of 1000 strings, that contains 487 'girls' and 513 'boys':

```
big_girls = np.repeat(['girl', 'boy'], [487, 513])
```

Now if we sample using the default in `rnd.choice`, each *element* in the input `big_girls` array will have the same chance of appearing in the sample, but because there are 487 'girls', and 513 'boys', each with an equal chance of appearing in the sample, we will get a 'girl' in roughly 487 out of every 1000 elements we draw, and a boy roughly 513 / 1000 times. That is, our chance of any one element of being a 'girl' is, as we want, 0.487.

```
# Now each element has probability 0.487 of 'girl', 0.513 of 'boy'.
realistic_children = rnd.choice(big_girls, size=4)
realistic_children
```

```
array(['boy', 'boy', 'girl', 'boy'], dtype='<U4')
```

[4]This fact was the origin of the very first "significance test" in statistics. In the early 1700s, John Arbuthnot (1710) noticed that there were more christenings of boys than girls in London in every year for which he had figures. He showed that this was vanishingly unlikely if male births were in fact exactly as likely as female births, and attributed this to divine providence, because society tended to lose males faster than females in in adulthood, due to manual labor and war.

But, there is an easier way than compiling a big 1000 element array, and that is to use the `p=` argument to `rnd.choice`. This allows us to specify the probability with which we will draw each of the input elements into the output sample. For example, to draw 'girl' with probability 0.487 and 'boy' with probability 0.513, we would do:

```
# Draw 'girl' with probability (p) 0.487 and 'boy' 0.513.
children_again = rnd.choice(['girl', 'boy'], size=4, p=[0.487, 0.513])
children_again
```

```
array(['girl', 'boy', 'girl', 'girl'], dtype='<U4')
```

The `p` argument allows us to specify the probability of each element in the input array — so if we had three elements in the input array, we would need three probabilities in `p`. For example, let's say we were looking at some poorly-entered hospital records, we might have 'girl' or 'boy' recorded as the child's gender, but the record might be missing — 'not-recorded' — with a 19% chance:

```
# Draw 'girl' with probability (p) 0.4, 'boy' with p=0.41, 'not-recorded' with
# p=0.19.
rnd.choice(['girl', 'boy', 'not-recorded'], size=30, p=[0.4, 0.41, 0.19])
```

```
array(['girl', 'girl', 'girl', 'girl', 'boy', 'girl', 'girl',
       'not-recorded', 'girl', 'boy', 'boy', 'girl', 'girl', 'boy',
       'not-recorded', 'girl', 'not-recorded', 'boy', 'girl', 'boy',
       'not-recorded', 'girl', 'boy', 'girl', 'boy', 'not-recorded',
       'girl', 'girl', 'boy', 'not-recorded'], dtype='<U12')
```

How does the `p` argument to `rnd.choice` work?

You might wonder how Python does this trick of choosing the elements with different probabilities.
One way of doing this is to use *uniform* random numbers from 0 through 1. These are floating point numbers that can take any value, at random, from 0 through 1.

```
# Run this cell a few times to see random numbers anywhere from 0 through 1.
rnd.uniform()
```

```
0.3358873070551027
```

Because this random uniform number has an equal chance of being anywhere in the range 0 through 1, there is a 50% chance that any given number will be less then 0.5 and a

50% chance it is greater than 0.5. (Of course it could be *exactly equal to* 0.5, but this is vanishingly unlikely, so we will ignore that for now).
So, if we thought girls were exactly as likely as boys, we could select from 'girl' and 'boy' using this simple logic:

```
if rnd.uniform() < 0.5:
    result = 'girl'
else:
    result = 'boy'
```

But, by the same logic, there is a 0.487 chance that the random uniform number will be less than 0.487 and a 0.513 chance it will be greater. So, if we wanted to give ourselves a 0.487 chance of 'girl', we could do:

```
if rnd.uniform() < 0.487:
    result = 'girl'
else:
    result = 'boy'
```

We can extend the same kind of logic to three options. For example, there is a 0.4 chance the random uniform number will be less than 0.4, a 0.41 chance it will be somewhere between 0.4 and 0.81, and a 0.19 chance it will be greater than 0.81.

12.4 The daughters problem with more accurate probabilities

We can use the probability argument to `rnd.choice` to do a more realistic simulation of the chance of a family with exactly three girls. In this case it is easy to make the chance for the Python simulation, but much more difficult using physical devices like coins to simulate the randomness.

Remember, the original code for the 50-50 case, has the following:

```
# Select 'girl' or 'boy' at random, four times.
children = rnd.choice(['girl', 'boy'], size=4)

# Count the number of girls and put the result in b.
b = np.sum(children == 'girl')
```

The only change we need to the above, for the 0.487 - 0.513 case, is the one you see above:

```
# Give 'girl' 48.7% of the time, 'boy' 51.3% of the time.
children = rnd.choice(['girl', 'boy'], size=4, p=[0.487, 0.513])

b = np.sum(children == 'girl')
```

The rest of the program remains unchanged.

12.5 A note on clarifying and labeling problems

In conventional analytic texts and courses on inferential statistics, students are taught to distinguish between various classes of problems in order to decide which formula to apply. I doubt the wisdom of categorizing and labeling problems in that fashion, and the practice is unnecessary here. I consider it better that the student think through every new problem in the most fundamental terms. The exercise of this basic thinking avoids the mistakes that come from too-hasty and superficial pigeon-holing of problems into categories. Nevertheless, in order to help readers connect up the resampling material with the conventional curriculum of analytic methods, the examples presented here are given their conventional labels. And the examples given here cover the range of problems encountered in courses in probability and inferential statistics.

To repeat, one does not need to classify a problem when one proceeds with the Monte Carlo resampling method; you simply model the features of the situation you wish to analyze. In contrast, with conventional methods you must classify the situation and then apply procedures according to rules that depend upon the classification; often the decision about which rules to follow must be messy because classification is difficult in many cases, which contributes to the difficulty of choosing correct conventional formulaic methods.

12.6 Binomial trials

The problem of the three daughters in four births is known in the conventional literature as a "binomial sampling experiment with equally-likely outcomes." "Binomial" means that the *individual* simple event (a birth or a coin flip) can have only *two outcomes* (boy or girl, heads or tails), "binomial" meaning "two names" in Latin.[5]

[5]Conventional labels such as "binomial" are used here for general background and as guideposts to orient the student of conventional statistics. You do not need to know these labels to understand the resampling approach; one of the advantages of resampling is that it avoids errors resulting from incorrect pigeonholing of problems.

A fundamental property of binomial processes is that the individual trials are *independent*, a concept discussed earlier. A binomial sampling process is a *series* of binomial (one-of-two-outcome) events about which one may ask many sorts of questions — the probability of exactly X heads ("successes") in N trials, or the probability of X or more "successes" in N trials, and so on.

"Equally likely outcomes" means we assume that the probability of a girl or boy in any one birth is the same (though this assumption is slightly contrary to fact); we represent this assumption with the equal-probability heads and tails of a coin. Shortly we will come to binomial sampling experiments where the probabilities of the individual outcomes are *not* equal.

The term "with replacement" was explained earlier; if we were to use a deck of red and black cards (instead of a coin) for this resampling experiment, we would *replace* the card each time a card is drawn.

The introductory poker example from Section 11.2, illustrated sampling without replacement, as will other examples to follow.

This problem would be done conventionally with the binomial theorem using probabilities of .5, or of .487 and .513, asking about 3 successes in 4 trials.

12.7 Example: Three or More Successful Basketball Shots in Five Attempts

This is an example of two-outcome sampling with unequally-likely outcomes, with replacement — a binomial experiment.

What is the probability that a basketball player will score three or more baskets in five shots from a spot 30 feet from the basket, if on the average she succeeds with 25 percent of her shots from that spot?

In this problem the probabilities of "success" or "failure" are not equal, in contrast to the previous problem of the daughters. Instead of a 50-50 coin, then, an appropriate "model" would be a thumbtack that has a 25 percent chance of landing "up" when it falls, and a 75 percent chance of landing down.

If we lack a thumbtack known to have a 25 percent chance of landing "up," we could use a card deck and let spades equal "success" and the other three suits represent "failure." Our resampling experiment could then be done as follows:

1. Let "spade" stand for "successful shot," and the other suits stand for unsuccessful shot.
2. Draw a card, record its suit ("spade" or "other") and replace. Do so five times (for five shots).
3. Record whether the outcome of step 2 was three or more spades. If so indicate "yes," and otherwise "no."

4. Repeat steps 2-4 perhaps four hundred times.
5. Count the proportion "yes" out of the four hundred throws. That proportion estimates the probability of getting three or more baskets out of five shots if the probability of a single basket is .25.

The first four repetitions on your score sheet might look like this (Table 12.2):

Table 12.2: First four repetitions of 3 or more shots simulation

Card 1	Card 2	Card 3	Card 4	Card 5	Result
Spade	Other	Other	Other	Other	No
Other	Other	Other	Other	Other	No
Spade	Spade	Other	Spade	Spade	Yes
Other	Spade	Other	Other	Spade	No

Instead of cards, we could have used two-digit random numbers, with (say) "1-25" standing for "success," and "26-00" ("00" in place of "100") standing for failure. Then the steps would simply be:

1. Let the random numbers "1-25" stand for "successful shot," "26-00" for unsuccessful shot.
2. Draw five random numbers;
3. Count how many of the numbers are between "01" and "25." If three or more, score "yes."
4. Repeat step 2 four hundred times.

If you understand the earlier "three_girls" program, then the program below should be easy: To create 10000 samples, we start with a `for` statement. We then sample 5 numbers between "1" and "4" into our variable `a` to simulate the 5 shots, each with a 25 percent — or 1 in 4 — chance of scoring. We decide that 1 will stand for a successful shot, and 2 through 4 will stand for a missed shot, and therefore we count (`sum`) the number of 1's in `a` to determine the number of shots resulting in baskets in the current sample. The next step is to transfer the results of each trial to array `n_baskets`. We then finish the loop by unindenting the next line of code. The final step is to search the array `n_baskets`, after the 10000 samples have been generated and `sum` the times that 3 or more baskets were made. We place the results in `n_more_than_2`, calculate the proportion in `propo_more_than_2`, and then display the result.

> **Note 20: Notebook: Three or more basketball shots**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
n_baskets = np.zeros(10000)

# Do 10000 experimental trials.
for i in range(10000):

    # Generate 5 random numbers, each between 1 and 4, put them in "a".
    # Let "1" represent a basket, "2" through "4" be a miss.
    a = rnd.integers(1, 5, size=5)

    # Count the number of baskets, put that result in b.
    b = np.sum(a == 1)

    # Keep track of each experiment's results in z.
    n_baskets[i] = b

    # End the experiment, go back and repeat until all 10000 are completed, then
    # proceed.

# Determine how many experiments produced more than two baskets, put that
# result in k.
n_more_than_2 = np.sum(n_baskets > 2)

# Convert to a proportion.
prop_more_than_2 = n_more_than_2 / 10000

# Print the result.
print(prop_more_than_2)
```

```
0.104
```

ℹ End of notebook: Three or more basketball shots

`basketball_shots` starts at Note 20.

12.8 Note to the student of analytic probability theory

This problem would be done conventionally with the binomial theorem, asking about the chance of getting 3 successes in 5 trials, with the probability of a success = .25.

12.9 Example: One in Black, Two in White, No Misses in Three Archery Shots

This is an example of a multiple outcome (*multinomial*) sampling with unequally likely outcomes; with replacement.

Assume from past experience that a given archer puts 10 percent of his shots in the black ("bullseye") and 60 percent of his shots in the white ring around the bullseye, but misses with 30 percent of his shots. How likely is it that in three shots the shooter will get exactly one bullseye, two in the white, and no misses? Notice that unlike the previous cases, in this example there are more than two outcomes for each trial.

This problem may be handled with a deck of three colors (or suits) of cards in proportions varying according to the probabilities of the various outcomes, and sampling with replacement. Using random numbers is simpler, however:

- **Step 1.** Let "1" = "bullseye," "2-7" = "in the white," and "8-0" = "miss."
- **Step 2.** Choose three random numbers, and examine whether there are one "1" and two numbers "2-7." If so, record "yes," otherwise "no."
- **Step 3.** Repeat step 2 perhaps 400 times, and count the proportion of "yeses." This estimates the probability sought.

This problem would be handled in conventional probability theory with what is known as the *Multinomial Distribution*.

This problem may be quickly solved on the computer using Python with the notebook labeled "bullseye" below. Bullseye has a complication not found in previous problems: It tests whether two different sorts of events *both* happen — a bullseye plus two shots in the white.

After generating three randomly-drawn numbers between 1 and 10, we check with the `sum` function to see if there is a bullseye. If there is, the `if` statement tells the computer to continue with the operations, checking if there are two shots in the white; if there is no bullseye, the `if` statement tells the computer to end the trial and start another trial. A thousand repetitions are called for, the number of trials meeting the criteria are counted, and the results are then printed.

In addition to showing how this particular problem may be handled with Python, the "bullseye" program teaches you some more fundamentals of computer programming. The `if` statement and the two loops, one within the other, are basic tools of programming.

Note 21: Notebook: Bullseye

- Download notebook
- Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
# Make an array to store the results of each trial.
white_counts = np.zeros(10000)

# Do 10000 experimental trials
for i in range(10000):

    # To represent 3 shots, generate 3 numbers at random between "1" and "10"
    # and put them in a. We will let a "1" denote a bullseye, "2"-"7" a shot in
    # the white, and "8"-"10" a miss.
    a = rnd.integers(1, 11, size=3)

    # Count the number of bullseyes, put that result in b.
    b = np.sum(a == 1)

    # If there is exactly one bullseye, we will continue with counting the
    # other shots. (If there are no bullseyes, we need not bother - the
    # outcome we are interested in has not occurred.)
    if b == 1:

        # Count the number of shots in the white, put them in c. (Recall we are
        # doing this only if we got one bullseye.)
        c = np.sum((a >= 2) & (a <=7))

        # Keep track of the results of this second count.
        white_counts[i] = c

        # End the "if" sequence - we will do the following steps without regard
        # to the "if" condition.

    # End the above experiment and repeat it until 10000 repetitions are
    # complete, then continue.

# Count the number of occasions on which there are two in the white and a
# bullseye.
```

```
n_desired = np.sum(white_counts == 2)

# Convert to a proportion.
prop_desired = n_desired / 10000

# Print the results.
print(prop_desired)
```

```
0.1052
```

> **i** End of notebook: Bullseye
>
> `bullseye` starts at Note 21.

This example illustrates the addition rule that was introduced and discussed in Chapter 9. In Section 12.9, a bullseye, an in-the-white shot, and a missed shot are "mutually exclusive" events because a single shot cannot result in more than one of the three possible outcomes. One can calculate the probability of *either of two* mutually-exclusive outcomes by adding their probabilities. The probability of *either* a bullseye or a shot in the white is .1 + .6 = .7. The probability of an arrow *either* in the white *or* a miss is .6 + .3 = .9. The logic of the addition rule is obvious when we examine the random numbers given to the outcomes. Seven of 10 random numbers belong to "bullseye" or "in the white," and nine of 10 belong to "in the white" or "miss."

12.10 Example: Two Groups of Heart Patients

We want to learn how likely it is that, by chance, group A would have as little as two deaths more than group B — Table 12.3:

Table 12.3: Two Groups of Heart Patients

	Live	Die
Group A	79	11
Group B	21	9

This problem, phrased here as a question in probability, is the prototype of a problem in statistics that we will consider later (which the conventional theory would handle with a "chi square distribution"). We can handle it in either of two ways, as follows:

Approach A

1. Put 120 balls into a bucket, 100 white (for live) and 20 black (for die).
2. Draw 30 balls randomly and assign them to Group B; the others are assigned to group A.
3. Count the numbers of black balls in the two groups and determine whether Group A's excess "deaths" (= black balls), compared to Group B, is two or fewer (or what is equivalent in this case, whether there are 11 or fewer black balls in Group A); if so, write "Yes," otherwise "No."
4. Repeat steps 2 and 3 perhaps 10000 times and compute the proportion "Yes."

A second way we shall think about this sort of problem may be handled as follows:

Approach B

1. Put 120 balls into a bucket, 100 white (for live) and 20 black (for die) (as before).
2. Draw balls one by one, replacing the drawn ball each time, until you have accumulated 90 balls for Group A and 30 balls for Group B. (You could, of course, just as well use a bucket for 4 white and 1 black balls or 8 white and 2 black in this approach.)
3. As in approach "A" above, count the numbers of black balls in the two groups and determine whether Group A's excess deaths is two or fewer; if so, write "Yes," otherwise "No."
4. As above, repeat steps 2 and 3 perhaps 10000 times and compute the proportion "Yes."

We must also take into account the possibility of a similar eye-catching "unbalanced" result of a much larger proportion of deaths in Group B. It will be a tough decision how to do so, but a reasonable option is to simply double the probability computed in step 4a or 4b.

Deciding which of these two approaches — the "permutation" (without replacement) and "bootstrap" (with replacement) methods — is the more appropriate is often a thorny matter; it will be discussed latter in Chapter 24. In many cases, however, the two approaches will lead to similar results.

Later, we will actually carry out these procedures with the aid of Python, and estimate the probabilities we seek.

12.11 Example: Dispersion of a Sum of Random Variables — Hammer Lengths — Heads and Handles

The distribution of lengths for hammer *handles* is as follows: 20 percent are 10 inches long, 30 percent are 10.1 inches, 30 percent are 10.2 inches, and 20 percent are 10.3 inches long. The distribution of lengths for hammer *heads* is as follows: 2.0 inches, 20 percent; 2.1 inches, 20 percent; 2.2 inches, 30 percent; 2.3 inches, 20 percent; 2.4 inches, 10 percent.

If you draw a handle and a head at random, what will be the mean total length? In Chapter 9 we saw that the conventional formulaic method tells you that an answer with a formula that

says the sum of the means is the mean of the sums, but it is easy to get the answer with simulation. But now we ask about the *dispersion* of the sum. There are formulaic rules for such measures as the variance. But consider this other example: What proportion of the hammers made with handles and heads drawn at random will have lengths equal to or greater than 12.4 inches? No simple formula will provide an answer. And if the number of categories is increased considerably, any formulaic approach will be become burdensome if not undoable. But Monte Carlo simulation produces an answer quickly and easily, as follows:

1. Fill a bucket with:

   - 2 balls marked "10" (inches),
   - 3 balls marked "10.1",
   - 3 marked "10.2", and
   - 2 marked "10.3".

   This bucket represents the handles.

   Fill another bucket with:

   - 2 balls marked "2.0",
   - 2 balls marked "2.1",
   - 3 balls marked "2.2",
   - 2 balls marked "2.3" and
   - 1 ball marked "2.4".

   This bucket represents the heads.

2. Pick a ball from each of the "handles" and "heads" bucket, calculate the sum, and replace the balls.

3. Repeat perhaps 200 times (more when you write a computer program), and calculate the proportion of the sums that are greater than 12.4 inches.

You may also want to forego learning the standard "rule," and simply estimate the mean this way, also. As an exercise, compute the interquartile range — the difference between the 25th and the 75th percentiles.

12.12 Example: The Product of Random Variables — Theft by Employees

The distribution of the number of thefts per month you can expect in your business is as follows:

Number	Probability
0	0.5
1	0.2
2	0.1
3	0.1
4	0.1

The amounts that may be stolen on any theft are as follows:

Amount	Probability
$50	0.4
$75	0.4
$100	0.1
$125	0.1

The same procedure as used above to estimate the mean length of hammers — add the lengths of handles and heads — can be used for this problem except that the results of the drawings from each bucket are multiplied rather than added.

In this case there is again a simple rule: The mean of the products equals the product of the means. But this rule holds only when the two urns are indeed independent of each other, as they are in this case.

The next two problems are a bit harder than the previous ones; you might skip them for now and come back to them a bit later. However, with the Monte Carlo simulation method they are within the grasp of any introductory student who has had just a bit of experience with the method. In contrast, a standard book whose lead author is Frederick Mosteller, as respected a statistician as there is, says of this type of problem: "Naturally, in this book we cannot expect to study such difficult problems in their full generality [that is, show how to solve them, rather than merely state them], but we can lay a foundation for their study." (Mosteller, Rourke, and Thomas 1961, 5)

12.13 Example: Flipping Pennies to the End

Two players, each with a stake of ten pennies, engage in the following game: A coin is tossed, and if it is (say) heads, player A gives player B a penny; if it is tails, player B gives player A a penny. What is the probability that one player will lose his or her entire stake of 10 pennies if they play for 200 tosses?

This is a classic problem in probability theory; it has many everyday applications in situations such as inventory management. For example, what is the probability of going out of stock of a given item in a given week if customers and deliveries arrive randomly? It also is a model for many processes in modern particle physics.

Solution of the penny-matching problem with coins is straightforward. Repeatedly flip a coin and check if one player or the other reaches a zero balance before you reach 200 flips. Or with random numbers:

1. Numbers "1-5" = head = "+1"; Numbers "6-0" = tail = "-1."
2. Proceed down a series of 200 numbers, keeping a running tally of the "+1"'s and the "-1"'s. If the tally reaches "+10" or "-10" on or before the two-hundredth digit, record "yes"; otherwise record "no."
3. Repeat step 2 perhaps 400 or 10000 times, and calculate the proportion of "yeses." This estimates the probability sought.

The following Python program also solves the problem. The heart of the program starts at the line where the program models a coin flip with the statement: `c = rnd.integers(1, 3)` After you study that, go back and notice the inner `for` loop starting with `for j in range(200):` that describes the procedure for flipping a coin 200 times. Finally, note how the outer `for i in range(10000):` loop simulates 10000 games, each game consisting of the 200 coin flips we generated with the inner `for` loop above.

> **Note 22: Notebook: Simulating the pennies game**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()
```

```
someone_won = np.zeros(10000)

# Do 10000 trials
for i in range(10000):

    # Record the number 10: a's stake
    a_stake = 10

    # Same for b
    b_stake = 10

    # An indicator flag that will be set to "1" when somebody wins.
```

```
    flag = 0

    # Repeat the following steps 200 times.
    # Notice we use "j" as the counter variable, to avoid overwriting
    # "i", the counter variable for the 10000 trials.
    for j in range(200):
        # Generate the equivalent of a coin flip, letting 1 = heads,
        # 2 = tails
        c = rnd.integers(1, 3)

        # If it's a heads
        if c == 1:

            # Add 1 to b's stake
            b_stake = b_stake + 1

            # Subtract 1 from a's stake
            a_stake = a_stake - 1

        # End the "if" condition

        # If it's a tails
        if c == 2:

            # Add one to a's stake
            a_stake = a_stake + 1

            # Subtract 1 from b's stake
            b_stake = b_stake - 1

        # End the "if" condition

        # If a has won
        if a_stake == 20:

            # Set the indicator flag to 1
            flag = 1

        # If b has won
        if b_stake == 20:

            # Set the indicator flag to 1
```

```
            flag = 1

    # End the repeat loop for 200 plays (note that the indicator flag stays at
    # 0 if neither a nor b has won)

    # Keep track of whether anybody won
    someone_won[i] = flag

# End the 10000 trials

# Find out how often somebody won
n_wins = np.sum(someone_won)

# Convert to a proportion
prop_wins = n_wins / 10000

# Print the results
print(prop_wins)
```

```
0.8918
```

> End of notebook: Simulating the pennies game
>
> `pennies` starts at Note 22.

A similar example: Your warehouse starts out with a supply of twelve capacirators. Every three days a new shipment of two capacirators is received. There is a .6 probability that a capacirator will be used each morning, and the same each afternoon. (It is as if a random drawing is made each half-day to see if a capacirator is used; two capacirators may be used in a single day, or one or none). How long will be it, on the average, before the warehouse runs out of stock?

12.14 Example: A Drunk's Random Walk

If a drunk chooses the direction of each step randomly, will he ever get home? If he can only walk on the road on which he lives, the problem is almost the same as the gambler's-ruin problem above ("pennies"). But if the drunk can go north-south as well as east-west, the problem becomes a bit different and interesting.

Looking now at Figure 12.1 — what is the probability of the drunk reaching *either* his house (at 3 steps east, 2 steps north) *or* my house (1 west, 4 south) before he finishes taking twelve steps?

One way to handle the problem would be to use a four-directional spinner such as is used with a child's board game, and then keep track of each step on a piece of graph paper. The reader may construct a Python program as an exercise.

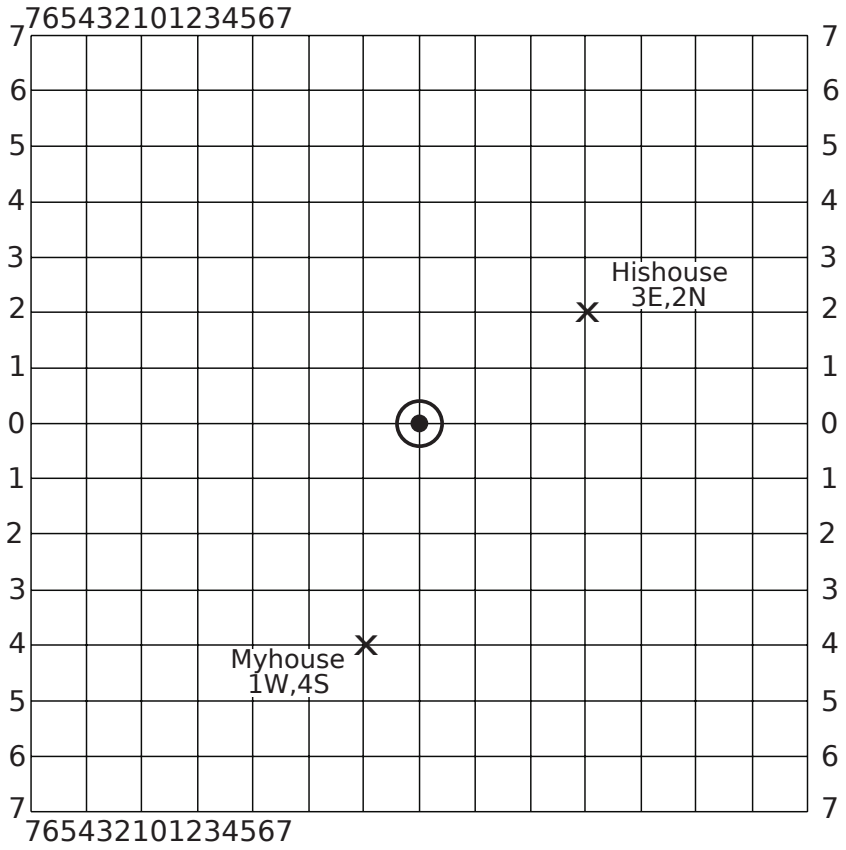


Figure 12.1: Drunk random walk

12.15 Example: public and private liquor pricing

Let's end this chapter with an actual example that will be used again in Chapter 13 when discussing probability in finite universes, and then at great length in the context of statistics in Chapter 24. This example also illustrates the close connection between problems in pure probability and those in statistical inference.

As of 1963, there were 26 U.S. states in whose liquor systems the retail liquor stores are

privately owned, and 16 "monopoly" states where the state government owns the retail liquor stores. (Some states were omitted for technical reasons.) These were the representative 1961 prices of a fifth of Seagram 7 Crown whiskey in the two sets of states (Table 12.6):

Table 12.6: Whiskey prices by state category

	Private	Government
	4.82	4.65
	5.29	4.55
	4.89	4.11
	4.95	4.15
	4.55	4.2
	4.9	4.55
	5.25	3.8
	5.3	4.0
	4.29	4.19
	4.85	4.75
	4.54	4.74
	4.75	4.5
	4.85	4.1
	4.85	4.0
	4.5	5.05
	4.75	4.2
	4.79	
	4.85	
	4.79	
	4.95	
	4.95	
	4.75	
	5.2	
	5.1	
	4.8	
	4.29	
Count	26	16
Mean	4.84	4.35

Let us consider that all these states' prices constitute one single universe (an assumption whose justification will be discussed later). If so, one can ask: If these 42 states constitute a single universe, how likely is it that one would choose two samples at random, containing 16 and 26 observations, that would have prices as different as $.49 (the difference between the means that was actually observed)?

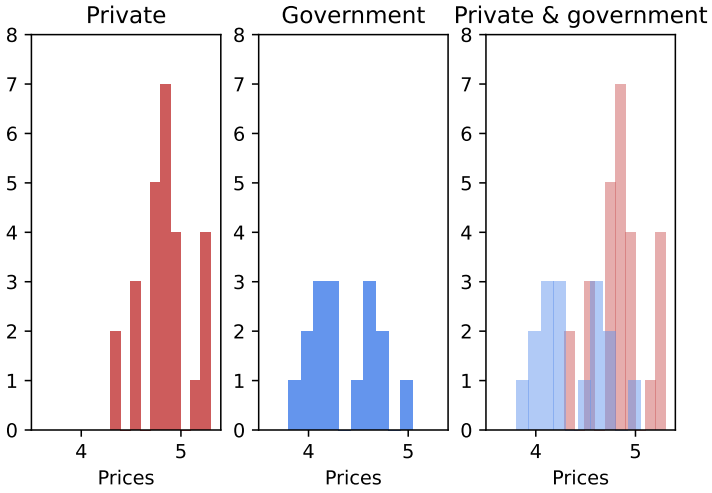


Figure 12.2: Whiskey prices by state category

This can be thought of as problem in pure probability because we begin with a known universe and ask how it would behave with random drawings from it. We sample *with replacement* ; the decision to do so, rather than to sample without replacement (which is the way I had first done it, and for which there may be better justification) will be discussed later. We do so to introduce a "bootstrap"-type procedure (defined later) as follows: Write each of the forty-two observed state prices on a separate card. The shuffled deck simulated a situation in which each state has an equal chance for each price. Repeatedly deal groups of 16 and 26 cards, replacing the cards as they are chosen, to simulate hypothetical monopoly-state and private-state samples. For each trial, calculate the difference in mean prices.

These are the steps systematically:

- **Step A:** Write each of the 42 prices on a card and shuffle.
- **Steps B and C** (combined in this case): i) Draw cards randomly with replacement into groups of 16 and 26 cards. Then ii) calculate the mean price difference between the groups, and iii) compare the simulation-trial difference to the observed mean difference of $4.84 - $4.35 = $.49; if it is as great or greater than $.49, write "yes," otherwise "no."
- **Step D:** Repeat step B-C a hundred or a thousand times. Calculate the proportion "yes," which estimates the probability we seek.

The probability that the postulated universe would produce a difference between groups as large or larger than observed in 1961 is estimated by how frequently the mean of the group of randomly-chosen sixteen prices from the simulated state-ownership universe is less than (or equal to) the mean of the actual sixteen state-ownership prices. The following notebook performs the operations described above.

12.15.1 Concatenating arrays

Before we start the simulation, we need a little extra NumPy machinery.

In what follows, we are going to make an array for the 26 *private* prices, and another array for the 16 *government* prices, and then concatenate these two arrays to make a new array with 26 + 16 = 42 elements, where the first 26 elements are the private prices and the last 16 elements are the government prices.

You will see that in action below, but this is what that *concatenation* looks like in NumPy for some example arrays.

We use Numpy's `concatenate` function to concatenate two arrays:

```
arr_a = np.array([1, 2, 3])
arr_b = np.array([10, 11, 12, 13])
# Use np.concatenate function to make new array consisting of elements in
# first array followed by elements in second array.
both = np.concatenate([arr_a, arr_b])
# Show the result.
both
```

```
array([ 1,  2,  3, 10, 11, 12, 13])
```

12.15.2 Plotting histograms

The other procedure we will use for the simulation, is graphing the results with a *histogram*.

> Note 23: Notebook: Plotting histograms
>
> - Download notebook
> - Interact

A histogram is a visual way to show the *distribution* of a sequence of values.

We now enter the world of *plotting* in Python. As Numpy is a Python library for working with arrays, Matplotlib is a library for making and showing plots.

To use the Numpy library, we `import` it. As you have seen, the usual convention is to make the standard `numpy` library name easier to read and type, by renaming the library to `np` on `import`, like this:

```
# Import numpy library and rename to "np"
import numpy as np
```

In a similar way, we need to import the Matplotlib library. In fact we will be using a particular part of the Matplotlib library, called `pyplot`.

We use the following standard convention to import the `pyplot` part of the Matplotlib library and give it the shorter name of `plt`:

```
import matplotlib.pyplot as plt
```

Note 24: Modules and submodules

We have been calling Numpy and Matlotlib *libraries*, but technically, Python calls these *modules*. Modules are collections of code and data that you can `import` into Python. For example, Numpy (now renamed as `np`) is a module:

```
# Show type for the import Numpy module (renamed as "np").
type(np)
```

```
<class 'module'>
```

We can get elements contained in (attached to) a module using the `.` syntax. For example, here we get the value of the `pi` variable, attached to the Numpy module.

```
# Get and show the value of the variable "pi" attached to (contained within)
# the Numpy module.
np.pi
```

```
3.141592653589793
```

One type of thing a module can contain, is other modules. These modules-attached-to-modules are called *submodules*. Perhaps without knowing, you have already used the `random` submodule attached to the Numpy module:

```
# "random" is itself a module, attached to (contained within) the Numpy
# module.  It is therefore a "submodule" of Numpy.
type(np.random)
```

```
<class 'module'>
```

We used the `default_rng` function from the `random` submodule to create random number generators:

```
rng = np.random.default_rng()
```

`pyplot` is a submodule of Matplotlib.

```
# Reimport the module to remind ourselves of the import line.
import matplotlib.pyplot as plt
# plt is a new name we have set for the "pyplot" submodule of Matplotlib.
type(plt)
```

```
<class 'module'>
```

The `pyplot` submodule of Matplotlib has many useful functions for making and displaying plots.

The easiest way to explain histograms is to show one.

Let's start with a sequence of values we are interested in:

Here are the 24 values for whiskey prices in states that did not have a liquor monopoly (`priv`).

```
priv = np.array([
    4.82, 5.29, 4.89, 4.95, 4.55, 4.90, 5.25, 5.30, 4.29, 4.85, 4.54, 4.75,
    4.85, 4.85, 4.50, 4.75, 4.79, 4.85, 4.79, 4.95, 4.95, 4.75, 5.20, 5.10,
    4.80, 4.29])
```

These are the 16 values for states with a liquor monopoly (`govt`):

```
govt = np.array([
    4.65, 4.55, 4.11, 4.15, 4.20, 4.55, 3.80, 4.00, 4.19, 4.75, 4.74, 4.50,
    4.10, 4.00, 5.05, 4.20])
```

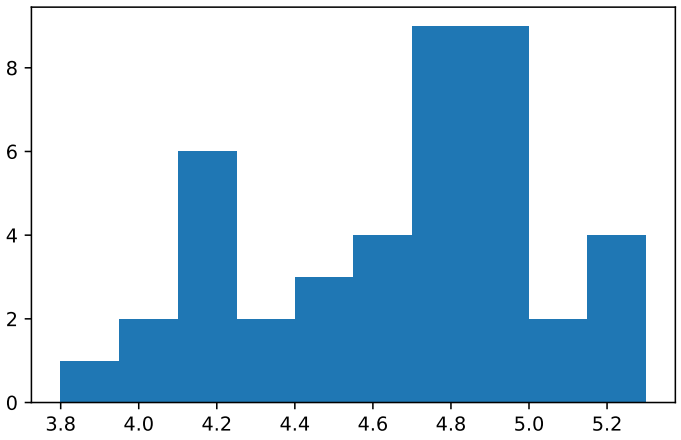
We concatenate these values to get a sequence (an array) of all 40 liquor prices:

```
prices = np.concatenate([priv, govt])
prices
```

```
array([4.82, 5.29, 4.89, 4.95, 4.55, 4.9 , 5.25, 5.3 , 4.29, 4.85, 4.54,
       4.75, 4.85, 4.85, 4.5 , 4.75, 4.79, 4.85, 4.79, 4.95, 4.95, 4.75,
       5.2 , 5.1 , 4.8 , 4.29, 4.65, 4.55, 4.11, 4.15, 4.2 , 4.55, 3.8 ,
       4.  , 4.19, 4.75, 4.74, 4.5 , 4.1 , 4.  , 5.05, 4.2 ])
```

We are interested in the distribution of these 40 values. To show the distribution, we can make and show a histogram of these prices, using the `hist` function attached to the `plt` submodule .

```
hist_res = plt.hist(prices)
```



`plt.hist` has calculated an array of suitable intervals (*bins*) to divide up the range of values, and then counted how many values in `prices` fall into each interval (bin).

You will notice that `plt.hist` has sent back some results from the process of making the histogram. In fact, the results are in the form of a list.

The first result of interest to us is the definition of the intervals (bins) into which the histogram has divided the range of `prices` values.

In fact, `plt.hist` sent back the edges of these bins in the second element of `hist_res`:

```
# The second element in the results list is the array of bin edges.
bin_edges = hist_res[1]
bin_edges
```

```
array([3.8 , 3.95, 4.1 , 4.25, 4.4 , 4.55, 4.7 , 4.85, 5.  , 5.15, 5.3 ])
```

Think of this array as the 10 values that start each of the 10 bins, followed by a final value that ends the final bin.

This means that the first bin was from (including) 3.8 up to, but not including 3.95, the second bin was from (including) 3.95 up to, but not including 4.1 and so on. The last bin is from (including) 5.15 through (including) 5.3. Notice there are 11 edges, forming 10 bins.

Put another way, the edges that `plt.hist` sent back are the 10 left hand (inclusive) edges of the 10 bins, and a final right hand (inclusive) edge of the final (10^{th}) bin.

The first element that comes back in the list of results is the array of counts of the values in `prices` that fall within each bin.

```
# The first element in the results list is the counts of values falling into
# each bin.
counts = hist_res[0]
counts
```

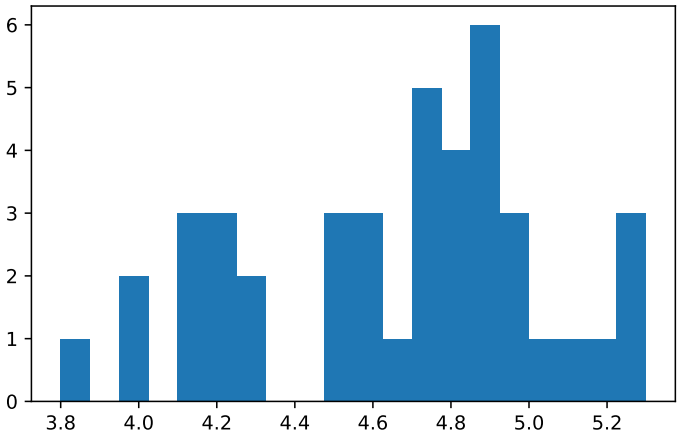
```
array([1., 2., 6., 2., 3., 4., 9., 9., 2., 4.])
```

The values tell us that 1 value from `prices` fell in the range 3.8 up to (not including) 3.95 (were within the first bin), 2 values fell in the range 3.95 up to (not including) 4.1, and so on.

That the counts correspond to the heights of the bars on the histogram, so the first bar has height 1, the second bar has height 2, and so on.

By default, `plt.hist` assumes you want 10 bins, and uses its default method of calculation to work out the edges for those 10 bins. You can specify another number of bins, by sending a number to the `bins` argument of `plt.hist`. For example, you might want 20 bins:

```
results_20 = plt.hist(prices, bins=20)
```



We now have 21 new edge values, the first 20 values giving the (inclusive) left-hand edges, and the last giving the (inclusive) right hand edge of the last bin.

```
bin_edges_20 = results_20[1]
bin_edges_20
```

```
array([3.8  , 3.875, 3.95 , 4.025, 4.1  , 4.175, 4.25 , 4.325, 4.4  ,
       4.475, 4.55 , 4.625, 4.7  , 4.775, 4.85 , 4.925, 5.   , 5.075,
       5.15 , 5.225, 5.3  ])
```

We can also specify our own edges, in order to bypass `plt.hist`s default algorithm to calculate edges. For example, we might prefer 16 bins of width 0.1, starting at 3.8, giving edges like this:

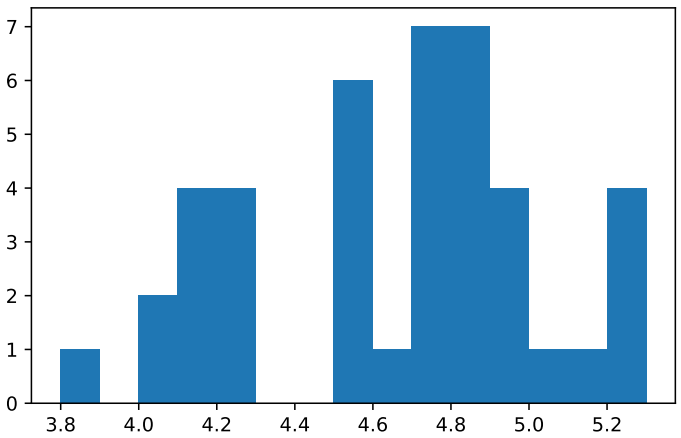
```
our_edges = 3.8 + np.arange(16) * 0.1
our_edges
```

```
array([3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. ,
       5.1, 5.2, 5.3])
```

We can send these directly to `plt.hist` to set the edges:

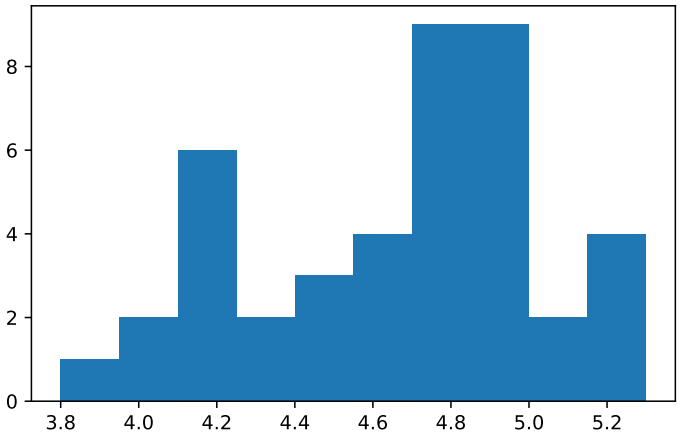
```
results_16 = plt.hist(prices, bins=our_edges)
# Show the edges that come back (these are the edges we sent).
results_16[1]
```

```
array([3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. ,
       5.1, 5.2, 5.3])
```



If you are running the notebook interactively in Jupyter, running `plt.hist` on its own, as below, will show the values as the result of the cell, along with the plot. (You won't see these results displayed in the textbook, because we use different software to show outputs when we build the textbook).

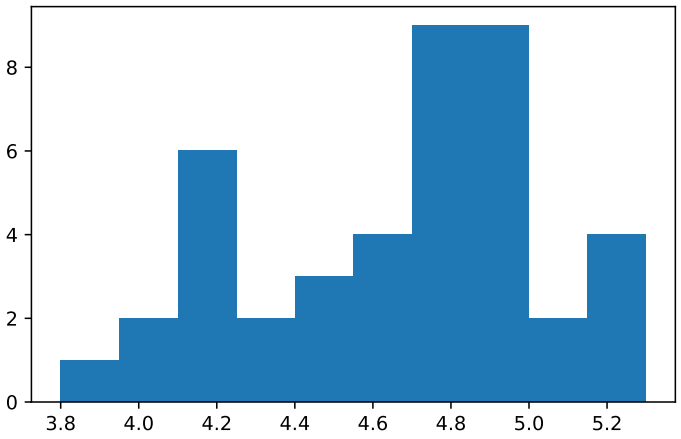
```
# If we don't collect the results, Jupyter shows them to us,
# if this is the last expression in the cell.
# (You won't see the results displayed in the textbook).
plt.hist(prices)
```



Interactive Jupyter will display the returned list of results, because we have not collected the results by assigning them to a variable. More technically, on its own, the `plt.hist` line is an *expression* (code that results in a value), and Jupyter will, by default, display the results of an expression that ends the code in a cell.

It can be distracting to see a display of the results list from a plotting cell, so from now on we will suppress Jupyter's default behavior of displaying the results list from `plt.hist`, by adding a semi-colon at the end of the code line, as in the cell below. (Remember, in the textbook, but not in Jupyter, this will give the same result as `plt.hist(prices)` above, because of the display system we use for the textbook.)

```
plt.hist(prices);  # Note the semi-colon
```



The semi-colon is a standard indicator to Jupyter that it should not display the results that came back from the function call. We will use it to suppress the display of various values that come back from these functions, as they are usually not of immediate interest.

> End of notebook: Plotting histograms
>
> `on_histograms` starts at Note 23.

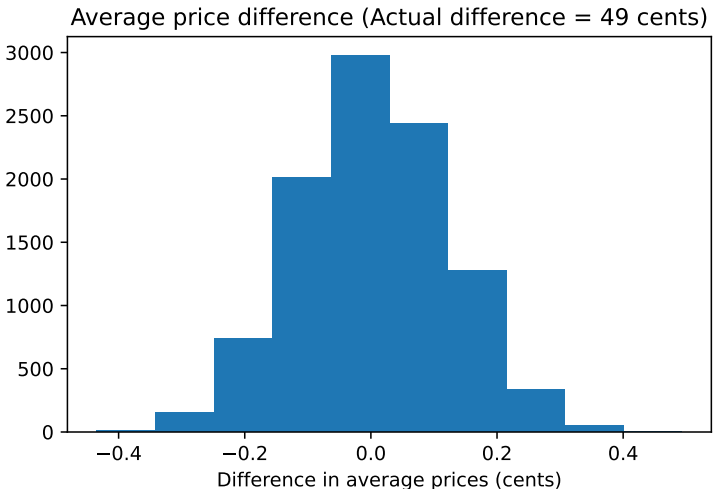
12.15.3 Price simulation

Now we have the machinery to concatenate arrays, and make histograms, we are ready to do the price simulation.

> Note 25: Notebook: Public and private liquor prices
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()

# Import the plotting library
import matplotlib.pyplot as plt
```

```
fake_diffs = np.zeros(10000)

priv = np.array([
    4.82, 5.29, 4.89, 4.95, 4.55, 4.90, 5.25, 5.30, 4.29, 4.85, 4.54, 4.75,
    4.85, 4.85, 4.50, 4.75, 4.79, 4.85, 4.79, 4.95, 4.95, 4.75, 5.20, 5.10,
    4.80, 4.29])

govt = np.array([
    4.65, 4.55, 4.11, 4.15, 4.20, 4.55, 3.80, 4.00, 4.19, 4.75, 4.74, 4.50,
    4.10, 4.00, 5.05, 4.20])

actual_diff = np.mean(priv) - np.mean(govt)

# Join the two arrays of data into one array.
both = np.concatenate([priv, govt])

# Repeat 10000 simulation trials
for i in range(10000):

    # Sample 26 with replacement for private group
    fake_priv = np.random.choice(both, size=26)

    # Sample 16 with replacement for govt. group
    fake_govt = np.random.choice(both, size=16)

    # Find the mean of the "private" group.
    p = np.mean(fake_priv)

    # Mean of the "govt." group
    g = np.mean(fake_govt)

    # Difference in the means
    diff = p - g

    # Keep score of the trials
    fake_diffs[i] = diff

# Graph of simulation results to compare with the observed result.
plt.hist(fake_diffs)
plt.xlabel('Difference in average prices (cents)')
plt.title('Average price difference (Actual difference = '
f'{actual_diff * 100:.0f} cents)');
```



ℹ End of notebook: Public and private liquor prices

`liquor_prices` starts at Note 25.

The results shown above — not even one "success" in 10,000 trials — imply that there is only a very small probability that two groups with mean prices as different as were observed would happen by chance if drawn with replacement from the universe of 42 observed prices.

Here we think of these states as if they came from a non-finite universe, which is one possible interpretation for one particular context. However, in Chapter 13 we will postulate a finite universe, which is appropriate if it is reasonable to consider that these observations constitute the entire universe (aside from those states excluded from the analysis because of data complexities).

12.16 The general procedure

Chapter 25 generalizes what we have done in the probability problems above into a general procedure, which will in turn be a subpart of a general procedure for all of resampling.

13 Probability Theory, Part 4: Estimating Probabilities from Finite Universes

13.1 Introduction

The examples in Chapter 12 dealt with *infinite universes*, in which the probability of a given simple event is unaffected by the outcome of the previous simple event. But now we move on to finite universes, situations in which you begin with a *given set of objects* whose number is not enormous — say, a total of two, or two hundred, or two thousand. If we liken such a situation to a bucket containing balls of different colors each with a number on it, we are interested in the probability of drawing various sets of numbered and colored balls from the bucket on the condition that we *do not replace* balls after they are drawn.

In the cases addressed in this chapter, it is important to remember that the single events no longer are independent of each other. A typical situation in which sampling without replacement occurs is when items are chosen from a finite universe — for example, when children are selected randomly from a classroom. If the class has five boys and five girls, and if you were to choose three girls in a row, then the chance of selecting a fourth girl on the next choice obviously is lower than the chance that you would pick a girl on the first selection.

The key to dealing with this type of problem is the same as with earlier problems: You must choose a simulation procedure that produces simple events having the same probabilities as the simple events in the actual problem involving sampling without replacement. That is, you must make sure that your simulation does not allow duplication of events that have already occurred. The easiest way to sample without replacement with resampling techniques is by simply ignoring an outcome if it has already occurred.

Examples Section 13.3.1 through Section 13.3.10 deal with some of the more important sorts of questions one may ask about drawings without replacement from such an urn. To get an overview, I suggest that you read over the summaries (in **bold**) introducing examples Section 13.3.1 to Section 13.3.10 before beginning to work through the examples themselves.

This chapter also revisits the general procedure used in solving problems in probability and statistics with simulation, here in connection with problems involving a finite universe. The steps that one follows in simulating the behavior of a universe of interest are set down in such fashion that one may, by random drawings, deduce the probability of various events. Having had by now the experience of working through the problems in Chapter 9 and Chapter 12, the

reader should have a solid basis to follow the description of the general procedure which then helps in dealing with specific problems.

Let us begin by describing some of the major sorts of problems with the aid of a bucket with six balls.

13.2 Some building-block programs

Case 1. Each of six balls is labeled with a number between "1" and "6." We ask: What is the probability of choosing balls 1, 2, and 3 *in that order* if we choose three balls without replacement? Figure 13.1 diagrams the events we consider "success."

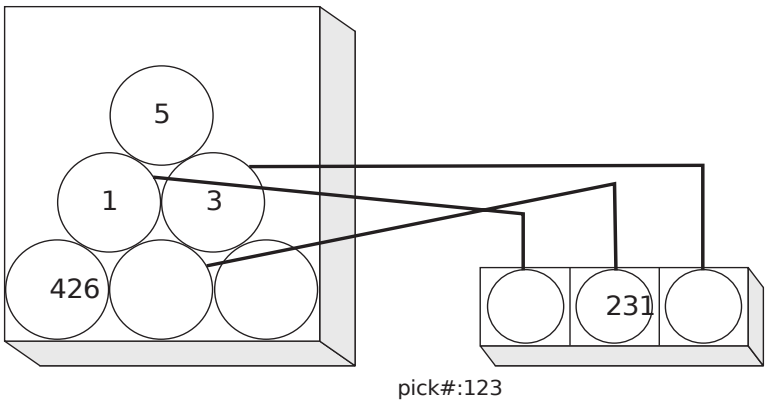


Figure 13.1: The Event Classified as "Success" for Case 1

Case 2. We begin with the same bucket as in Case 1, but now ask the probability of choosing balls 1, 2, and 3 *in any order* if we choose three balls without replacement. Figure 13.2 diagrams two of the events we consider success. These possibilities include that which is shown in Figure 13.1 above, plus other possibilities.

Case 3. The odd-numbered balls "1," "3," and "5," are painted red and the even-numbered balls "2," "4," and "6" are painted black. What is the probability of getting a red ball and then a black ball in that order? Some possibilities are illustrated in Figure 13.3, which includes the possibility shown in Figure 13.1. It also includes *some but not* all possibilities found in Figure 13.2; for example, Figure 13.2 includes choosing balls 2, 3 and 1 in that order, but Figure 13.3 does not.

Case 4. What is the probability of getting two red balls and one black ball in any order?

Case 5. Various questions about *matching* may be asked with respect to the six balls. For example, what is the probability of getting ball 1 on the first draw *or* ball 2 on the second

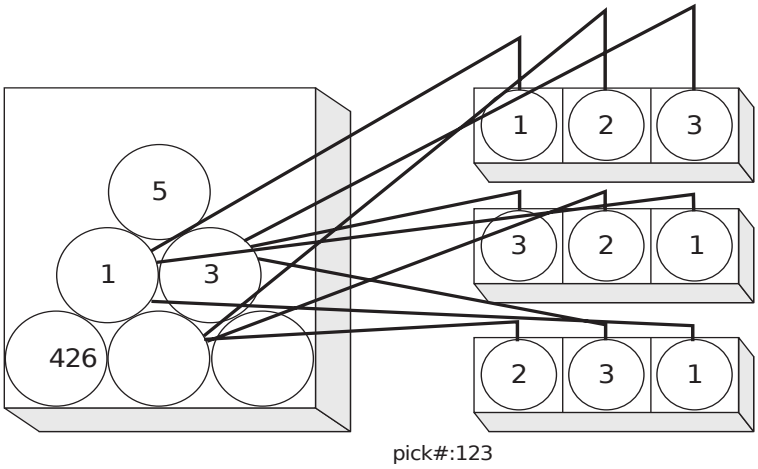


Figure 13.2: An Incomplete List of the Events Classified as “Success” for Case 2

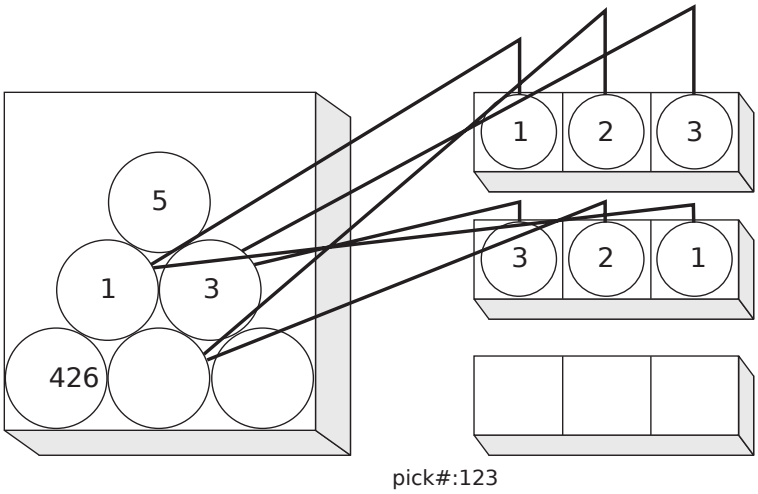


Figure 13.3: An Incomplete List of the Events Classified as “Success” for Case 3

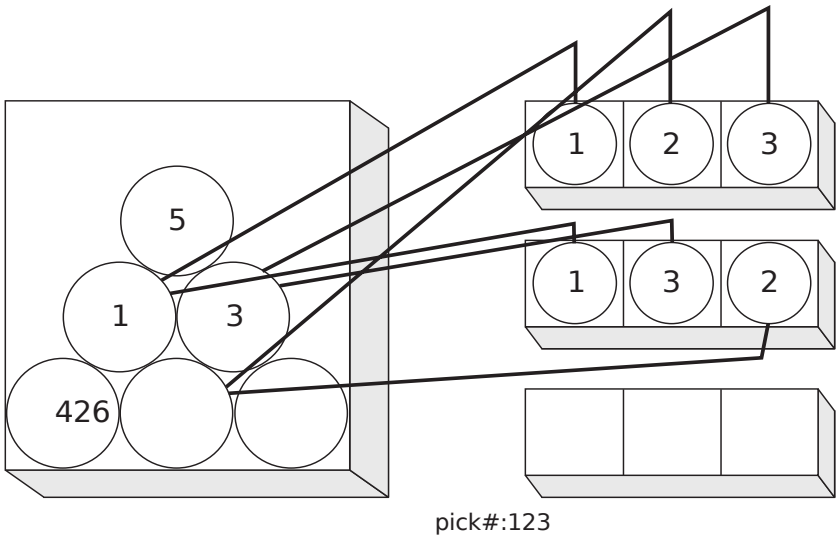


Figure 13.4: An Incomplete List of the Events Classified as "Success" for Case 4

draw *or* ball 3 on the third draw? (Figure 13.5) Or, what is the probability of getting all balls on the draws corresponding to their numbers?

13.3 Problems in finite universes

13.3.1 Example: four girls and one boy

What is the probability of selecting four girls and one boy when selecting five students from any group of twenty-five girls and twenty-five boys? This is an example of sampling without replacement when there are two outcomes and the order does not matter.

The important difference between this example and the infinite-universe examples in the prior chapter is that the probability of obtaining a boy or a girl in a single simple event *differs* from one event to the next in this example, whereas it stays the same when the sampling is with replacement. To illustrate, the probability of a girl is .5 (25 out of 50) when the first student is chosen, but the probability of a girl is either 25/49 or 24/49 when the second student is chosen, depending on whether a boy or a girl was chosen on the first pick. Or after, say, three girls and one boy are picked, the probability of getting a girl on the next choice is (28-3)/(50-4) = 22/46 which is clearly not equal to .5.

As always, we must create a satisfactory analog to the process whose probability we want to learn. In this case, we can use a deck of 50 cards, half red and half black, and deal out five cards *without replacing them* after each card is dealt; this simulates the choice of five students from among the fifty.

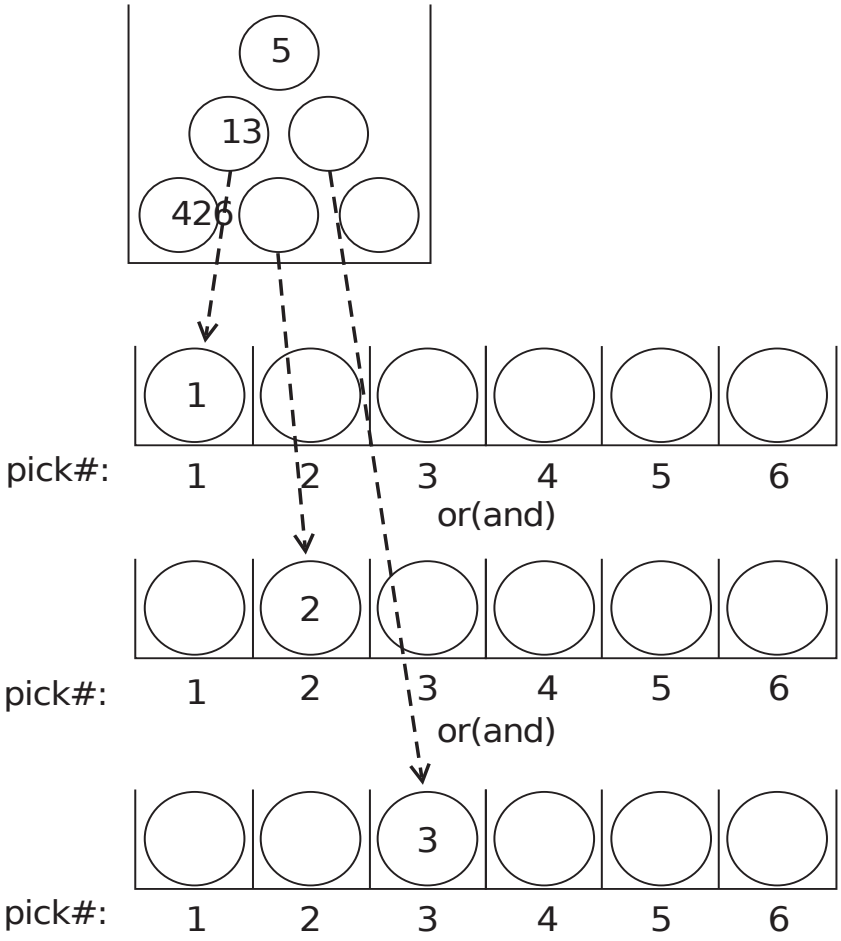


Figure 13.5: An Incomplete List of the Events Classified as “Success” for Case 5

We can no longer use our procedure from before. If we designated "1-25" as being girls and "26-50" as being boys and then proceeded to draw random numbers, the probability of a girl would be the same on each pick.

At this point, it is important to note that — for this particular problem — we do not need to distinguish between particular girls (or boys). That is, it does not matter *which* girl (or boy) is selected in a given trial. Nor did we pay attention to the *order* in which we selected girls or boys. This is an instance of Case 4 discussed above. Subsequent problems will deal with situations where the order of selection, and the particular individuals, do matter.

Our approach then is to mimic having the class in front of us: an array of 50 strings, half of the entries 'boy' and the other half 'girl'. We then shuffle the class (the array), and choose the first N students (strings).

- **Step 1.** Create a list with 50 labels, half 'boy' and half 'girl'.
- **Step 2.** Shuffle the class and select five students. Count whether there are four labels equal 'girl'. If so, write "yes," otherwise "no".
- **Step 3.** Repeat step 2, say, 10,000 times, and count the proportion "yes", which estimates the probability sought.

The results of a few experimental trials are shown in Table 13.1.

Table 13.1: A few experimental trials of four girls and one boy

Experiment	Strings Chosen	Success?
1	'girl', 'boy', 'boy', 'girl', 'boy'	No
2	'boy', 'girl', 'girl', 'girl', 'girl'	Yes
3	'girl, 'girl', 'girl', 'boy', 'girl'	Yes

A solution to this problem with Python is presented below.

> **Note 26: Notebook: Four girls and one boy**
>
> - Download notebook
> - Interact

```
import numpy as np

rnd = np.random.default_rng()
```

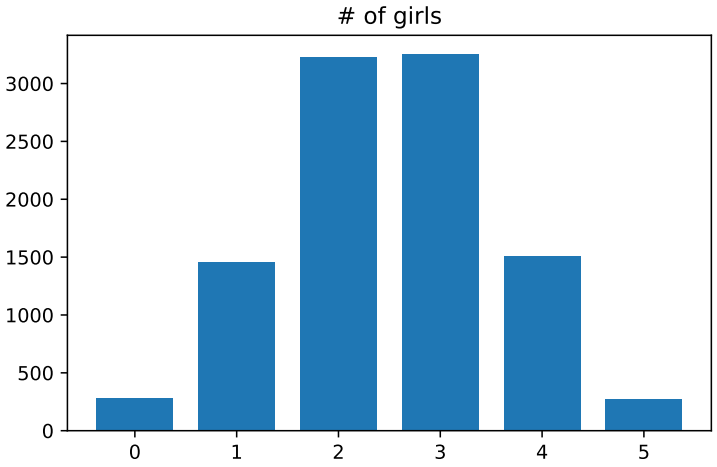
```
N = 10000
trial_results = np.zeros(N)

# Constitute the set of 25 girls and 25 boys.
whole_class = np.repeat(['girl', 'boy'], [25, 25])

# Repeat the following steps N times.
for i in range(N):

    # Shuffle the numbers
    shuffled = rnd.permuted(whole_class)

    # Take the first 5 numbers, call them c.
    c = shuffled[:5]

    # Count how many girls there are, put the result in d.
    d = np.sum(c == 'girl')

    # Keep track of each trial result in z.
    trial_results[i] = d

    # End the experiment, go back and repeat until all 1000 trials are
    # complete.

# Count the number of times we got four girls, put the result in k.
k = np.sum(trial_results == 4)

# Convert to a proportion.
kk = k / N

# Print the result.
print(kk)
```

```
0.1505
```

We can also find the probabilities of other outcomes from a histogram of trial results obtained with the following command:

```
# Import the plotting package.
import matplotlib.pyplot as plt
```

```
# Do histogram, with one bin for each possible number.
plt.hist(trial_results, bins=range(7), align='left', rwidth=0.75)
plt.title('# of girls');
```



In the resulting histogram we can see that in 15 percent of the trials, 4 of the 5 selected were girls.

It should be noted that for this problem — as for most other problems — there are several other resampling procedures that will also do the job correctly.

In analytic probability theory this problem is worked with a formula for "combinations."

> **i** End of notebook: Four girls and one boy
>
> `four_girls_one_boy` starts at Note 26.

13.3.2 Example: Five spades and four clubs in a bridge hand

> **i** Note 27: Notebook: Five spades and four clubs
>
> - Download notebook
> - Interact

This is an example of multiple-outcome sampling without replacement, order does not matter.

The problem is similar to the example in Section 13.3.1, except that now there are four equally-likely outcomes instead of only two. A Python solution is:

```
import numpy as np

rnd = np.random.default_rng()
```

```
# Constitute the deck of 52 cards.
# Repeat the suit names 13 times each, to make a 52 card deck.
deck = np.repeat(['spade', 'club', 'diamond', 'heart'],
                 [13, 13, 13, 13])
# Show the deck
deck
```

```
array(['spade', 'spade', 'spade', 'spade', 'spade', 'spade', 'spade',
       'spade', 'spade', 'spade', 'spade', 'spade', 'spade', 'club',
       'club', 'club', 'club', 'club', 'club', 'club', 'club', 'club',
       'club', 'club', 'club', 'club', 'diamond', 'diamond', 'diamond',
       'diamond', 'diamond', 'diamond', 'diamond', 'diamond', 'diamond',
       'diamond', 'diamond', 'diamond', 'diamond', 'heart', 'heart',
       'heart', 'heart', 'heart', 'heart', 'heart', 'heart', 'heart',
       'heart', 'heart', 'heart', 'heart'], dtype='<U7')
```

```
N = 10000
trial_results = np.zeros(N)

# Repeat the trial N times.
for i in range(N):

    # Shuffle the deck and draw 13 cards.
    hand = rnd.choice(deck, size=13, replace=False)

    # Count the number of spades in "hand", put the result in "n_spades".
    n_spades = np.sum(hand == 'spade')

    # If we have five spades, we'll continue on to count the clubs. If we don't
    # have five spades, the number of clubs is irrelevant - we have not gotten
    # the hand we are interested in.
    if n_spades == 5:
        # Count the clubs, put the result in "n_clubs"
        n_clubs = np.sum(hand == 'club')
        # Keep track of the number of clubs in each trial
```

```
        trial_results[i] = n_clubs

    # End one experiment, go back and repeat until all N trials are done.

# Count the number of trials where we got 4 clubs. This is the answer we want -
# the number of hands out of 1000 with 5 spades and 4 clubs. (Recall that we
# only counted the clubs if the hand already had 5 spades.)
n_5_and_4 = np.sum(trial_results == 4)

# Convert to a proportion.
prop_5_and_4 = n_5_and_4 / N

# Print the result
print(prop_5_and_4)
```

```
0.0224
```

> **i** End of notebook: Five spades and four clubs
>
> `five_spades_four_clubs` starts at Note 27.

13.3.3 Example: a total of fifteen points in a bridge hand

> **i** Note 28: Notebook: Fifteen points in a bridge hand
>
> - Download notebook
> - Interact

Let us assume that ace counts as 4, king = 3, queen = 2, and jack = 1.

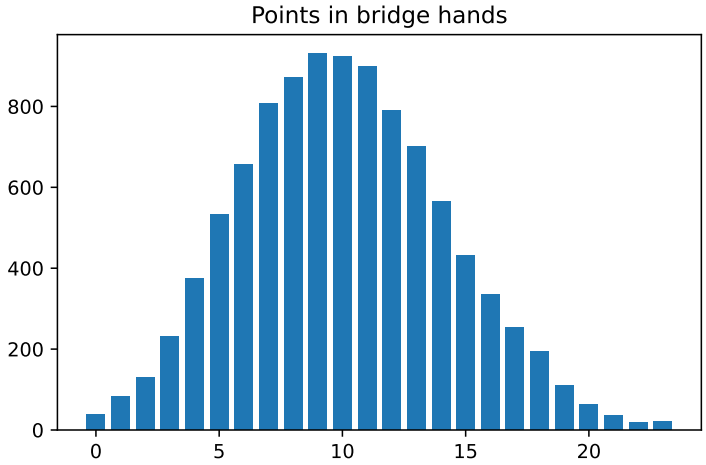
```
import numpy as np

rnd = np.random.default_rng()

import matplotlib.pyplot as plt
```

```
# Constitute a deck with 4 jacks (point value 1), 4 queens (value 2), 4
# kings (value 3), 4 aces (value 4), and 36 other cards with no point
# value
whole_deck = np.repeat([1, 2, 3, 4, 0], [4, 4, 4, 4, 36])
whole_deck
```

```
array([1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0])
```

```
N = 10000
trial_results = np.zeros(N)

# Do N trials.
for i in range(N):
    # Shuffle the deck of cards and draw 13
    hand = rnd.choice(whole_deck, size=13, replace=False)

    # Total the points.
    points = np.sum(hand)

    # Keep score of the result.
    trial_results[i] = points

    # End one experiment, go back and repeat until all N trials are done.
```

```
# Produce a histogram of trial results.
plt.hist(trial_results, bins=range(25), align='left', rwidth=0.75)
plt.title('Points in bridge hands');
```



From this histogram, we see that in about 4 percent of our trials we obtained a total of exactly 15 points. We can also compute this directly:

```
# How many times did we have a hand with fifteen points?
k = np.sum(trial_results == 15)

# Convert to a proportion.
kk = k / N

# Show the result.
kk
```

```
np.float64(0.0431)
```

> **i** End of notebook: Fifteen points in a bridge hand
>
> `fifteen_points_in_bridge` starts at Note 28.

13.3.4 Example: Four girls then one boy from 25 girls and 25 boys

> Note 29: Notebook: Four girls then one boy from 25/25
>
> - Download notebook
> - Interact

In this problem, order matters; we are sampling without replacement, with two outcomes, several of each item.

What is the probability of getting an ordered series of *four girls and then one boy*, from a universe of 25 girls and 25 boys? This illustrates Case 3 above. Clearly we can use the same sampling mechanism as in the example Section 13.3.1, but now we record "yes" for a smaller number of composite events.

We record "no" even if a single one boy is chosen but he is chosen 1st, 2nd, 3rd, or 4th, whereas in Section 13.3.1, such outcomes are recorded as "yes"-es.

- **Step 1.** Generate a class (array) of length 50, consisting of 25 strings valued "boy" and 25 strings valued "girl".
- **Step 2.** Shuffle the class array, and select the first five elements.
- **Step 3.** If the first five elements are exactly `'girl', 'girl', 'girl', 'girl', 'boy'`, write "yes," otherwise "no."
- **Step 4.** Repeat steps 2 and 3, say, 10,000 times, and count the proportion of "yes" results, which estimates the probability sought.

Let us start the single trial procedure like so:

```
import numpy as np

rnd = np.random.default_rng()
```

```
# Constitute the set of 25 girls and 25 boys.
whole_class = np.repeat(['girl', 'boy'], [25, 25])

# Shuffle the class into a random order.
shuffled = rnd.permuted(whole_class)
# Take the first 5 class members, call them c.
c = shuffled[:5]
# Show the result.
c
```

```
array(['boy', 'girl', 'boy', 'girl', 'girl'], dtype='<U4')
```

Our next step (step 3) is to check whether `c` is exactly equal to the result of interest. The result of interest is:

```
# The result we are looking for - four girls and then a boy.
result_of_interest = np.repeat(['girl', 'boy'], [4, 1])
result_of_interest
```

```
array(['girl', 'girl', 'girl', 'girl', 'boy'], dtype='<U4')
```

We can then use an array *comparison* with `==` to do an element by element (*elementwise*) check, asking whether the corresponding elements are equal:

```
# A Boolean array, with True where corresponding elements are equal, False
# otherwise.
are_equal = c == result_of_interest
are_equal
```

```
array([False,  True, False,  True, False])
```

We are nearly finished with step 3 — it only remains to check whether *all* of the elements were equal, by checking whether *all* of the values in `are_equal` are `True`.

We know that there are 5 elements, so we could check whether there are 5 `True` values with `np.sum`:

```
# Are there exactly 5 True values in `are_equal`?
np.sum(are_equal) == 5
```

```
np.False_
```

Another way to ask the same question is by using the `np.all` function on `are_equal`. This returns `True` if *all* the elements in `are_equal` are `True`, and `False` otherwise.

> **Note 30: Testing whether all elements of an array are the same**
>
> The `np.all`, applied to a Boolean array (as here), checks whether *all* of the elements in the Boolean array are `True`. If so, it returns `True`, otherwise, it returns `False`.
> For example:

```
# All elements are True, `np.all` returns True
np.all([True, True, True, True])
```

```
np.True_
```

```
# At least one element is False, `np.all` returns False
np.all([True, True, False, True])
```

```
np.False_
```

Here is the full procedure for steps 2 and 3 (a single trial):

```
# Shuffle the class into a random order.
shuffled = rnd.permuted(whole_class)
# Take the first 5 class members, call them c.
c = shuffled[:5]
# For each element, test whether the result is the result of interest.
are_equal = c == result_of_interest
# Check whether we have the result we are looking for.
is_four_girls_then_one_boy = np.all(are_equal)
```

All that remains is to put the single trial procedure into a loop.

```
N = 10000
trial_results = np.zeros(N)

# Repeat the following steps 1000 times.
for i in range(N):

    # Shuffle the class into a random order.
    shuffled = rnd.permuted(whole_class)
    # Take the first 5 class members, call them c.
    c = shuffled[:5]
    # For each element, test whether the result is the result of interest.
    are_equal = c == result_of_interest
    # Check whether we have the result we are looking for.
    is_four_girls_then_one_boy = np.all(are_equal)

    # Store the result of this trial.
    trial_results[i] = is_four_girls_then_one_boy
```

```
    # End the experiment, go back and repeat until all N trials are
    # complete.

# Count the number of times we got four girls then a boy
k = np.sum(trial_results)

# Convert to a proportion.
kk = k / N

# Print the result.
print(kk)
```

```
0.0311
```

This type of problem is conventionally done with a *permutation* formula.

> End of notebook: Four girls then one boy from 25/25
>
> `four_girls_then_one_boy_25` starts at Note 29.

13.3.5 Example: repeat pairings from random pairing

> Note 31: Notebook: An icebreaker for two universities
>
> - Download notebook
> - Interact

First put two groups of 10 people into 10 pairs. Then re-randomize the pairings. What is the chance that four or more pairs are the same in the second random pairing? This is a problem in the probability of matching by chance.

Ten representatives each from two universities, Birmingham and Berkeley, attend a meeting. As a social icebreaker, representatives are divided, randomly, into pairs consisting of one person from each university.

If they held a second round of the icebreaker, with a new random pairing, what is the chance that four or more pairs will be the same?

In approaching this problem, we start at the point where the first icebreaker is complete. We now have to determine what happens after the second round.

- **Step 1.** Let "ace" through "10" of hearts represent the ten representatives from Birmingham University. Let "ace" through "10" of spades be their allocated partners (in round one) from Berkeley.
- **Step 2.** Shuffle the hearts and deal them out in a row; shuffle the spades and deal in a row just below the hearts.
- **Step 3.** Count the pairs — a pair is one card from the heart row and one card from the spade row — that contain the same denomination. If 4 or more pairs match, record "yes," otherwise "no."
- **Step 4.** Repeat steps (2) and (3), say, 10,000 times.
- **Step 5.** Count the proportion "yes." This estimates the probability of 4 or more pairs.

Exercise for the student: Write the steps to do this example with random numbers. The Python solution follows below.

```
import numpy as np

rnd = np.random.default_rng()

import matplotlib.pyplot as plt
```

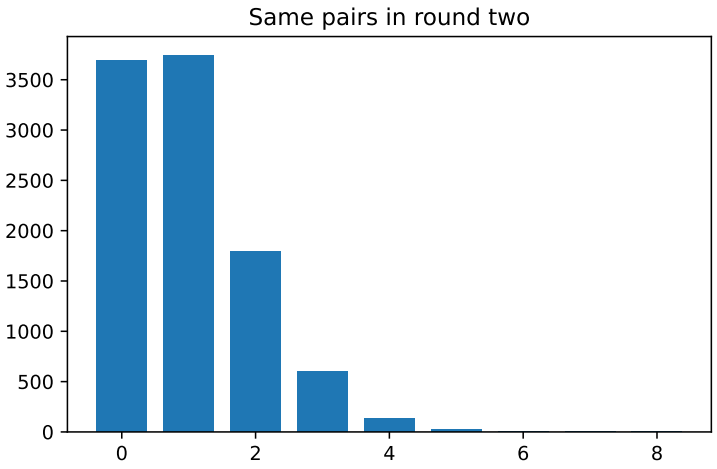
```
N = 10000
trial_results = np.zeros(N)

# Assign numbers to each student, according to their pair, after the first
# icebreaker
birmingham = np.arange(10)
berkeley = np.arange(10)

for i in range(N):
    # Randomly shuffle the students from Berkeley
    shuffled_berkeley = rnd.permuted(berkeley)

    # Randomly shuffle the students from Birmingham
    # (This step is not really necessary - shuffling one array is enough to make the matching
    shuffled_birmingham = rnd.permuted(birmingham)

    # Count in how many cases people landed with the same person as in the
    # first round, and store in trial_results.
    matches = np.sum(shuffled_berkeley == shuffled_birmingham)
    trial_results[i] = matches

# Count the number of times we got 4 or more people assigned to the same person
k = np.sum(trial_results >= 4)
```

```
# Convert to a proportion.
kk = k / N

# Print the result.
print(kk)
```

```
0.0165
```

We see that in about 2 percent of the trials did 4 or more couples end up being re-paired with their own partners. This can also be seen from the histogram:

```
# Produce a histogram of trial results.
plt.hist(trial_results, bins=range(10), align='left', rwidth=0.75)
plt.title('Same pairs in round two');
```



i End of notebook: An icebreaker for two universities

`university_icebreaker` starts at Note 31.

13.3.6 Example: Matching Santa Hats

> Note 32: Notebook: Santas' hats
>
> - Download notebook
> - Interact

The welcome staff at a restaurant mix up the hats of a party of six Christmas Santas. What is the probability that at least one will get their own hat?.

After a long Christmas day, six Santas meet in the pub to let off steam. However, as luck would have it, their hosts have mixed up their hats. When the hats are returned, what is the chance that at least one Santa will get his own hat back?

First, assign each of the six Santas a number, and place these numbers in an array. Next, shuffle the array (this represents the mixed-up hats) and compare to the original. The rest of the problem is the same as the pairs one from before, except that we are now interested in any trial where at least one (≥ 1) Santa received the right hat.

```
import numpy as np

rnd = np.random.default_rng()
```

```
N = 10000
trial_results = np.zeros(N, dtype=bool)

# Assign numbers to each owner
owners = np.arange(6)

# Each hat gets the number of their owner
hats = np.arange(6)

for i in range(N):
    # Randomly shuffle the hats and compare to their owners
    shuffled_hats = rnd.permuted(hats)

    # In how many cases did at least one person get their hat back?
    trial_results[i] = np.sum(shuffled_hats == owners) >= 1

# How many times, over all trials, did at least one person get their hat back?
k = np.sum(trial_results)

# Convert to a proportion.
```

```
kk = k / N

# Print the result.
print(kk)
```

```
0.6391
```

We see that in roughly 64 percent of the trials at least one Santa received their own hat back.

> End of notebook: Santas' hats
>
> `santas_hats` starts at Note 32.

13.3.7 Example: Twenty executives assigned to two divisions of a firm

> Note 33: Notebook: Twenty executives, two divisions
>
> - Download notebook
> - Interact

The top manager wants to spread the talent reasonably evenly, but she does not want to label particular executives with a quality rating and therefore considers distributing them with a random selection. She therefore wonders: What are probabilities of the best ten among the twenty being split among the divisions in the ratios 5 and 5, 4 and 6, 3 and 7, etc., if their names are drawn from a hat? One might imagine much the same sort of problem in choosing two teams for a football or baseball contest.

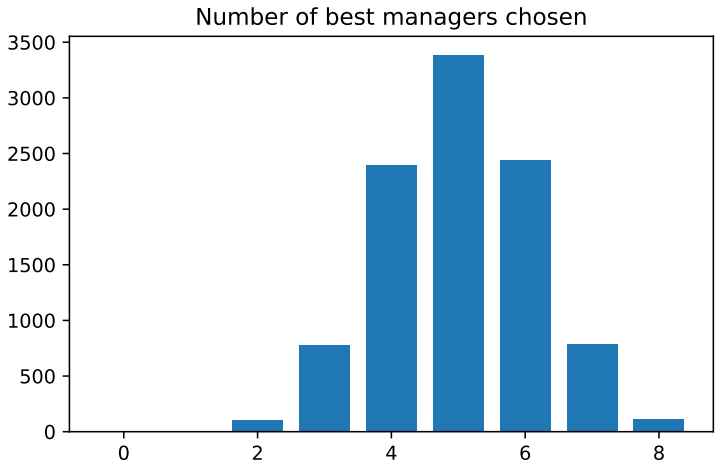
One may proceed as follows:

1. Put 10 balls labeled "W" (for "worst") and 10 balls labeled "B" (best) in a bucket.
2. Draw 10 balls without replacement and count the W's.
3. Repeat (say) 400 times.
4. Count the number of times each split — 5 W's and 5 B's, 4 and 6, etc. — appears in the results.

The problem can be done with Python as follows:

```
import numpy as np

rnd = np.random.default_rng()

import matplotlib.pyplot as plt
```

```
N = 10000
trial_results = np.zeros(N)

managers = np.repeat(['Worst', 'Best'], [10, 10])

for i in range(N):
    chosen = rnd.choice(managers, size=10, replace=False)
    trial_results[i] = np.sum(chosen == 'Best')

plt.hist(trial_results, bins=range(10), align='left', rwidth=0.75)
plt.title('Number of best managers chosen')
```



End of notebook: Twenty executives, two divisions

`twenty_executives` starts at Note 33.

13.3.8 Example: Executives Moving

A major retail chain moves its store managers from city to city every three years in order to calculate individuals' knowledge and experience. To make the procedure seem fair, the new locations are drawn at random. Nevertheless, the movement is not popular with managers' families. Therefore, to make the system a bit sporting and to give people some hope of remaining in the same location, the chain allows managers to draw in the lottery the same posts they are now in. What are the probabilities that 1, 2, 3 … will get their present posts again if the number of managers is 30?

The problem can be solved with the following steps:

1. Number a set of green balls from "1" to "30" and put them into Bucket A. Number a set of red balls from "1" to "30" and then put into Bucket B. For greater concreteness one could use 30 little numbered dolls in Bucket A and 30 little toy houses in Bucket B.
2. Shuffle Bucket A, and array all its green balls into a row (vector A). Array all the red balls from Bucket B into a second row B just below row A.
3. Count how many green balls in row A have the same numbers as the red balls just below them, and record that number on a scoreboard.
4. Repeat steps 2 and 3 perhaps 1000 times. Then count in the scoreboard the numbers of "0," "1," "2," "3."

13.3.9 Example: State Liquor Systems Again

Let's end this chapter with the example of state liquor systems that we first examined in Chapter 12 and which will be discussed again later in the context of problems in statistics.

Remember that as of 1963, there were 26 U.S. states in whose liquor systems the retail liquor stores are privately owned ("Private"), and 16 *monopoly* states where the state government owns the retail liquor stores ("Government"). See Table 12.6 for the prices in the Private and Government states.

We found the average prices were:

- Private: $4.35;
- Government: $4.84;
- Difference (Government - Private): $0.49.

Let us now consider that all these states' prices constitute one single *finite* universe. We ask: If these 42 states constitute a universe, and if they are all shuffled together, how likely is it that if one divides them into two samples at random (sampling without replacement), containing 16 and 26 observations respectively, the difference in mean prices turns out to be as great as $0.49 (the difference that was actually observed)?

Again we write each of the forty-two observed state prices on a separate card. The shuffled deck simulates a situation in which each state has an equal chance for each price. Repeatedly deal groups of 16 and 26 cards, *without replacing* the cards as they are chosen, to simulate hypothetical monopoly-state and private-state samples. In each trial calculate the difference in mean prices.

The steps more systematically:

- **Step A.** Write each of the 42 prices on a card and shuffle.
- **Steps B and C (combined in this case).** i) Draw cards randomly without replacement into groups of 16 and 26 cards. Then ii) calculate the mean price difference between the groups, and iii) compare the simulation-trial difference to the observed mean difference of $4.84 - $4.35 = $0.49; if it is as great or greater than $0.49, write "yes," otherwise "no."
- **Step D.** Repeat step B-C a hundred or a thousand times. Calculate the proportion "yes," which estimates the probability we seek.

The probability that the postulated universe would produce a difference between groups as large or larger than observed in 1961 is estimated by how frequently the mean of the group of randomly-chosen sixteen prices from the simulated state ownership universe is less than (or equal to) the mean of the actual sixteen state-ownership prices.

Please notice how the only difference between this treatment of the problem and the treatment in Chapter 12 is that the drawing in this case is without replacement whereas in Chapter 12 the drawing is with replacement.

In Chapter 12 we thought of these states as if they came from a non-finite universe, which is one possible interpretation in one context. But one can also reasonably think about them in another context — as if they constitute the entire universe (aside from those states excluded from the analysis because of data complexities). If so, one can ask: If these 42 states constitute a universe, how likely is it that one would choose two samples at random, containing 16 and 26 observations, that would have prices as different as $.49 (the difference that was actually observed)?

13.3.10 Example: Five or More Spades in One Bridge Hand; Four Girls and a Boy

> Note 34: Notebook: Five spades, four girls
>
> - Download notebook
> - Interact

This is a compound problem: what are the chances of *both* five or more spades in one bridge hand, and four girls and a boy in a five-child family?

"Compound" does not necessarily mean "complicated". It means that the problem is a compound of two or more simpler problems.

A natural way to handle such a compound problem is in stages, as we saw in the archery problem of Section 12.9. If a "success" is achieved in the first stage, go on to the second stage; if not, don't go on. More specifically in this example:

- **Step 1.** Use a bridge card deck, and five coins with heads = "girl".
- **Step 2.** Deal a 13-card bridge hand and count the spades. If 5 or more spades, record "no" and end the experimental trial. Otherwise, continue to step 3.
- **Step 3.** Throw five coins, and count "heads." If four heads, record "yes," otherwise record "no."
- **Step 4.** Repeat steps 2 and 3 a thousand times.
- **Step 5.** Compute the proportion of "yes" in step 3. This estimates the probability sought.

The Python solution to this compound problem is neither long nor difficult. We tackle it almost as if the two parts of the problem were to be dealt with separately. We first determine, in a random bridge hand, whether 5 spades or more are dealt, as was done in the problem Section 13.3.2. Then, `if` 5 or more spades are found, we use `rnd.choice` to generate a random family of 5 children. This means that we need not generate families if 5 or more spades were not dealt to the bridge hand, because a "success" is only recorded if both conditions are met. After we record the number of girls in each sample of 5 children, we need only finish the loop (by unindenting the next line) and then use `np.sum` to count the number of samples that had 4 girls, storing the result in `k`. Since we only drew samples of children for those trials in which a bridge hand of 5 spades had already been dealt, `k` will have the number of trials out of 10000 in which both conditions were met.

```
import numpy as np

rnd = np.random.default_rng()
```

```
N = 10000
trial_results = np.zeros(N)

# Deck with 13 spades and 39 other cards
deck = np.repeat(['spade', 'others'], [13, 52 - 13])

for i in range(N):
    # Shuffle deck and draw 13 cards
    hand = rnd.choice(deck, size=13, replace=False)

    n_spades = np.sum(hand == 'spade')
```

```
    if n_spades >= 5:
        # Generate a family, zeros for boys, ones for girls
        children = rnd.choice(['girl', 'boy'], size=5)
        n_girls = np.sum(children == 'girl')
        trial_results[i] = n_girls

k = np.sum(trial_results == 4)

kk = k / N

print(kk)
```

```
0.0282
```

Here is an alternative approach to the same problem, but getting the result at the end of the loop, by combining Boolean arrays (see Section 10.6).

```
N = 10000
trial_spades = np.zeros(N)
trial_girls = np.zeros(N)

# Deck with 13 spades and 39 other cards
deck = np.repeat(['spade', 'other'], [13, 39])

for i in range(N):
    # Shuffle deck and draw 13 cards
    hand = rnd.choice(deck, 13, replace=False)

    n_spades = np.sum(hand == 'spade')
    trial_spades[i] = n_spades

    # Generate a family, zeros for boys, ones for girls
    children = rnd.choice(['girl', 'boy'], size=5)
    n_girls = np.sum(children == 'girl')
    trial_girls[i] = n_girls

k = np.sum((trial_spades >= 5) & (trial_girls == 4))

kk = k / N

print(kk)
```

```
0.0264
```

> **End of notebook: Five spades, four girls**
>
> `five_spades_four_girls` starts at Note 34.

> **Speed and readability**
>
> The last version is a fraction more expensive, but has the advantage that the condition we are testing for is summarized on one line. However, this would not be a good approach to take if the experiments were not completely unrelated.

13.4 Summary

This completes the discussion of problems in probability — that is, problems where we assume that the structure is known. Whereas Chapter 12 dealt with samples drawn from universes considered *not finite*, this chapter deals with problems drawn from *finite universes* and therefore you *sample without replacement*.

14 On Variability in Sampling

> [Debra said]: "I've had such good luck with Japanese cars and poor luck with American..."
>
> The '65 Ford Mustang: "It was fun, but I had to put two new transmissions in it."
>
> The Ford Torino: "That got two transmissions too. That finished me with Ford."
>
> The Plymouth Horizon: "The disaster of all disasters. That should've been painted bright yellow. What a lemon."

(From *Washington Post Magazine*, May 17, 1992, p. 19)

Do the quotes above convince you that Japanese cars are better than American? Has Debra got enough evidence to reach the conclusion she now holds? That sort of question, and the reasoning we use to address it, is the subject of this chapter.

More generally, how should one go about using the available data to test the hypothesis that Japanese cars are better? That is an example of the questions that are the subject of statistics.

14.1 Variability and small samples

Perhaps the most important idea for sound statistical inference — the section of the book we are now beginning, in contrast to problems in probability, which we have studied in the previous chapters — is recognition of the *presence of variability in the results of small samples*. The fatal error of relying on too-small samples is all too common among economic forecasters, journalists, and others who deal with trends and public opinion. Athletes, sports coaches, sportswriters, and fans too frequently disregard this principle both in their decisions and in their discussion.

Our intuitions often carry us far astray when the results vary from situation to situation — that is, when there is variability in outcomes — and when we have only a small sample of outcomes to look at.

To motivate the discussion, I'll tell you something that almost no American sports fan will believe: There is no such thing as a slump in baseball batting. That is, a batter often goes an

alarming number of at-bats without getting a hit, and everyone — the manager, the sportswriters, and the batter himself — assumes that something has changed, and the probability of the batter getting a hit is now lower than it was before the slump. It is common for the manager to replace the player for a while, and for the player and coaches to change the player's hitting style so as to remedy the defect. But the chance of a given batter getting a hit is just the same after he has gone many at-bats without a hit as when he has been hitting well. A belief in slumps causes managers to play line-ups which may not be their best.

By "slump" I mean that a player's probability of getting a hit in a given at-bat is lower during a period than during average periods. And when I say there is no such thing as a slump, I mean that the chances of getting a hit after any sequence of at-bats without a hit is *not* different than the long-run average.

The "hot hand" in basketball is another illusion. In practical terms, the hot hand does not exist — or rather — if it does, the effect is weak.[1] The chance of a shooter scoring is more or less the same after they have just missed a flock of shots as when they have just sunk a long string. That is, the chance of scoring a basket is not appreciably higher after a run of successes than after a run of failures. But even professional teams choose plays on the basis of who supposedly has a hot hand.

Managers who substitute for the "slumping" or "cold-handed" players with other players who, in the long run, have lower batting averages, or set up plays for the shooter who supposedly has a hot hand, make a mistake. The supposed hot hand in basketball, and the slump in baseball, are illusions because the observed long runs of outs, or of baskets, are statistical artifacts, due to ordinary random variability. The identification of slumps and hot hands is superstitious behavior, classic cases of the assignment of pattern to a series of events when there really is no pattern.

How do statisticians ascertain that slumps and hot hands are very weak effects, or do not exist? In brief, in baseball we simulate a hitter with a given average — say .250 — and compare the results with actual hitters of that average, to see whether they have "slumps" longer than the computer. The method of investigation is roughly as follows. You program a computer or other machine to behave the way a player would, given the player's long-run average, on the assumption that each trial is a random drawing. For example, if a player has a .250 season-long batting average, the machine is programmed like a bucket containing three black balls and one white ball. Then for each simulated at bat, the machine shuffles the "balls" and draws one; it then records whether the result is black or white, after which the ball is replaced in the bucket. To study a season with four hundred at-bats, a simulated ball is drawn four hundred times.

The records of the player's real season and the simulated season are then compared. If there really is such a thing as a non-random slump or streak, there will be fewer but longer "runs" of hits or outs in the real record than in the simulated record. On the other hand, if performance

[1] We are simplifying here; the Hot-Hand is a controversial topic, and it has its own Wikipedia page with information and discussion: https://en.wikipedia.org/wiki/Hot_hand. It may be overstating to say the effect does not exist, but if it does exist, the effect is much smaller than most sports fans believe it to be.

is independent from at-bat trial to at-bat trial, the actual record will change from hit to out and from out to hit as often as does the random simulated record. I suggested this sort of test for the existence of slumps in my 1969 book that first set forth the resampling method, a predecessor of this book.

For example, Table 14.1 shows the results of one 400 at-bat season for a simulated .250 hitter. (H = hit, O = out, sequential at-bats ordered vertically) Note the "slump" — 1 for 24 — in columns 7 & 8 (in bold).

Table 14.1: 400 simulated at-bats (ordered vertically)

O	O	O	O	O	O	H	**O**	O	O	O	H	O	H	O	O
O	O	O	O	O	H	O	**O**	H	H	H	O	H	H	O	O
O	O	O	H	O	O	O	**O**	H	O	O	O	H	H	O	O
O	O	O	O	O	H	H	**O**	O	O	O	H	O	O	O	H
H	O	H	O	O	H	O	**O**	O	H	O	O	O	O	H	O
H	O	O	H	O	O	H	**H**	O	H	O	O	H	O	H	O
O	O	H	O	O	O	**O**	H	O	O	O	O	O	O	H	O
O	O	H	O	O	O	**O**	H	H	O	O	O	O	O	O	O
O	H	O	O	O	O	**O**	O	H	H	O	O	O	H	O	O
O	H	H	O	O	O	**O**	H	O	H	O	O	H	O	H	O
O	O	H	H	O	H	**O**	H	O	H	H	H	O	O	O	O
H	O	O	O	O	O	**O**	O	O	H	O	H	H	O	O	O
O	H	O	O	O	H	**O**	O	O	O	O	O	O	O	H	H
H	O	H	O	O	O	**H**	O	O	O	O	H	H	O	O	H
O	O	O	O	H	H	**O**	O	O	O	O	H	H	H	H	O
O	O	O	O	H	H	**O**	O	O	O	O	H	O	O	O	O
H	O	O	O	O	O	**O**	O	O	O	O	O	O	O	O	O
O	H	H	H	O	O	**O**	H	O	H	O	O	O	O	O	O
O	H	O	H	O	O	**O**	O	H	O	O	O	O	H	O	O
O	O	O	H	H	O	**O**	O	O	O	H	O	H	O	O	H
O	H	O	O	H	O	**O**	O	O	O	H	O	O	O	O	O
H	H	H	O	O	O	**O**	H	O	O	O	O	H	O	O	H
O	O	O	H	H	O	**O**	O	O	O	O	O	O	O	H	O
O	H	O	O	O	O	**O**	H	H	O	O	O	O	O	O	H
O	O	O	O	O	H	**O**	O	O	H	O	H	O	H	O	O

Harry Roberts investigated the batting records of a sample of major leaguers.[2] He compared players' season-long records against the behavior of random-number drawings. If slumps existed rather than being a fiction of the imagination, the real players' records would shift from

[2]See Simon's description of these experiments in Batter's slump and other illusions, Washington Post, August 9th, 1987.

a string of hits to a string of outs less frequently than would the random-number sequences. But in fact the number of shifts, and the average lengths of strings of hits and outs, are on average the same for players as for player-simulating random-number devices.

Over long periods, averages may vary systematically, as Ty Cobb's annual batting averages varied non-randomly from season to season, Roberts found. But in the short run, most individual and team performances have shown results similar to the outcomes that a lottery-type random number machine would produce.

Thomas Gilovich, Robert Vallone and Amos Twersky (1985) performed a similar study of basketball shooting. They examined the records of shots from the floor by the Philadelphia 76'ers, foul shots by the Boston Celtics, and a shooting experiment of Cornell University teams. They found that "basketball players and fans alike tend to believe that a player's chance of hitting a shot are greater following a hit than following a miss on the previous shot. However, detailed analyses…provided no evidence for a positive correlation between the outcomes of successive shots."

To put their conclusion differently, knowing whether a shooter has scored or not scored on the previous shot — or in any previous sequence of shots — is of absolutely no use in predicting whether the shooter will or will not score on the next shot. Similarly, knowledge of the past series of at-bats in baseball does not improve a prediction of whether a batter will get a hit this time.

Of course a batter *feels* — and intensely — as if she or he has a better chance of getting a hit at some times than at other times. After a series of successful at-bats, both sandlot players and professionals feel confident that this time will be a hit, too. And after you have hit a bunch of baskets from all over the court, you feel as if you can't miss.

But notice that card players get the same poignant feeling of being "hot" or "cold," too. After a poker player "fills" several straights and flushes in a row, s/he feels s/he will hit the next one too. (Of course there are some players who feel just the opposite, that the "law of averages" is about to catch up with them.)

You will agree, I'm sure, that the cards don't have any memory, and a player's chance of filling a straight or flush remains the same no matter how he or she has done in the last series of hands. Clearly, then, a person can have a strong feeling that something is about to happen even when that feeling has no foundation. This supports the idea that even though a player in sports "feels" that s/he is in a slump or has a hot hand, this does not imply that the feeling has any basis in reality.

Why, when a batter is low in his/her mind because s/he has been making a lot of outs or for personal reasons, does her/ his batting not suffer? And why the opposite? Apparently at any given moment there are many influences operating upon a player's performance in a variety of directions, with none of them clearly dominant. Hence there is no simple convincing explanation why a player gets a hit or an out, a basket or a miss, on any given attempt.

But though science cannot provide an explanation, the sports commentators always are ready to offer their analyses. Listen, for example, to how they tell you that Joe Zilch must have been trying extra hard *just because of* his slump. There is a sportswriter's explanation for *anything* that happens.

Why do we believe the nonsense we hear about "momentum," "comeback," "she's due this time," and so on? The adult of the human species has a powerful propensity to believe that he or she can find a pattern even when there is no pattern to be found. Two decades ago I cooked up series of numbers with a random-number machine that looked as if they were prices on the stock market. Subjects in the experiment were told to buy and sell whichever stocks they chose. Then I gave them "another day's prices," and asked them to buy and sell again. The subjects did all kinds of fancy figuring, using an incredible variety of assumptions — even though there was no way for the figuring to help them. That is, people sought patterns even though there was no reason to believe that there were any patterns to be found.

When I stopped the game before the ten buy-and-sell sessions the participants expected, people asked that the game continue. Then I would tell them that there was no basis for any patterns in the data. "Winning" or "losing" had no meaning. But the subjects demanded to continue anyway. They continued believing that they could find patterns even after I told them that the numbers were randomly looked up and not real stock prices.

The illusions in our thinking about sports have important counterparts in our thinking about such real-world phenomena as the climate, the stock market, and trends in the prices of raw materials such as mercury, copper and wheat. And private and public decisions made on the basis of faulty understanding of these real situations, caused by illusory thinking on the order of belief in slumps and hot hands, are often costly and sometimes disastrous.

An example of the belief that there are patterns when there are none: Systems for finding patterns in the stock market are peddled that have about the same reliability as advice from a racetrack tout — and millions buy them.

One of the scientific strands leading into research on variability was the body of studies that considers the behavior of stock prices as a "random walk." That body of work asserts that a stock broker or chartist who claims to be able to find patterns in past price movements of stocks that will predict future movements should be listened to with about the same credulity as a racetrack tout or an astrologer. A second strand was the work in psychology in the last decade or two which has recognized that people's estimates of uncertain events are systematically biased in a variety of interesting and knowable ways.

The U.S. government has made — and continues to make — blunders costing the public scores of billions of dollars, using slump-type fallacious reasoning about resources and energy. Forecasts are issued and policies are adopted based on the belief that a short-term increase in price constitutes a long-term trend. But the "experts" employed by the government to make such forecasts do no better on average than do private forecasters, and often the system of forecasting that they use is much more misleading than would be a random-number generating machine of the sort used in the baseball slump experiments.

Please look at the data in Figure 14.1 for the height of the Nile River over about half a century. Is it not natural to think that those data show a decline in the height of the river? One can imagine that if our modern communication technology existed then, the Cairo newspapers would have been calling for research to be done on the fall of the Nile, and the television anchors would have been warning the people to change their ways and use less water.

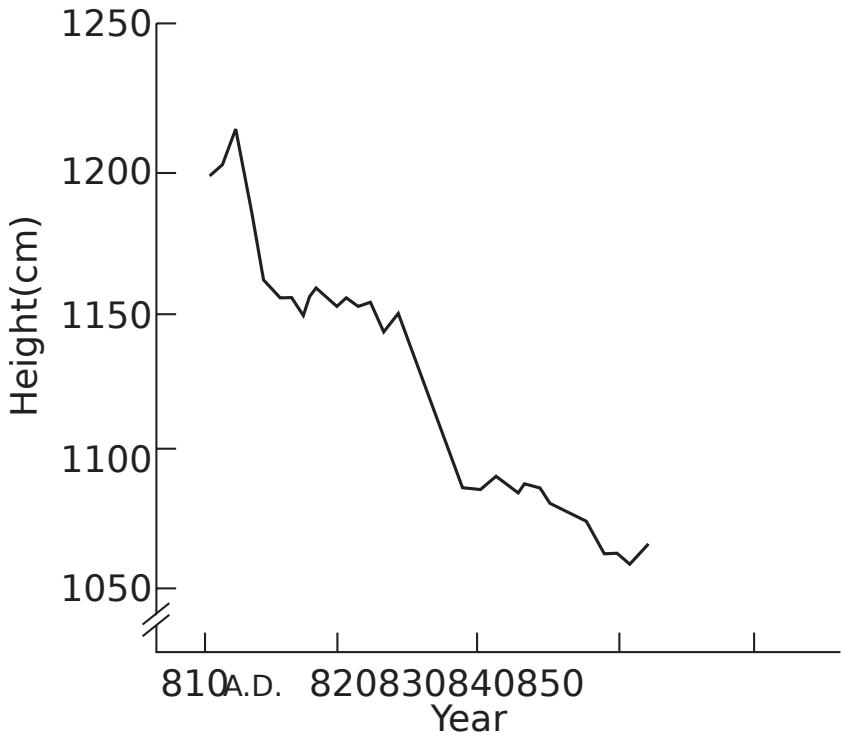


Figure 14.1: Height of the Nile River Over Half of a Century

Let's look at Figure 14.2 which represents the data over an even longer period. What now would you say about the height of the Nile? Clearly the "threat" was non-existent, and only appeared threatening because the time span represented by the data was too short. The point of this display is that looking at too-short a segment of experience frequently leads us into error. And "too short" may be as long as a century.

Another example is the price of mercury, which is representative of all metals. Figure 14.3 shows a forecast made in 1976 by natural-scientist Earl Cook (1976). He combined a then-recent upturn in prices with the notion that there is a finite amount of mercury on the earth's surface, plus the mathematical charm of plotting a second-degree polynomial with the computer. Figure 14.4 and Figure 14.5 show how the forecast was almost immediately falsified, and the price continued its long-run decline.

Lack of sound statistical intuition about variability can lead to manipulation of the public being by unscrupulous persons. Commodity funds sellers use a device of this sort to make their

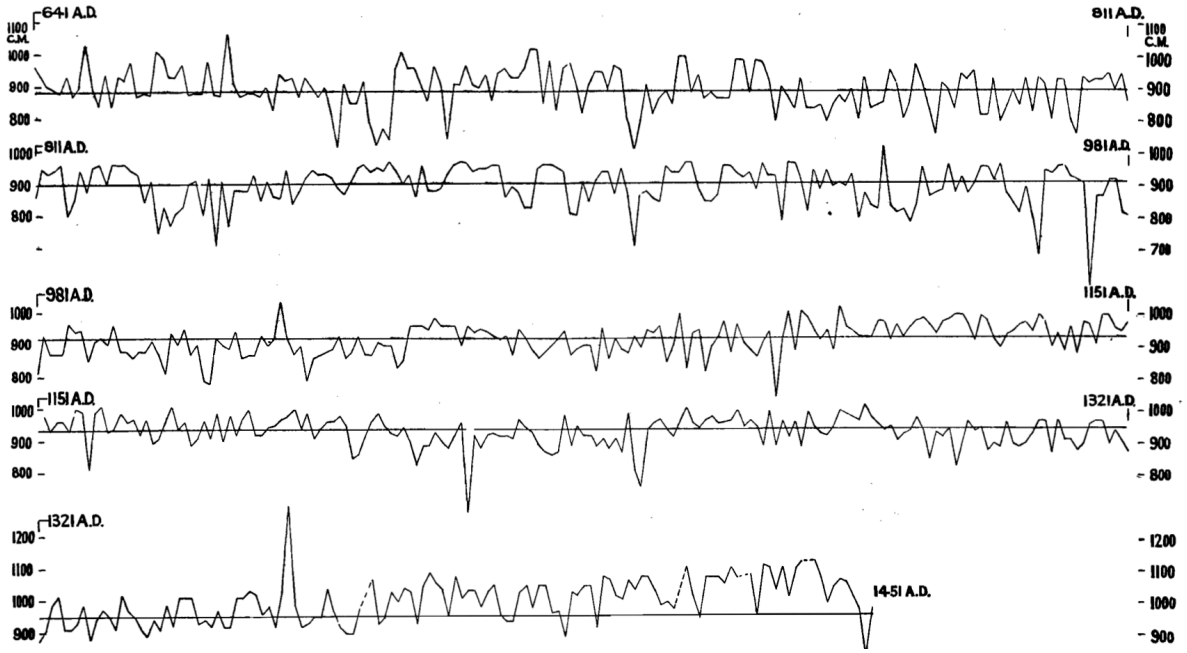


Figure 14.2: Variations in the height of Nile Flood in centimeters. The sloping line indicates the secular raising of the bed of the Nile by deposition of silt. From Brooks (1928)

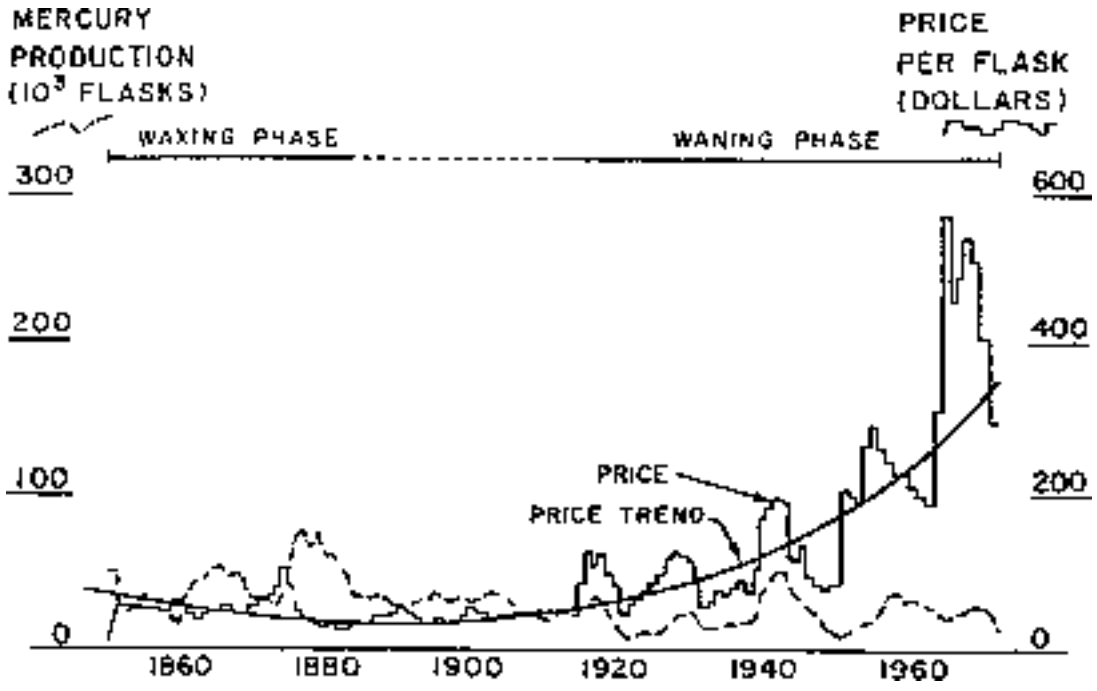


Figure 14.3: The Price of Mercury from Cook (1976)

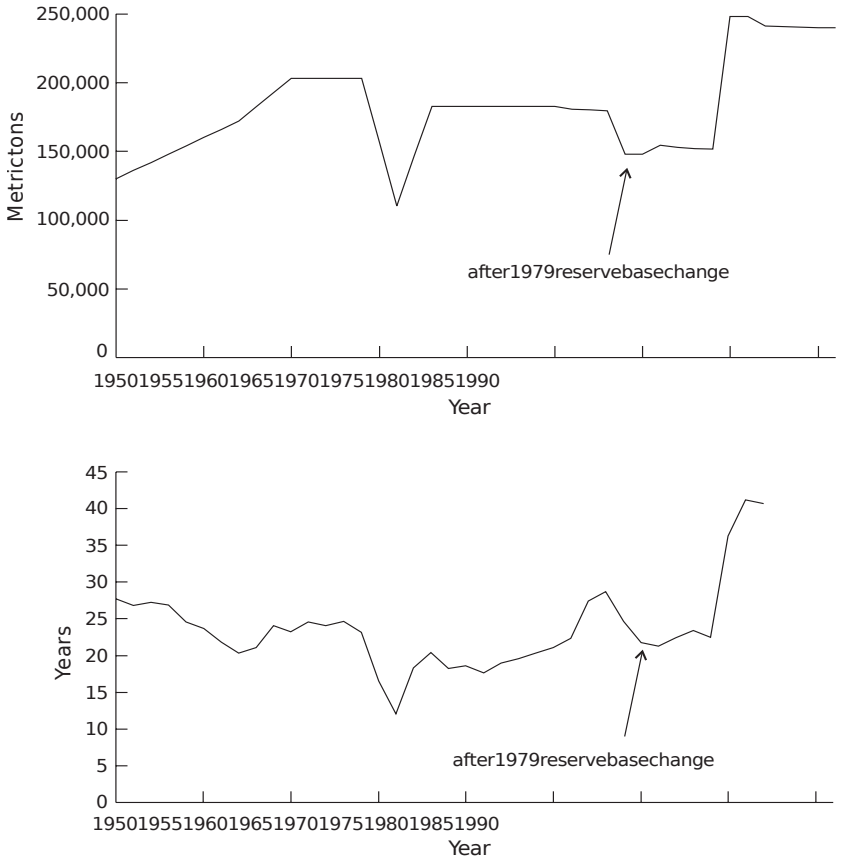


Figure 14.4: Mercury Reserves, 1950-1990

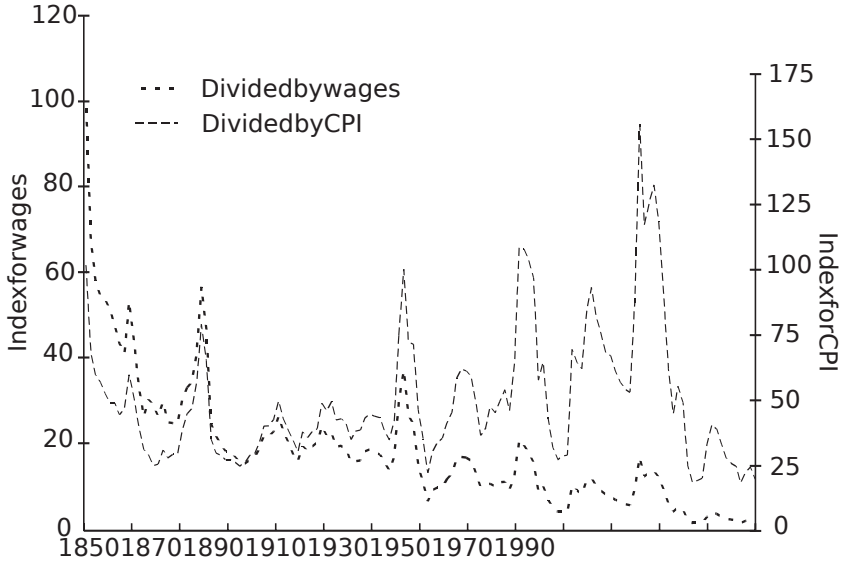


Figure 14.5: Mercury Price Indexes, 1950-1990

results look good (*The Washington Post*, Sep 28, 1987, p. 71). Some individual commodity traders inevitably do well in their private trading, just by chance. A firm then hires one of them, builds a public fund around him, and claims the private record for the fund's own history. But of course the private record has no predictive power, any more than does the record of someone who happened to get ten heads in a row flipping coins.

How can we avoid falling into such traps? It is best to look at the longest possible sweep of history. That is, use the largest possible sample of observations to avoid sampling error. For copper we have data going back to the 18th century B.C. In Babylonia, over a period of 1000 years, the price of iron fell to one fifth of what it was under Hammurabi (almost 4000 years ago), and the price of copper then cost about a thousand times its current price in the U.S., relative to wages. So the inevitable short-run increases in price should be considered in this long-run context to avoid drawing unsound conclusions due to small-sample variability.

Proof that it is sound judgment to rely on the longest possible series is given by the accuracy of predictions one would have made in the past. In the context of copper, mercury, and other raw materials, we can refer to a sample of years in the past, and from those years imagine ourselves forecasting the following year. If you had bet every time that prices would go down in consonance with the long-run trend, you would have been a big winner on average.

14.2 Regression to the mean

> UP, DOWN "The Dodgers demoted last year's NL rookie of the year, OF Todd Hollandsworth (.237, 1 HR, 18 RBI) to AAA Albuquerque..." (Item in *Washington Post* , 6/14/97)

It is a well-known fact that the Rookie of the Year in a sport such as baseball seldom has as outstanding a season in their sophomore year. Why is this so? Let's use the knowledge we have acquired of probability and simulation to explain this phenomenon.

The matter at hand might be thought of as a problem in pure probability — if one simply asks about the chance that a given player (the Rookie of the Year) will repeat. Or it could be considered a problem in statistics, as discussed in coming chapters. Let's consider the matter in the context of baseball.

Imagine 10 mechanical "ball players," each a machine that has three white balls (hits) and 7 black balls. Every time the machine goes to bat, you take a ball out of the machine, look to see if it is a hit or an out, and put it back. For each "ball player" you do this 100 times. One of them is going to do better than the others, and that one becomes the Rookie of the Year. See Table 14.2.

Table 14.2: Rookie Seasons (100 at bats)

# of Hits	Batting Average
32	.320
34	.340
33	.330
30	.300
35	.350
33	.330
30	.300
31	.310
28	.280
25	.250

Would you now expect that the player who happened to be the best among the top ten in the first year to again be the best among the top ten in the next year, also? The sports writers do. But of course this seldom happens. The Rookie of the Year in major-league baseball seldom has as outstanding a season in their sophomore year as in their rookie year. You can expect them to do better than the average of all sophomores, but not necessarily better than all of the rest of the group of talented players who are now sophomores. (Please notice that we are not saying that there is no long-run difference among the top ten rookies. But suppose there is. Table 14.3 shows the season's performance for ten batters of differing performances).

Table 14.3: Simulated season's performance for 10 batters of differing "true" averages

"True"	Rookie
.270	.340
.270	.240
.280	.330
.280	.300
.300	.280
.300	.420
.320	.340
.320	.350
.330	.260
.330	.330

We see from Table 14.3 that we have ten batters whose "true" batting averages range from .270 to .330. Their rookie year performance (400 at bats), simulated on the basis of their "true"average is on the right. Which one is the rookie of the year? It's #6, who hit .420

during the rookie session. Will they do as well next year? Not likely — their "true" average is only .300.

> **Note 35: Notebook: Experiment in sampling variability**
>
> - Download notebook
> - Interact

Try generating some rookie "seasons" yourself with the following commands, ranging the batter's "true" performance by changing the value of `p_hit` (the probability of a hit).

```
import numpy as np

rnd = np.random.default_rng()
```

```
# Simulate a rookie season of 400 at-bats.

# You might try changing the value below and rerunning.
# This is the true (long-run) probability of a hit for this batter.
p_hit = 0.4
print('True average is:', p_hit)
```

```
True average is: 0.4
```

```
at_bats = rnd.choice(['Hit', 'Out'], p=[p_hit, 1 - p_hit], size=400)
simulated_average = np.sum(at_bats == 'Hit') / 400
# Show the result
print('Simulated average is:', simulated_average)
```

```
Simulated average is: 0.4075
```

Simulate a set of 10 or 20 such rookie seasons, and look at the one who did best. How did their rookie season compare to their "true" average?

> **End of notebook: Experiment in sampling variability**
>
> `sampling_variability` starts at Note 35.

The explanation is the presence of *variability*. And lack of recognition of the role of variability is at the heart of much fallacious reasoning. Being alert to the role of variability is crucial.

Or consider the example of having a superb meal at a restaurant — the best meal you have ever eaten. That fantastic meal is almost surely the combination of the restaurant being better than average, plus a lucky night for the chef and the dish you ordered. The next time you return you can expect a meal better than average, because the restaurant is better than average in the long run. But the meal probably will be less good than the superb one you had the first time, because there is no reason to believe that the chef will get so lucky again and that the same sort of variability will happen this time.

These examples illustrate the concept of "regression to the mean" — a confusingly-titled and very subtle effect caused by variability in results among successive samples drawn from the same population. This phenomenon was given its title more than a century ago by Francis Galton, one of the great founders of modern statistics, when at first he thought that the height of the human species was becoming more uniform, after he noticed that the children of the tallest and shortest parents usually are closer to the average of all people than their parents are. But later he discovered his fallacy — that the variability in heights of children of quite short and quite tall parents also causes some people to be even more exceptionally tall or short than their parents. So the spread in heights among humans remains much the same from generation to generation; there is no "regression to the mean." The heart of the matter is that any exceptional observed case in a group is likely to be the result of two forces — a) an *underlying propensity* to differ from the average in one direction or the other, plus b) *some chance sampling variability* that happens (in the observed case) to push even further in the exceptional direction.

A similar phenomenon arises in direct-mail marketing. When a firm tests many small samples of many lists of names and then focuses its mass mailings on the lists that performed best in the tests, the full list "rollouts" usually do not perform as well as the samples did in the initial tests. It took many years before mail-order experts (see especially (Burnett 1988)) finally understood that regression to the mean *inevitably* causes an important part of the dropoff from sample to rollout observed in the set of lists that give the very best results in a multi-list test.

The larger the test samples, the less the dropoff, of course, because larger samples reduce variability in results. But larger samples risk more money. So the test-sample-size decision for the marketer inevitably is a trade-off between accuracy and cost.

And one last amusing example: After I (JLS) lectured to the class on this material, the student who had gotten the best grade on the first mid-term exam came up after class and said: "Does that mean that on the second mid-term I should expect to do well but not the best in the class?" And that's exactly what happened: He had the second-best score in the class on the next midterm.

A related problem arises when one conducts multiple tests, as when testing thousands of drugs for therapeutic value. Some of the drugs may appear to have a therapeutic effect just by chance. We will discuss this problem later when discussing hypothesis testing.

14.3 Summary and conclusion

The heart of statistics is clear thinking. One of the key elements in being a clear thinker is to have a sound gut understanding of statistical processes and variability. This chapter amplifies this point.

A great benefit to using simulations rather than formulas to deal with problems in probability and statistics is that the presence and importance of variability becomes manifest in the course of the simulation work.

15 The Procedures of Monte Carlo Simulation (and Resampling)

Until now, the steps to follow in solving particular problems have been chosen to fit the specific facts of that problem. And so they always must. Now let's generalize what we have done in the previous chapters on probability into a general procedure for such problems, which will in turn become the basis for a detailed procedure for resampling simulation in statistics. The generalized procedure describes what we are doing when we estimate a probability using Monte Carlo simulation problem-solving operations.

15.1 A definition and general procedure for Monte Carlo simulation

This is what we shall mean by the term *Monte Carlo simulation* when discussing problems in probability: *Using the given data-generating mechanism (such as a coin or die) that is a model of the process you wish to understand, produce new samples of simulated data, and examine the results of those samples.* That's it in a nutshell. In some cases, it may also be appropriate to amplify this procedure with additional assumptions.

This definition fits both problems in pure probability as well as problems in statistics, but in the latter case the process is called *resampling.* The reason that the same definition fits is that *at the core of every problem in inferential statistics lies a problem in probability* ; that is, the procedure for handling every statistics problem is the procedure for handling a problem in probability. (There is related discussion of definitions in Chapter 8 and Chapter 20.)

The following series of steps should apply to all problems in probability. I'll first state the procedure straight through without examples, and then show how it applies to individual examples.

- **Step A** *Construct a simulation "universe"* of cards or dice or some other randomizing mechanism whose composition is similar to the universe whose behavior we wish to describe and investigate. The term "universe" refers to the system that is relevant for a single simple event.
- **Step B** *Specify the procedure* that produces a pseudo-sample which simulates the real-life sample in which we are interested. That is, specify the procedural rules by which the sample is drawn from the simulated universe. These rules must correspond to the

behavior of the real universe in which you are interested. To put it another way, the simulation procedure must produce simple experimental events with the same probabilities that the simple events have in the real world.

- **Step C** *Describe any composite events.* If several simple events must be combined into a composite event, and if the composite event was not described in the procedure in step B, describe it now.
- **Step D.** *Calculate the probability* of interest from the tabulation of outcomes of the resampling trials.

Now let us apply the general procedure to some examples to make it more concrete.

Here are four problems to be used as illustrations:

1. **Three percent gizmos** — if on average 3 percent of the gizmos sent out are defective, what is the chance that there will be more than 10 defectives in a shipment of 200?
2. **Three girls, 106 in 206** — what are the chances of getting three or more girls in the first four children, if the probability of a female birth is 106/206?
3. **Less than 20 baskets** — what are the chances of Joe Hothand scoring 20 or fewer baskets in 57 shots if his long-run average is 47 percent?
4. **Same birthday in 25** — what is the probability of two or more people in a group of 25 persons having the same birthday — i. e., the same month and same day of the month?

15.2 Apply step A — construct a simulation universe

As a reminder:

- **Step A** *Construct a simulation "universe"* of cards or dice or some other randomizing mechanism whose composition is similar to the universe whose behavior we wish to describe and investigate. The term "universe" refers to the system that is relevant for a single simple event.

For our example problems:

1. *Three percent gizmos*: A random drawing with replacement from the set of numbers 1 through 100 with 1 through 3 designated as defective, simulates the system that produces 3 defective gizmos among 100.
2. *Three girls, 106 in 206*: You could take two decks of cards, from which you take out both Aces of spades, and replace these with a Joker. You now have 103 cards (206 / 2), of which 53 (106 / 2) are red, counting the Joker as red. You could also use a random drawing from two sets of numbers, one comprising 1 through 106 and the other 107 through 206. Either universe can simulate the system that produces a single male or female birth, when we are estimating the probability of three girls in the first four children. Notice that in this universe the probability of a girl remains the same from trial

event to trial event — that is, the trials are independent — demonstrating a universe from which we sample with replacement.
3. *Less than 20 baskets*: A random drawing with replacement from a bucket containing a hundred balls, 47 red and 53 black, simulates the system that produces 47 percent baskets for Joe Hothand.
4. *Same birthday in 25*: A random drawing with replacement from the numbers 1 through 365 simulates the system that produces a birthday.

This step A includes two operations:

1. Decide which symbols will stand for the elements of the universe you will simulate.
2. Determine whether the sampling will be with or without replacement. (This can be ambiguous in a complex modeling situation.)

Hard thinking is required in order to determine the appropriate "real" universe whose properties interest you.

15.3 Apply step B — specify the procedure

- **Step B** *Specify the procedure* that produces a pseudo-sample which simulates the real-life sample in which we are interested. That is, specify the procedural rules by which the sample is drawn from the simulated universe. These rules must correspond to the behavior of the real universe in which you are interested. To put it another way, the simulation procedure must produce simple experimental events with the same probabilities that the simple events have in the real world.

For example:

1. *Three percent gizmos*: For a single gizmo, you can draw a single number from an infinite universe. Or one can use a finite set with replacement and shuffling.
2. *Three girls, 106 in 206*: In the case of three or more daughters among four children, you could use the deck of 103 cards, from Step A, of which 53 count as red. To simulate one child, you can draw a card and then replace it, noting female for a red card or a Joker. Or if you are using random numbers from the computer, the random numbers automatically simulate replacement. Just as the chances of having a boy or a girl do not change depending on the sex of the preceding child, so we want to ensure through sampling with replacement that the chances do not change each time we choose from the deck of cards.
3. *Less than 20 baskets*: In the case of Joe Hothand's shooting, the procedure is to consider the numbers 1 through 47 as "baskets," and 48 through 100 as "misses," with the same other considerations as the gizmos.

4. *Same birthday in 25.* In the case of the birthday problem, the drawing must be with replacement, because the fact that you have drawn — say — a 10 (10^{th} day in year), should not affect the chances of drawing 10 for a second person in the room.

Recording the outcome of the sampling must be indicated as part of this step, e.g., "record 'yes' if girl or basket, 'no' if a boy or a miss."

15.4 Apply step C — describe any composite events

- **Step C** *Describe any composite events.* If several simple events must be combined into a composite event, and if the composite event was not described in the procedure in step B, describe it now.

For example:

1. *Three percent gizmos*: For the gizmos, draw a sample of 200.
2. *Three girls, 106 in 206*: For the three or more girls among four children, the procedure for each simple event of a single birth was described in step B. Now we must specify repeating the simple event four times, and counting whether the outcome is or is not three girls.
3. *Less than 20 baskets*: In the case of Joe Hothand's shots, we must draw 57 numbers to make up a sample of shots, and examine whether there are 20 or more misses.

Recording the results as "ten or more defectives," "three or more girls" or "two or less girls," and "20 or more misses" or "19 or fewer," is part of this step. This record indicates the results of all the trials and is the basis for a tabulation of the final result.

15.5 Apply step D — calculate the probability

- **Step D.** *Calculate the probability* of interest from the tabulation of outcomes of the resampling trials.

For example: the proportions of "yes" and "no," and "20 or more" and "19 or fewer" estimate the probability we seek in step C.

The above procedure is similar to the procedure followed with the analytic formulaic method except that the latter method constructs notation and manipulates it.

15.6 Summary

This chapter gives a more general description of the specific steps used in prior chapters to solve problems in probability.

16 Ranks, Quantiles and Standard Scores

Imagine we have a set of measures, in some particular units. We may want some way to see quickly how these measures compare to one another, and how they may compare to other measures, in different units.

Ranks are one way of having an implicit comparison between values.[1] Is the value large in terms of the other values (with high rank) — or is it small (low rank)?

We can convert ranks to quantile positions. Quantile positions are values from 0 through 1 that are closer to 1 for high rank values, and closer to 0 for low rank values. The smallest value (and the value with the lowest rank) will have quantile position 0, the largest value (highest rank) will have quantile position 1. Each value in the data has a rank, and a corresponding quantile position. We can also look at the *value* corresponding to each quantile position, and these are the *quantile values*, usually called simply — *quantiles*. You will see what we mean later in the chapter.

Ranks and quantile positions give an idea whether the measure is high or low compared to the other values, but they do not immediately tell us whether the measure is exceptional or unusual. To do that, we may want to ask whether the measure falls outside the typical range of values — that is, how the measure compares to the *distribution* of values. One common way of doing this is to re-express the measures (values) as *standard scores*, where the standard score for a particular value tells you how far the value is from the *center* of the distribution, in terms of the typical *spread* of the distribution. (We will say more about what we mean by "typical" later.) Standard values are particularly useful to allow us to *compare* different types of measures on a standard scale. They translate the units of measurement into *standard* and comparable units.

16.1 Household income and congressional districts

Democratic congresswoman Marcy Kaptur has represented the 9th district of Ohio since 1983. Ohio's 9th district is relatively working class, and the Democratic party has, traditionally, represented people with lower income. However, Kaptur has pointed out that this pattern

[1] To get ranks we have to be able to sort our values in some reasonable way from low to high. This usually makes sense for measured data, where the measurement gives a number in the same unit for each observation, but may not make sense for other data, such as names or labels. For example, it may or may not be meaningful to rank names alphabetically. As usual, this is a matter of judgment.

appears to be changing; more of the high-income congressional districts now lean Democrat, and the Republican party is now more likely to represent lower-income districts. The French economist Thomas Piketty has described this phenomenon across several Western countries. Voters for left parties are now more likely to be highly educated and wealthy. He terms this shift "Brahmin Left Vs Merchant Right" (Piketty 2018). The data below come from a table Kaptur prepared that shows this pattern in the 2023 US congress. The table lists the top 20 districts by the median income of the households in that district, along with their representatives and their party.[2]

Table 16.1: 20 most wealthy 2023 Congressional districts by household income

	Ascending_Rank	District	Median Income	Representative	Party
422	422	MD-3	114804	J. Sarbanes	Democrat
423	423	MA-5	115618	K. Clark	Democrat
424	424	NY-12	116070	J. Nadler	Democrat
425	425	VA-8	116332	D. Beyer	Democrat
426	426	MD-5	117049	S. Hoyer	Democrat
427	427	NJ-11	117198	M. Sherrill	Democrat
428	428	NY-3	119185	G. Santos	Republican
429	429	CA-14	119209	E. Swalwell	Democrat
430	430	NJ-7	119567	T. Kean	Republican
431	431	NY-1	120031	N. LaLota	Republican
432	432	WA-1	120671	S. DelBene	Democrat
433	433	MD-8	120948	J. Raskin	Democrat
434	434	NY-4	121979	A. D'Esposito	Republican
435	435	CA-11	124456	N. Pelosi	Democrat
436	436	CA-15	125855	K. Mullin	Democrat
437	437	CA-10	135150	M. DeSaulnier	Democrat
438	438	VA-11	139003	G. Connolly	Democrat
439	439	VA-10	140815	J. Wexton	Democrat
440	440	CA-16	150720	A. Eshoo	Democrat
441	441	CA-17	157049	R. Khanna	Democrat

You may notice right away that many of the 20 richest districts have Democratic Party representatives.

In fact, if we look at all 441 congressional districts in Kaptur's table, we find a large difference in the average median household income for Democrat and Republican districts; the Democrat districts are, on average, about 14% richer (Table 16.2).

[2]For now, let us define the median value M as the value such that (as close as possible to) half of the households in the district have a lower income than M, and (as close as possible to) half have a higher income. We will give more detail on the median later in the chapter.

Table 16.2: Means for median household income by party

	Mean of median household income
Democrat	$76,933
Republican	$67,474

Next we are going to tip our hand, and show how we got these data. In previous chapters, we had cells like this in which we enter the values we will analyze. These values come from the example we introduced in Section 12.15:

```
# Liquor prices for US states with private market.
priv = np.array([
    4.82, 5.29, 4.89, 4.95, 4.55, 4.90, 5.25, 5.30, 4.29, 4.85, 4.54, 4.75,
    4.85, 4.85, 4.50, 4.75, 4.79, 4.85, 4.79, 4.95, 4.95, 4.75, 5.20, 5.10,
    4.80, 4.29])
```

Now we have 441 values to enter, and it is time to introduce Python's standard tools for loading data.

16.1.1 Comma-separated-values (CSV) format

The data we will load is in a file on disk called `data/congress_2023.csv`. These are data from Kaptur's table in a comma-separated-values (CSV) format file. We refer to this file with its *filename*, that starts with the directory (folder) containing the file — `data/` — followed by the name of the file (`congress_2023.csv`), giving a filename of `data/congress_2023.csv`.

The *CSV format* is a very simple text format for storing table data. Usually, the first line of the CSV file contains the column names of the table, and the rest of the lines contain the row values. As the name suggests, commas (`,`) separate the column names in the first line, and the row values in the following lines. If you opened the `data/congress_2023.csv` file in some editor, such as Notepad on Windows or TextEdit on Mac, you would find that the first few lines looked like this:

```
Ascending_Rank,District,Median_Income,Representative,Party
1,PR-At Large,22237,J. González-Colón,Republican
2,AS-At Large,28352,A. Coleman,Republican
3,MP-At Large,31362,G. Sablan,Democrat
4,KY-5,37910,H. Rogers,Republican
5,MS-2,37933,B. G. Thompson,Democrat
```

In the code that follows, we will *read* the values from the CSV file directly into Python.

> **Note 36: Download links with data files**
>
> Up till now, the Download links for notebook files point to a single file — the *notebook*. The Jupyter notebook file has extension `.ipynb` .
> From now on, some examples involve reading a data file. In order to run the notebook, you need the notebook file (with extension `.ipynb`) as well as the data file. For those cases, the download link points to a `.zip` file containing the notebook file and the data file. To run the example on your computer, download the `.zip` file, extract the contents, and then open the notebook file using Jupyter.

16.1.2 Introducing the Pandas library

> **Note 37: Notebook: Starting with Pandas**
>
> - Download zip with notebook + data file
> - Interact

Here we use the Pandas library to load table data into Python.

Thus far we have used the Numpy library to work with data in arrays. As always with Python, when we want to use a library — like Numpy, or Pandas — we have to `import` it first.

We have used the term *library* here, but Python uses the term *module* to refer to libraries of code and data that you `import`. We will use the terms "library" and "module" to mean the same thing — a Python module.

When using Numpy, we write:

```
# Import the Numpy library (module), name it "np".
import numpy as np
```

Now we will use the Pandas library (module).

We can import Pandas like this:

```
# Import the Pandas library (module)
import pandas
```

As Numpy has a standard abbreviation `np`, that almost everyone writing Python code will recognize and use, so Pandas has the standard abbreviation `pd`:

```
# Import the Pandas library (module), name it "pd".
import pandas as pd
```

Pandas is the standard data science library for Python. It is particularly good at loading data files, and presenting them to us as a useful table-like structure, called a *data frame*.

We start by using Pandas to load our data file:

```
district_income = pd.read_csv('data/congress_2023.csv')
```

We have thus far done many operations that returned Numpy *arrays*. `pd.read_csv` returns a Pandas *data frame*:

```
type(district_income)
```

```
<class 'pandas.core.frame.DataFrame'>
```

A data frame is Pandas' own way of representing a table, with columns and rows. You can think of it as Python's version of a spreadsheet. As strings or Numpy arrays have *methods* (functions attached to the array), so Pandas data frames have methods. These methods do things with the data frame to which they are attached. For example, the `head` method of the data frame gives us (by default) the first five rows in the table:

```
# Show the first five rows in the data frame
district_income.head()
```

```
   Ascending_Rank     District  Median_Income     Representative       Party
0               1  PR-At Large          22237  J. González-Colón  Republican
1               2  AS-At Large          28352         A. Coleman  Republican
2               3  MP-At Large          31362          G. Sablan    Democrat
3               4         KY-5          37910          H. Rogers  Republican
4               5         MS-2          37933     B. G. Thompson    Democrat
```

The data are in income order, from lowest to highest, so the first five districts are those with the lowest household income.

We are particularly interested in the column named `Median_Income`.

You may remember the idea of *indexing*, introduced in Section 6.6. Indexing occurs when we fetch data from within a container, such as a string or an array. We do this by putting square

brackets `[]` after the value we want to index into, and put something inside the brackets to say what we want.

For example, to get the *first* element of the `priv` array above, we use indexing:

```
# Fetch the first element of the priv array with indexing.
# This is the element at position 0.
priv[0]
```

```
np.float64(4.82)
```

As you can index into strings and Numpy arrays, by using square brackets, so you can index into Pandas data frames. Instead of putting the *position* between the square brackets, we can put the *column name*. This fetches the data from that column, returning a new type of value called a Pandas *Series*.

```
# Index into Pandas data frame to get one column of data.
# Notice we use a string between the square brackets, giving the column name.
income_col = district_income['Median_Income']
# The value that comes back is of type Series.  A Series represents the
# data from a single column.
type(income_col)
```

```
<class 'pandas.core.series.Series'>
```

We want to go straight to our familiar Numpy arrays, so we convert the column of data into a Numpy array, using the `np.array` function you have already seen:

```
# Convert column data into a Numpy array.
incomes = np.array(income_col)
# Show the first five values, by indexing with a slice.
incomes[:5]
```

```
array([22237, 28352, 31362, 37910, 37933])
```

> **i** End of notebook: Starting with Pandas
>
> `starting_pandas` starts at Note 37.

16.1.3 Incomes and Ranks

We now have the `incomes` values as an array.

There are 441 values in the whole vector, one of each congressional district:

```
len(incomes)
```

```
441
```

While we are at it, let us also get the values from the `Ascending_Rank` column, with the same procedure. These are ranks from low to high, meaning 1 is the lowest median income, and 441 is the highest median income.

```
lo_to_hi_ranks = np.array(district_income['Ascending_Rank'])
# Show the first five values, by indexing with a slice.
lo_to_hi_ranks[:5]
```

```
array([1, 2, 3, 4, 5])
```

In our case, the DataFrame has the `Ascending_Rank` column with the ranks we need, but if we need the ranks and we don't have them, we can calculate them using the `rankdata` function from the Scipy `stats` package.

16.1.4 Introducing Scipy

Earlier in this chapter we introduced the Pandas module. We used Pandas to load the CSV data into Python.

Now we introduce another fundamental Python library for working with data called Scipy. The name Scipy comes from the compression of SCIentific PYthon, and the library is nearly as broad as the name suggests — it is a huge collection of functions and data that implement a wide range of scientific algorithms. Scipy is an umbrella package, in that it contains sub-packages, each covering a particular field of scientific computing. One of those sub-packages is called `stats`, and, yes, it covers statistics.

We can get the Scipy `stats` sub-package with:

```
import scipy.stats
```

but, as for Numpy and Pandas, we often `import` the package with an abbreviation, such as:

```
# Import the scipy.stats package with the name "sps".
import scipy.stats as sps
```

One of the many functions in `scipy.stats` is the `rankdata` function.

16.1.5 Calculating ranks

As you might expect, `sps.rankdata` accepts an array as an input argument. Let's say that there are `n = len(data)` values in the array that we pass to `sps.rankdata`. The function returns an array, length n, where the elements are the ranks of each corresponding element in the input `data` array. A rank value of 1 corresponds the lowest value in `data` (closest to negative infinity), and a rank of n corresponds to the highest value (closest to positive infinity).

Here's an example `data` array to show how `sps.rankdata` works.

```
# The data.
data = np.array([3, -1, 5, -2])
# Corresponding ranks for the data.
sps.rankdata(data)
```

```
array([3., 2., 4., 1.])
```

We can use `sps.rankdata` to recalculate the ranks for the congressional median household income values.

```
# Recalculate the ranks.
recalculated_ranks = sps.rankdata(incomes)
# Show the first 5 ranks.
recalculated_ranks[:5]
```

```
array([1., 2., 3., 4., 5.])
```

16.2 Comparing two values in the district income data

Let us say that we have taken an interest in two particular members of Congress: the Speaker of the House of Representatives, Republican Kevin McCarthy, and the progressive activist and Democrat Alexandria Ocasio-Cortez. We will refer to both using their initials: KM for Kevin Owen McCarthy and AOC for Alexandra Ocasio-Cortez.

By scrolling through the CSV file, or (in our case) using some simple Pandas code that we won't cover now, we find the rows corresponding to McCarthy (KM) and Ocasio-Cortez (AOC) — Table 16.3.

Table 16.3: Rows for Kevin McCarthy and Alexandra Ocasio-Cortez

Ascending_Rank	District	Median Income	Representative	Party
81	NY-14	56129	A. Ocasio-Cortez	Democrat
295	CA-20	77205	K. McCarthy	Republican

The rows show the *rank* of each congressional district in terms of median household income. The districts are ordered by this rank, so we can get their respective *indices* (positions) in the `incomes` array from their rank. Remember, Python's indices start at 0, whereas the ranks start at 1, so we need to subtract 1 from the rank to get the index

```
# Rank of McCarthy's district in terms of median household income.
km_rank = 295
# Index (position) of McCarthy's value in the "incomes" array.
# Subtract one from rank, because Python starts indices at 0 rather than 1.
km_index = km_rank - 1
```

Now we have the index (position) of KM's value, we can find the household income for his district from the `incomes` array:

```
# Show the median household income from McCarthy's district
# by indexing into the "incomes" array:
km_income = incomes[km_index]
km_income
```

```
np.int64(77205)
```

Here is the corresponding index and `incomes` value for AOC:

```
# Index (position) of AOC's value in the "incomes" array.
aoc_rank = 81
aoc_index = aoc_rank - 1
# Show the median household income from AOC's district
# by indexing into the "incomes" array:
aoc_income = incomes[aoc_index]
aoc_income
```

```
np.int64(56129)
```

Notice that we fetch the same value for median household income from `incomes` as you see in the corresponding rows.

16.3 Comparing values with ranks and quantile positions

We have KM's and AOC's district median household income values, but our next question might be — how unusual are these values?

Of course, it depends what we mean by *unusual*. We might mean, are they greater or smaller than most of the other values?

One way of answering that question is simply looking at the rank of the values. If the rank is lower than $\frac{441}{2} = 220.5$ then this is a district with lower income than most districts. If it is greater than 220.5 then it has higher income than most districts. We see that KM's district, with rank `r get_var('km_rank')`, is wealthier than most, whereas AOC's district (rank `r get_var('aoc_rank')`) is poorer than most.

But we can't interpret the ranks without remembering that there are 441 values, so — for example - a rank of 81 represents a relatively low value, whereas one of 295 is relatively high.

We would like some scale that tells us immediately whether this is a relatively low or a relatively high value, without having to remembering how many values there are.

This is a good use for *quantile positions* (QPs). The QP of a value tells you where the value ranks relative to the other values, on a scale from 0 through 1. A QP of 0 tells you this is the lowest-ranking value, and a QP of 1 tells you this is the highest-ranking value.

We can calculate the QP for each rank. Think of the low-to-high ranks as being a line starting at 1 (the lowest rank — for the lowest median income) and going up to 441 (the highest rank — for the highest median income).

The QP corresponding to any particular rank tells you how far along this line the rank is. Notice that the length of the line is the distance from the first to the last value, so 441 - 1 = 440.

So, if the rank was 1, then the value is at the start of the line. It has got $\frac{0}{440}$ of the way along the line, and the QP is 0. If the rank is 441, the value is at the end of the line, it has got $\frac{440}{440}$ of the way along the line and the QP is 1.

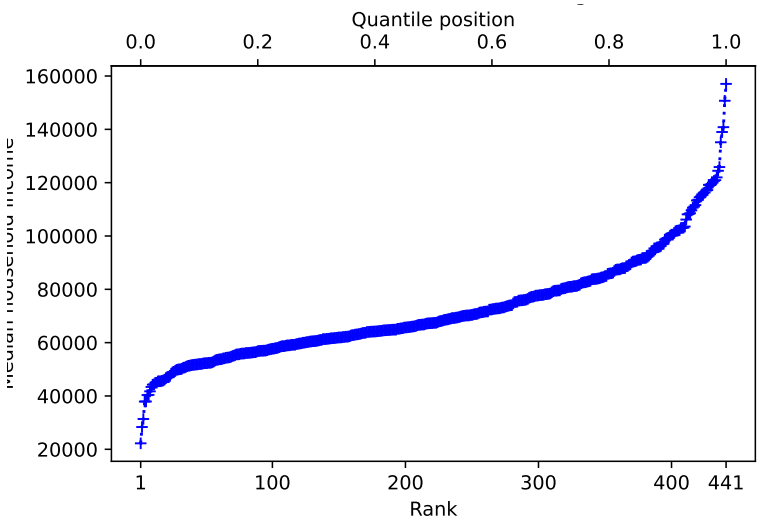
Now consider the rank of 100. It has got $\frac{(100-1)}{440}$ of the way along the line, and the QP position is 0.22.

More generally, we can translate the high-to-low ranks to QPs with:

```
# Length of the line defining quantile positions.
# Start of line is rank 1 (quantile position 0).
# End of line is rank 441 (quantile position 1).
distance = len(lo_to_hi_ranks) - 1  # 440 in our case.
# What proportion along the line does each value get to?
quantile_positions = (lo_to_hi_ranks - 1) / distance
# Show the first five.
quantile_positions[:5]
```

```
array([0.        , 0.00227273, 0.00454545, 0.00681818, 0.00909091])
```

Let's plot the ranks and the QPs together on the x-axis:



The QPs for KM and AOC tell us where their districts' incomes are in the ranks, on a 0 to 1 scale:

```
km_quantile_position = quantile_positions[km_index]
km_quantile_position
```

```
np.float64(0.6681818181818182)
```

```
aoc_quantile_position = quantile_positions[aoc_index]
aoc_quantile_position
```

```
np.float64(0.18181818181818182)
```

If we multiply the QP by 100, we get the *percentile positions* — so the percentile position ranges from 0 through 100.

```
# Percentile positions are just quantile positions * 100
print('KM percentile position:', km_quantile_position * 100)
```

```
KM percentile position: 66.81818181818183
```

```
print('AOC percentile position:', aoc_quantile_position * 100)
```

```
AOC percentile position: 18.181818181818183
```

Now consider one particular QP: 0.5. The 0.5 QP is exactly half-way along the line from rank 1 to rank 441. In our case this corresponds to rank $\frac{441-1}{2} + 1 = 221$.

```
# For rank 221 we need index 220, because Python indices start at 0
print('Middle rank:', lo_to_hi_ranks[220])
```

```
Middle rank: 221
```

```
print('Quantile position:', quantile_positions[220])
```

```
Quantile position: 0.5
```

The *value* corresponding to any particular QP is the *quantile value*, or just the *quantile* for short. For a QP of 0.5, the *quantile* (quantile value) is:

```
# Quantile value for 0.5
print('Quantile value for QP of 0.5:', incomes[220])
```

```
Quantile value for QP of 0.5: 67407
```

In fact we can ask Python for this *value* (quantile) directly, using the `quantile` function:

```
np.quantile(incomes, 0.5)
```

```
np.float64(67407.0)
```

> **i** `quantile` and sorting
>
> In our case, the `incomes` data is already sorted from lowest (at position 0 in the array to highest (at position 440 in the array). The `quantile` function does not need the data to be sorted; it does its own internal sorting to do the calculation.
> For example, we could shuffle `incomes` into a random order, and still get the same values from `quantile`.
>
> ```
> rnd = np.random.default_rng()
> shuffled_incomes = rnd.permuted(incomes)
> # Quantile still gives the same value.
> np.quantile(incomes, 0.5)
> ```
>
> ```
> np.float64(67407.0)
> ```

Above we have the 0.5 *quantile* — the value corresponding to the QP of 0.5.

The 0.5 quantile is an interesting value. By the definition of QP, exactly half of the remaining values (after excluding the 0.5 quantile value) have lower rank, and are therefore less than the 0.5 quantile value. Similarly exactly half of the remaining values are greater than the 0.5 quantile. You may recognize this as the *median* value. This is such a common quantile value that NumPy has a function `np.median` as a shortcut for `np.quantile(data, 0.5)`.

```
np.median(incomes)
```

```
np.float64(67407.0)
```

Another interesting QP is 0.25. We find the QP of 0.25 at rank:

```
qp25_rank = (441 - 1) * 0.25 + 1
qp25_rank
```

```
111.0
```

```
# Therefore, index 110 (Python indices start from 0)
print('Rank corresponding to QP 0.25:', qp25_rank)
```

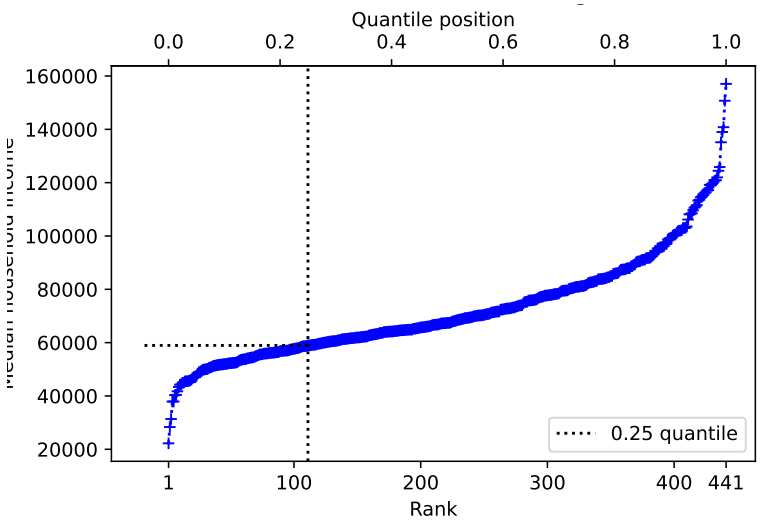
```
Rank corresponding to QP 0.25: 111.0
```

```
print('0.25 quantile value:', incomes[110])
```

```
0.25 quantile value: 58961
```

```
print('0.25 quantile value using np.quantile:',
      np.quantile(incomes, 0.25))
```

```
0.25 quantile value using np.quantile: 58961.0
```



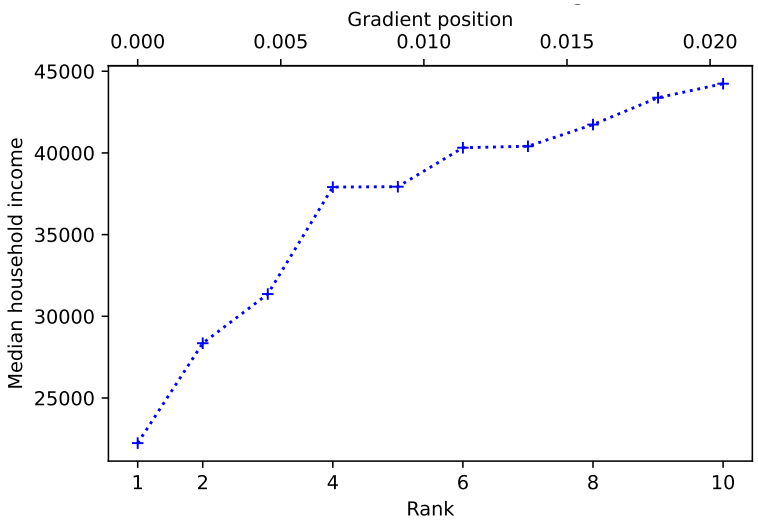
Call the 0.25 quantile value V. V is the number such that 25% of the *remaining* values are less than V, and 75% are greater.

Now let's think about the 0.01 quantile. We don't have an *income value* exactly corresponding to this QP, because there is no rank exactly corresponding to the 0.01 QP.

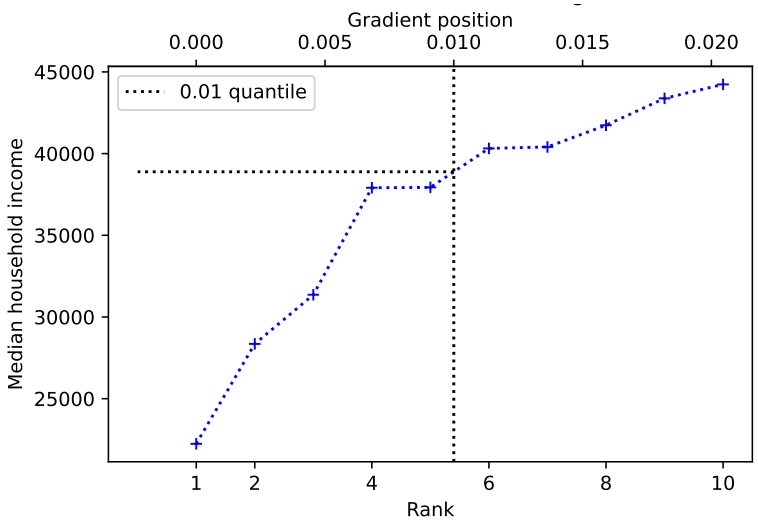
```
rank_for_qp001 = (441 - 1) * 0.01 + 1
rank_for_qp001
```

```
5.4
```

Let's have a look at the first 10 values for rank / QP and incomes:



What then, is the quantile value for QP = 0.01? There are various ways to answer that question (Hyndman and Fan 1996), but one obvious way, and the default for NumPy, is to draw a straight line up from the matching rank — or equivalently, down from the QP — then note where that line crosses the lines joining the values to the left and right of the QP on the graph above, and look across to the y-axis for the corresponding value:




```
np.quantile(incomes, 0.01)
```

```
np.float64(38887.4)
```

This is called the *linear* method — because it uses straight lines joining the points to estimate the quantile value for a QP that does not correspond to a whole-number rank.

Calculating quantiles using the linear method

We gave a graphical explanation of how to calculate the quantile for a QP that does not correspond to whole-number rank in the data. A more formal way of getting the value using the numerical equivalent of the graphical method is linear interpolation. Linear interpolation calculates the quantile value as a *weighted average* of the quantile values for the QPs of the whole number ranks just less than, and just greater than the QP we are interested in. For example, let us return to the QP of 0.01. Here are the QPs, whole-number ranks and corresponding values either side of the QP 0.01:

Table 16.4: Ranks, QPs and corresponding values around QP of 0.01

Rank	Quantile position	Quantile value
5	0.0099	37933
5.4	0.01	**V**
6	0.0113	40319

What value should we should give V in the table? One answer is to take the average of the two values either side of the desired QP — in this case $(37933 + 40319)/2$. We could write this same calculation as $37933 * 0.5 + 40319 * 0.5$ — showing that we are giving equal weight (0.5) to the two values either side.
But giving both values equal weight doesn't seem quite right, because the QP we want is closer to the QP for rank 5 (and corresponding value 37933) than it is to the QP for rank 6 (and corresponding value 40319). We should give more weight to the rank 5 value than the rank 6 value. Specifically the lower value is 0.4 rank units away from the QP rank we want, and the higher is 0.6 rank units away. So we give higher weight for shorter distance, and multiply the rank 5 value by $1 - 0.4 = 0.6$, and the rank 6 value by $1 - 0.6 = 0.4$. Therefore the weighted average is $37933 * 0.6 + 40319 * 0.4 = 38887.4$. This is a mathematical way to get the value we described graphically, of tracking up from the rank of 5.4 to the line drawn between the values for rank 5 and 6, and reading off the y-value at which this track crosses that line.

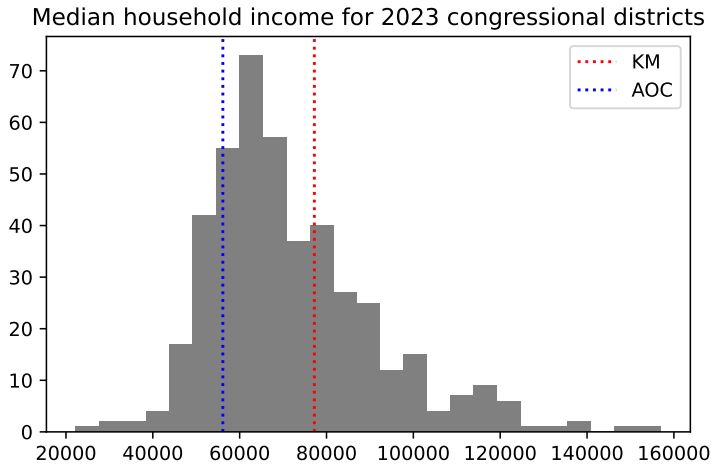
16.4 Unusual values compared to the distribution

Now we return the problem of whether KMs and AOCs districts are unusual in terms of their median household incomes. From what we have so far, we might conclude that AOC's district is fairly poor, and KM's district is relatively wealthy. But — are either of their districts *unusual* in their wealth or poverty?

To answer that question, we have to think about the *distribution* of values. Are either AOC's or KM's district outside the *typical spread* of values for districts?

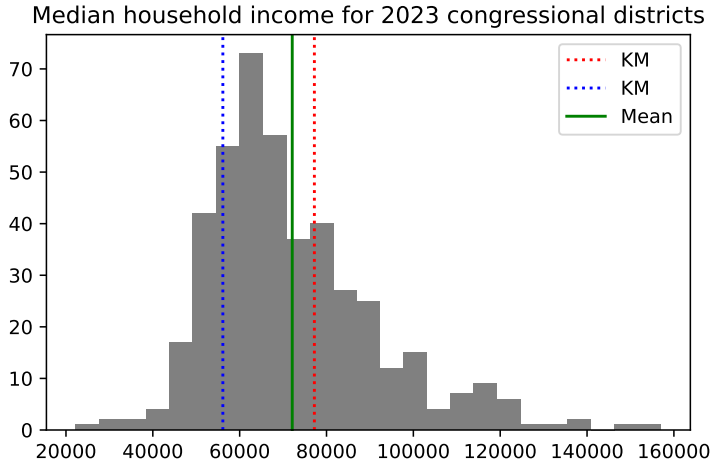
The rest of this section is an attempt to answer what we could mean by *outside* and *typical spread*.

Let us start with a histogram of the district incomes, marking the position of the KM and AOC districts.



What could we mean by "outside" the "typical spread". By outside, we mean somewhere away from the center of the distribution. Let us take the mean of the distribution to be its center, and add that to the plot.

```
mean_income = np.mean(incomes)
```



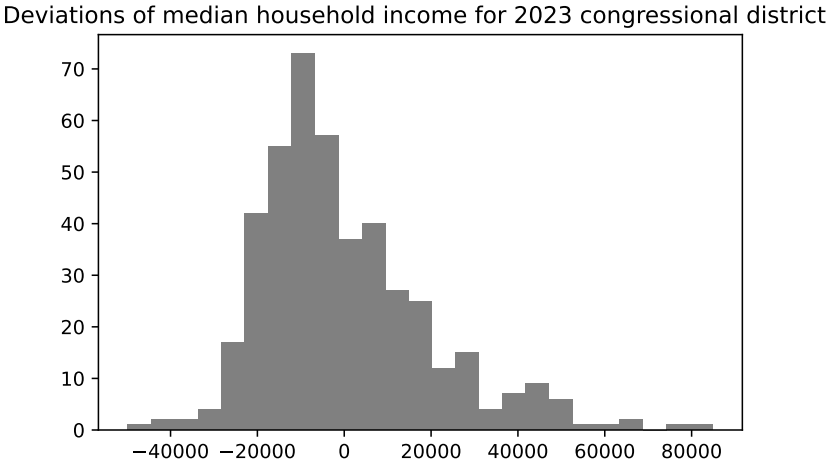
16.5 On deviations

Now let us ask what we could mean by *typical spread*. By *spread* we mean deviation either side of the center.

We can calculate how far away each income is away from the mean, by subtracting the mean from all the income values. Call the result — the *deviations* from the mean, or *deviations* for short.

```
deviations = incomes - np.mean(incomes)
```

The deviation values give, for each district, how far that district's income is from the mean. Values near the mean will have small (positive or negative) values, and values further from the mean will have large (positive and negative) values. Here is a histogram of the *deviation* values.



Notice that the shape of the distribution has not changed — all that changed is the position of the distribution on the x-axis. In fact, the distribution of deviations centers on zero — the deviations have a mean of (as near as the computer can accurately calculate) zero:

```
# Show the mean of the deviations, rounded to 8 decimal places.
np.round(np.mean(deviations), 8)
```

```
np.float64(0.0)
```

16.6 The mean absolute deviation

Now let us consider the deviation value for KM and AOC:

```
print('Deviation for KM:', deviations[km_index])
```

```
Deviation for KM: 5098.036281179142
```

```
print('Deviation for AOC:', deviations[aoc_index])
```

```
Deviation for AOC: -15977.963718820858
```

We have the same problem as before. Yes, we see that KM has a positive deviation, and therefore, that his district is more wealthy than average across the 441 districts. Conversely AOC's district has a negative deviation, and is poorer than average. But we still lack a standard measure of how far away from the mean each district is, in terms of the spread of values in the histogram.

To get such a standard measure, we would like idea of a typical or average deviation. Then we will compare KM's and AOC's deviations to the average deviation, to see if they are unusually far from the mean.

You have just seen above that we cannot use the literal average (mean) of the deviations for this purpose because the positive and negative deviations will exactly cancel out, and the mean deviation will always be as near as the computer can calculate to zero.

To stop the negatives canceling the positives, we can simply knock the minus signs off all the negative deviations.

This is the job of the NumPy `abs` function — where `abs` is short for *absolute*.

> **Note 38: The `np.abs` function**
>
> The `np.abs` function will knock minus signs off negative values, like this:

```
np.abs(-3.1)
```

```
np.float64(3.1)
```

```
np.abs([-1, 0, 1, -2])
```

```
array([1, 0, 1, 2])
```

To get an average of the deviations, *regardless of whether they are positive or negative*, we can take the mean of the absolute deviations, like this:

```
# The Mean Absolute Deviation (MAD)
abs_deviations = np.abs(deviations)
mad = np.mean(abs_deviations)
# Show the result
mad
```

```
np.float64(15101.657570662428)
```

This is the Mean Absolute Deviation (MAD). It is one measure of the typical spread. MAD is the average distance (regardless of positive or negative) of a value from the mean of the values.

We can get an idea of how typical a particular deviation is by dividing the deviation by the MAD value, like this:

```
print('Deviation in MAD units for KM:', deviations[km_index] / mad)
```

```
Deviation in MAD units for KM: 0.33758123949803737
```

```
print('Deviation in MAD units AOC:', deviations[aoc_index] / mad)
```

```
Deviation in MAD units AOC: -1.0580271499375542
```

16.7 The standard deviation

We are interested in the average deviation, but we find that a simple average of the deviations from the mean always gives 0 (perhaps with some tiny calculation error), because the positive and negative deviations cancel exactly.

The MAD calculation solves this problem by knocking the signs off the negative values before we take the mean.

16.7.1 Squares of values and arrays

Another very popular way of solving the same problem is to precede the calculation by squaring all the deviations. Python has an operator for the operation of taking values to the power of another value: it is `**`. For example, to square a single number, we take the number *to the power of* 2, like this:

```
# 10 to the power of 2.
10 ** 2
```

```
100
```

You can also use the power of `**` operator on an array, and, as usual, that works *elementwise*. The array you get back is the result of applying the to-the-power-of operation to all elements of the array in turn:

```
an_arr = np.array([2, 10, 12])
# All the elements in the array, to the power of 2.
an_arr ** 2
```

```
array([  4, 100, 144])
```

16.7.2 Calculating the standard deviation

We can therefore square the deviations we calculated above, like this:

```
squared_deviations = deviations ** 2
# Show the first five values.
squared_deviations[:5]
```

```
array([2.48701328e+09, 1.91449685e+09, 1.66015207e+09, 1.16943233e+09,
       1.16785980e+09])
```

i Exponential format for showing very large and very small numbers

The `squared_deviation` values above appear in *exponential notation* (E-notation). Other terms for E-notation are scientific notation, *scientific form*, or *standard form*. E-notation is a useful way to express very large (far from 0) or very small (close to 0) numbers in a more compact form.
E-notation represents a value as a floating point value m multiplied by 10 to the power of an exponent n:

$$m * 10^n$$

m is a floating point number with one digit before the decimal point — so it can be any value from 1.0 through 9.9999… n is an integer (positive or negative whole number).
For example, the median household income of KM's district is 77205 (dollars). We can express that same number in E-notation as $7.7205 * 10^4$. Python writes this as `7.7205e4`, where the number before the `e` is m and the number after the `e` is the exponent value n. E-notation is another way of writing the number, because $7.7205 * 10^4 = 77205$.

```
7.7205e4 == 77205
```

```
True
```

It is no great advantage to use E-notation in this case; 77205 is probably easier to read and understand than `7.7205e4`. The notation comes into its own where you start to

> lose track of the powers of 10 when you read a number — and that does happen when the number becomes very long without E-notation. For example, $77205^2 = 5960612025$. 5960612025 is long enough that you start having to count the digits to see how large it is. In E-notation, that number is `5.960612025e9`. If you remember that 10^9 is one US billion, then the E-notation tells you at a glance that the value is about 5.9 billion.
> Python makes its own decision whether to print out numbers using E-notation. This only affects the display of the numbers; the underlying values remain the same whether Python chooses to show them in E-notation or not.

The process of squaring the deviations turns all the negative values into positive values.

We can then take the average (mean) of the *squared deviations* to give a measure of the typical *squared* deviation:

```
mean_squared_deviation = np.mean(squared_deviations)
mean_squared_deviation
```

```
np.float64(385971462.1165975)
```

Rather confusingly, the field of statistics uses the term *variance* to refer to the mean squared deviation value. Just to emphasize that naming, let's do the same calculation but using "variance" as the variable name.

```
# Statistics calls the mean squared deviation - the "variance"
variance = np.mean(squared_deviations)
variance
```

```
np.float64(385971462.1165975)
```

It will come as no surprise to find that Numpy has a function to do the whole variance calculation — subtracting the mean, and returning the average squared deviation — `np.var`:

```
# Use np.var to calculate the mean squared deviation directly.
np.var(incomes)
```

```
np.float64(385971462.1165975)
```

The variance is the typical (in the sense of the mean) *squared* deviation. The units for the variance, in our case, would be *squared dollars*. But we are more interested in the typical *deviation*, in our original units – dollars rather than squared dollars.

So we take the square root of the mean squared deviation (the square root of the variance), to get the *standard deviation*. It is the *standard* deviation in that it is a measure of *typical* deviation, in the specific sense of the square root of the mean squared deviations.

```
# The standard deviation is the square root of the mean squared deviation.
# (and therefore, the square root of the variance).
standard_deviation = np.sqrt(mean_squared_deviation)
standard_deviation
```

```
np.float64(19646.156420954136)
```

Again, Numpy has a function to do this calculation directly: `np.std`:

```
# Use np.std to calculate the square root of the mean squared deviation
# directly.
np.std(incomes)
```

```
np.float64(19646.156420954136)
```

```
# Of course, np.std(incomes) is the same as:
np.sqrt(np.var(incomes))
```

```
np.float64(19646.156420954136)
```

The *standard deviation* (the square root of the mean squared deviation) is a popular alternative to the Mean Absolute Deviation, as a measure of typical spread.

Figure 16.1 shows another histogram of the `income` values, marking the mean, the mean plus or minus one standard deviation, and the mean plus or minus two standard deviations. You can see that the mean plus or minus one standard deviation includes a fairly large proportion of the data. The mean plus or minus two standard deviation includes a much larger proportion.

Now let us return to the question of how unusual our two congressional districts are in terms of the distribution. First we calculate the number of standard deviations of each district from the mean:

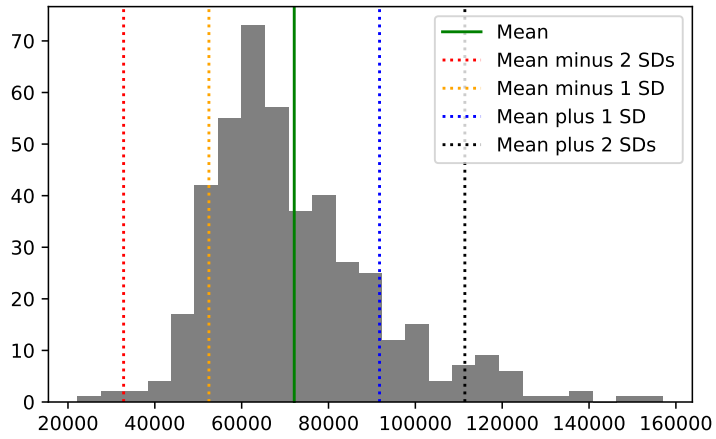


Figure 16.1: Income histogram plus or minus 1 and 2 standard deviations

```
km_std_devs = deviations[km_index] / standard_deviation
print('Deviation in standard deviation units for KM:',
      np.round(km_std_devs, 2))
```

```
Deviation in standard deviation units for KM: 0.26
```

```
aoc_std_devs = deviations[aoc_index] / standard_deviation
print('Deviation in standard deviation units for AOC:',
      np.round(aoc_std_devs, 2))
```

```
Deviation in standard deviation units for AOC: -0.81
```

The values for each district are a re-expression of the income values in terms of the distribution. They give the distance from the mean (positive or negative) in units of standard deviation.

16.8 Standard scores

We will often find uses for the procedure we have just applied, where we take the original values (here, `incomes`) and:

- Subtract the mean to convert to deviations, then
- Divide by the standard deviation

Let's apply that procedure to all the `incomes` values.

First we calculate the standard deviation:

```
deviations = incomes - np.mean(incomes)
income_std = np.sqrt(np.mean(deviations ** 2))
```

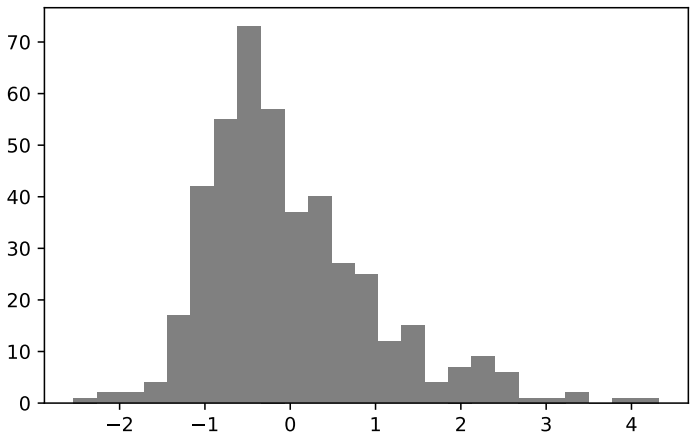
Then we calculate standard scores:

```
deviations_in_stds = deviations / income_std
# Show the first five values.
deviations_in_stds[:5]
```

```
array([-2.53840816, -2.22715135, -2.07394072, -1.74064397, -1.73947326])
```

This procedure converts the original data (here `incomes`) to deviations from the mean in terms of the standard deviation. The resulting values are called standard scores or *z-scores*. One name for this procedure is "z-scoring".

If you plot a histogram of the standard scores, you will see they have a mean of (actually exactly) 0, and a standard deviation of (actually exactly) 1.



With all this information — what should we conclude about the two districts in question? KM's district is 0.26 standard deviations *above* the mean, but that's not enough to conclude that it is unusual. We see from the histogram that a large proportion of the districts are at least this distance from the mean. We can calculate that proportion directly.

```
# Distances (negative or positive) from the mean.
abs_std_devs = np.abs(deviations_in_stds)
# Number where distance greater than KM distance.
n_gt_km = np.sum(abs_std_devs > km_std_devs)
prop_gt_km = n_gt_km / len(deviations_in_stds)
print("Proportion of districts further from mean than KM:",
      np.round(prop_gt_km, 2))
```

```
Proportion of districts further from mean than KM: 0.82
```

A full 82% of districts are further from the mean than is KM's district. KM's district is richer than average, but not unusual. The benefit of the standard deviation distance is that we can see this directly from the value, without doing the calculation of proportions, because the standard deviation is a measure of typical spread, and KM's district is well-within this measure.

AOC's district is -0.81 standard deviations from the mean. This is a little more unusual than KM's score.

```
# Number where distance greater than AOC distance.
# Make AOC's distance positive to correspond to distance from the mean.
n_gt_aoc = np.sum(abs_std_devs > np.abs(aoc_std_devs))
prop_gt_aoc = n_gt_aoc / len(deviations_in_stds)
print("Proportion of districts further from mean than AOC:",
      np.round(prop_gt_aoc, 2))
```

```
Proportion of districts further from mean than AOC: 0.35
```

Only 35% of districts are further from the mean than AOC's district, but this is still a reasonable proportion. We see from the standard score that AOC is within one standard deviation. AOC's district is poorer than average, but not to a remarkable degree.

16.9 Standard scores to compare values on different scales

Why are standard scores so useful? They allow us to compare values on very different scales.

Consider the values in Table 16.5. Each row of the table corresponds to a team competing in the English Premier League (EPL) for the 2021-2022 season. For those of you with absolutely no interest in sports, the EPL is the league of the top 20 teams in English football, or *soccer* to our North American friends. The `points` column of the table gives the total number of

points at the end of the 2021 season (from 38 games). The team gets 3 points for a win, and 1 point for a draw, so the maximum possible points from 38 games are $3 * 38 = 114$. The `wages` column gives the estimated total wage bill in thousands of British Pounds (£1000).

Table 16.5: 2021 points and wage bills (£1000s) for EPL teams

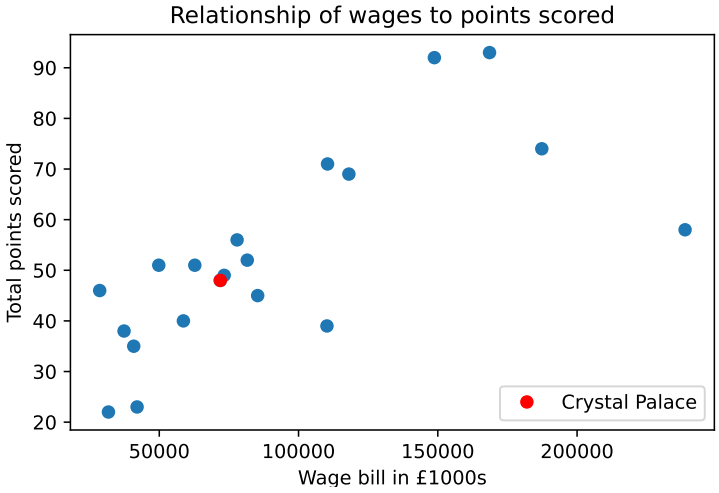
team	points	wages
Manchester City	93	168572
Liverpool	92	148772
Chelsea	74	187340
Tottenham Hotspur	71	110416
Arsenal	69	118074
Manchester United	58	238780
West Ham United	56	77936
Leicester City	52	81590
Brighton and Hove Albion	51	49820
Wolverhampton Wanderers	51	62756
Newcastle United	49	73308
Crystal Palace	48	71910
Brentford	46	28606
Aston Villa	45	85330
Southampton	40	58657
Everton	39	110202
Leeds United	38	37354
Burnley	35	40830
Watford	23	42030
Norwich City	22	31750

Let's say we own Crystal Palace Football Club. Crystal Palace was a bit below average in the league in terms of points. Now we are thinking about whether we should invest in higher-paid players for the coming season, to improve our points score, and therefore, league position.

One thing we might like to know is whether there is an association between the wage bill and the points scored.

To look at that, we can do a scatter plot. This is a plot with — say — wages on the x-axis, and points on the y-axis. For each team we have a pair of values — their wage bill and their points scored. For each team, we put a marker on the scatter plot at the coordinates given by the wage value (on the x-axis) and the points value (on the y-axis).

Here is that plot for our EPL data in Table 16.5, with the Crystal Palace marker picked out in red.



It looks like there is a rough association of wages and points; teams that spend more in wages tend to have more points.

At the moment, the points and wages are in very different units. Points are on a possible scale of 0 (lose every game) to 38 * 3 = 114 (win every game). Wages are in thousands of pounds. Maybe we are not interested in the values in these units, but in how unusual the values are, in terms of wages, and in terms of points.

This is a good application of standard scores. Standard scores convert the original values to values on a standard scale, where 0 corresponds to an average value, 1 to a value one standard deviation above the mean, and -1 to a value one standard deviation below the mean. If we follow the standard score process for both points and wages, the values will be in the same standard units.

To do this calculation, we need the values from the table. We follow the same recipe as before, in loading the data with Pandas, and converting to arrays.

```
import numpy as np
import pandas as pd

points_wages = pd.read_csv('data/premier_league.csv')
points = np.array(points_wages['points'])
wages = np.array(points_wages['wages'])
```

As you recall, the standard deviation is the square root of the mean squared deviation. In code:

```
# The standard deviation is the square root of the
# mean squared deviation.
wage_deviations = wages - np.mean(wages)
wage_std = np.sqrt(np.mean(wage_deviations ** 2))
wage_std
```

```
np.float64(55523.946071289814)
```

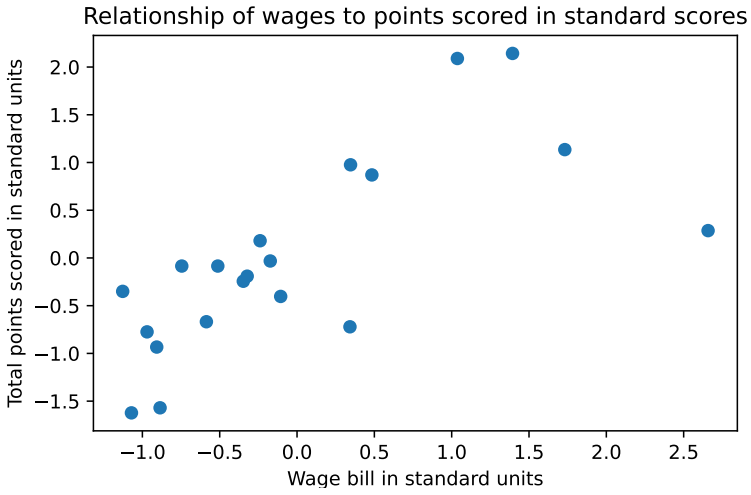
Now we can apply the standard score procedure to wages. We divide the deviations by the standard deviation.

```
standard_wages = (wages - np.mean(wages)) / wage_std
```

We apply the same procedure to the points:

```
point_deviations = points - np.mean(points)
point_std = np.sqrt(np.mean(point_deviations ** 2))
standard_points = point_deviations / point_std
```

Now, when we plot the standard score version of the points against the standard score version of the wages, we see that they are in comparable units, each with a mean of 0, and a spread (a standard deviation) of 1.



Let us go back to our concerns as the owners of Crystal Palace. Counting down from the top in the table above, we see that Crystal Palace is the 12th row. Therefore, we can get the Crystal Palace wage value with:

```
# In Python the 12th value is at position (index) 11
cp_index = 11
cp_wages = wages[cp_index]
cp_wages
```

```
np.int64(71910)
```

We can get our wage bill in standard units in the same way:

```
cp_standard_wages = standard_wages[cp_index]
cp_standard_wages
```

```
np.float64(-0.3474473873890471)
```

Our wage bill is a below average, but its still within striking distance of the mean.

We know that we are comparing ourselves against the other teams, so perhaps we want to increase our wage bill by one standard deviation, to push us above the mean, and somewhat away from the center of the pack. If we add one standard deviation to our wage bill, that increases the standard score of our wages by 1.

But — if we increase our wages by one standard deviation — how much can we expect that to increase our points — in standard units.

That is question about the strength of the association between two measures — here wages and points — and we will cover that topic in much more detail in Chapter 29. But, racing ahead — here is the answer to the question we have just posed — the amount we expect to gain in points, in standard units, if we increase our wages by one standard deviation (and therefore, 1 in standard units).

For reasons we won't justify now, we calculate the r value of association between wages and points, like this:

```
standards_multiplied = standard_wages * standard_points
r = np.mean(standards_multiplied)
r
```

```
np.float64(0.7080086644844557)
```

The r value is the answer to our question. For every one unit increase in standard scores in wages, we expect an increase of r (0.708) standard score units in points.

16.10 Conclusion

When we look at a set of values, we often ask questions about whether individual values are unusual or surprising. One way of doing that is to look at where the values are in the sorted order — for example, using the raw rank of values, or the proportion of values below this value — the quantile position or percentile position of a value. Another measure of interest is where a value is in comparison to the spread of all values either side of the mean. We use the term "deviations" to refer to the original values after we have subtracted the mean of the values. We can measure spread either side of the mean with metrics such as the mean of the absolute deviations (MAD) and the square root of the mean squared deviations (the standard deviation). One common use of the deviations and the standard deviation is to transform values into *standard scores*. These are the deviations divided by the standard deviation, and they transform values to have a standard mean (zero) and spread (standard deviation of 1). Standard scores make it easier to compare sets of values with very different ranges and means.

17 The Basic Ideas in Statistical Inference

Probabilistic statistical inference is a crucial part of the process of informing ourselves about the world around us. Statistics and statistical inference help us understand our world and make sound decisions about how to act.

More specifically, statistical inference is the process of drawing conclusions about populations or other collections of objects about which we have only partial knowledge from samples. Technically, inference may be defined as the selection of a probabilistic model to resemble the process you wish to investigate, investigation of that model's behavior, and interpretation of the results. Fuller understanding of the nature of statistical inference comes with practice in handling a variety of problems.

Until the 18^{th} century, humanity's extensive knowledge of nature and technology was not based on formal probabilistic statistical inference. But now that we have already dealt with many of the big questions that are easy to answer without probabilistic statistics, and now that we live in a more ramified world than in earlier centuries, the methods of inferential statistics become ever more important.

Furthermore, statistical inference will surely become ever more important in the future as we voyage into realms that are increasingly difficult to comprehend. The development of an accurate chronometer to tell time on sea voyages became a crucial need when Europeans sought to travel to the New World. Similarly, probability and statistical inference become crucial as we voyage out into space and down into the depths of the ocean and the earth, as well as probe into the secrets of the microcosm and of the human mind and soul.

Where probabilistic statistical inference is employed, the inferential procedures may well not be the crucial element. For example, the wording of the questions asked in a public-opinion poll may be more critical than the statistical-inferential procedures used to discern the reliability of the poll results. Yet we dare not disregard the role of the statistical procedures.

17.1 Knowledge without probabilistic statistical inference

Let us distinguish two kinds of knowledge with which inference at large (that is, not just probabilistic statistical inference) is mainly concerned: a) one or more *absolute* measurements on one or more dimensions of a collection of one or more items — for example, your income, or the mean income of the people in your country; and b) *comparative* measurements and

evaluations of two or more collections of items (especially whether they are equal or unequal)—for example, the mean income in Brazil compared to the mean income in Argentina. Types (a) and (b) both include asking whether there has been a *change* between one observation and another.

What is the conceptual basis for gathering these types of knowledge about the world? I believe that our rock bottom conceptual tool is the assumption of what we may call *sameness*, or *continuity*, or *constancy*, or *repetition*, or *equality*, or *persistence* ; "constancy" and "continuity" will be the terms used most frequently here, and I shall use them interchangeably.

Continuity is a non-statistical concept. It is a best guess about the next point beyond the known observations, without any idea of the accuracy of the estimate. It is like testing the ground ahead when walking in a marsh. It is local rather than global. We'll talk a bit later about why continuity seems to be present in much of the world that we encounter.

The other great concept in statistical inference, and perhaps in all inference taken together, is *representative (usually random) sampling*, to be discussed in Chapter 18. Representative sampling — which depends upon the assumption of sameness (homogeneity) throughout the universe to be investigated — is quite different than continuity; representative sampling assumes that there is *no greater chance* of a connection between any two elements that might be drawn into the sample than between any other two elements; the order of drawing is immaterial. In contrast, continuity assumes that *there is a greater chance* of connection between two contiguous elements than between either one of the elements and any of the many other elements that are not contiguous to either. Indeed, the process of randomizing is a device for doing away with continuity and autocorrelation within some bounded closed system — the sample "frame." It is an attempt to map (describe) the entire area ahead using the device of the systematic survey. Random representative sampling enables us to make probabilistic inferences about a population based on the evidence of a sample.

To return now to the concept of sameness: Examples of the principle are that we assume: a) our house will be in the same place tomorrow as today; b) a hammer will break an egg every time you hit the latter with the former (or even the former with the latter); c) if you observe that the first fifteen persons you see walking out of a door at the airport are male, the sixteenth probably will be male also; d) paths in the village stay much the same through a person's life; e) religious ritual changes little through the decades; f) your best guess about tomorrow's temperature or stock price is that will be the same as today's. This principle of constancy is related to David Hume's concept of *constant conjunction*. When my children were young, I would point to a tree on our lawn and ask: "Do you think that tree will be there tomorrow?" And when they would answer "Yes," I'd ask, "Why doesn't the tree fall?" That's a tough question to answer.

There are two reasonable bases for predicting that the tree will be standing tomorrow. First and most compelling for most of us is that almost all trees continue standing from day to day, and this particular one has never fallen; hence, what has been in the past is likely to continue. This assessment requires no scientific knowledge of trees, yet it is a very functional way to

approach most questions concerning the trees — such as whether to hang a clothesline from it, or whether to worry that it will fall on the house tonight. That is, we can predict the outcome in this case with very high likelihood of being correct even though we do not utilize anything that would be called either science or statistical inference. (But what do you reply when your child says: "Why should I wear a seat belt? I've never been in an accident"?)

A second possible basis for prediction that the tree will be standing is scientific analysis of the tree's roots — how the tree's weight is distributed, its sickness or health, and so on. Let's put aside this sort of scientific-engineering analysis for now.

The first basis for predicting that the tree will be standing tomorrow — sameness — is the most important heuristic device in all of knowledge-gathering. It is often a weak heuristic; certainly the prediction about the tree would be better grounded (!) after a skilled forester examines the tree. But persistence alone might be a better heuristic in a particular case than an engineering-scientific analysis alone.

This heuristic appears more obvious if the child — or the adult — were to respond to the question about the tree with another question: Why should I expect it to *fall* ? In the absence of some reason to expect change, it is quite reasonable to expect no change. And the child's new question does not duck the central question we have asked about the tree, any more than one ducks a probability estimate by estimating the complementary probability (that is, unity minus the probability sought); indeed, this is a very sound strategy in many situations.

Constancy can refer to location, time, relationship to another variable, or yet another dimension. Constancy may also be cyclical. Some cyclical changes can be charted or mapped with relative certainty — for example the life-cycles of persons, plants, and animals; the diurnal cycle of dark and light; and the yearly cycle of seasons. The courses of some diseases can also be charted. Hence these kinds of knowledge have long been well known.

Consider driving along a road. One can predict that the price of the next gasoline station will be within a few cents of the gasoline station that you just passed. But as you drive further and further, the dispersion increases as you cross state lines and taxes differ. This illustrates continuity.

The attention to constancy can focus on a single event, such as leaves of similar shape appearing on the same plant. Or attention can focus on single sequences of "production," as in the process by which a seed produces a tree. For example, let's say you see two puppies — one that looks like a low-slung dachshund, and the other a huge mastiff. You also see two grown male dogs, also apparently dachshund and mastiff. If asked about the parentage of the small ones, you are likely — using the principle of sameness — to point — quickly and with surety — to the adult dogs of the same breed. (Here it is important to notice that this answer implicitly assumes that the fathers of the puppies are among these dogs. But the fathers might be somewhere else entirely; it is in these ways that the principle of sameness can lead you astray.)

When applying the concept of sameness, the object of interest may be collections of data, as in Semmelweiss's (1983, 64) data on the consistent differences in rates of maternal deaths from

childbed fever in two clinics with different conditions (see Table 17.1), or the similarities in sex ratios from year to year in Graunt's (1759, 304) data on christenings in London (Table 17.2), or the stark effect in John Snow's (Winslow 1980, 276) data on the numbers of cholera cases associated with two London water suppliers (Table 17.3), or Kanehiro Takaki's (Kornberg 1991, 9) discovery of the reduction in beriberi among Japanese sailors as a result of a change in diet (Table 17.4). These data seem so overwhelmingly clear cut that our naive statistical sense makes the relationships seem deterministic, and the conclusions seems straightforward. (But the same statistical sense frequently misleads us when considering sports and stock market data.)

Table 17.1: Deaths of Mothers from childbed fever in two clinics

	First clinic			Second clinic		
	Births	Deaths	Rate	Births	Deaths	Rate
1841	3,036	237	7.7	2,442	86	3.5
1842	3,287	518	15.8	2,659	202	7.5
1843	3,060	274	8.9	2,739	164	5.9
1844	3,157	260	8.2	2,956	68	2.3
1845	3,492	241	6.8	3,241	66	2.03
1845	4,010	459	11.4	3,754	105	2.7
Total	20,042	1,989		17,791	691	
Average			9.92			3.38

Table 17.2: Ratio of number of male to number of female christenings in London

Period	Male / Female ratio
1629-1636	1.072
1637-1640	1.073
1641-1648	1.063
1649-1656	1.095
1657-1660	1.069

Table 17.3: Rates of death from cholera for three water suppliers

Water supplier	Cholera deaths per 10,000 houses
Southwark and Vauxhall	71
Lambeth	5
Rest of London	9

Table 17.4: Takaki's Japanese Naval Records of Deaths from Beriberi

Year	Diet	Total Navy Personnel	Deaths from Beriberi
1880	Rice diet	4,956	1,725
1881	Rice diet	4,641	1,165
1882	Rice diet	4,769	1,929
1883	Rice Diet	5,346	1,236
1884	Change to new diet	5,638	718
1885	New diet	6,918	41
1886	New diet	8,475	3
1887	New diet	9,106	0
1888	New diet	9,184	0

Constancy and sameness can be seen in macro structures; consider, for example, the constant location of your house. Constancy can also be seen in micro aggregations — for example, the raindrops and rain that account for the predictably fluctuating height of the Nile, or the ratio of boys to girls born in London, cases in which we can *average* to see the "statistical" sameness. The total sum of the raindrops produces the level of a reservoir or a river from year to year, and the sum of the behaviors of collections of persons causes the birth rates in the various years.

Statistical inference is only needed when a person thinks that s/he *might* have found a pattern but the pattern is not completely obvious to all. Probabilistic inference works to test — either to confirm or discount — the belief in the pattern's existence. We will see such cases in the following chapter.

People have always been forced to think about and act in situations that have not been constant — that is, situations where the amount of variability in the phenomenon makes it impossible to draw clear cut, sensible conclusions. For example, the appearance of game animals in given places and at given times has always been uncertain to hunters, and therefore it has always been difficult to know which target to hunt in which place at what time. And of course variability of the weather has always made it a very uncertain element. The behavior of one's enemies and friends has always been uncertain, too, though uncertain in a manner different from the behavior of wild animals; there often is a gaming element in interactions with other humans. But in earlier times, data and techniques did not exist to enable us to bring statistical inference to bear.

17.2 The treatment of uncertainty

The purpose of *statistical* inference is to help us peer through the veil of variability when it obscures the main thrust of the data, so as to improve the decisions we make. Statistical

inference (or in most cases, simply probabilistic estimation) can help:

- a gambler deciding on the appropriate odds in a betting game when there seems to be little or no difference between two or more outcomes;
- an astronomer deciding upon one or another value as the central estimate for the location of a star when there is considerable variation in the observations s/he has made of the star;
- a basketball coach pondering whether to remove from the game her best shooter who has heretofore done poorly tonight;
- an oil-drilling firm debating whether to follow up a test-well drilling with a full-bore drilling when the probability of success is not overwhelming but the payoff to a gusher could be large.

Returning to the tree near the Simon house: Let's change the facts. Assume now that one major part of the tree is mostly dead, and we expect a big winter storm tonight. What is the danger that the tree will fall on the house? Should we spend $1500 to have the mostly-dead third of it cut down? We know that last year a good many trees fell on houses in the neighborhood during such a storm.

We can gather some data on the proportion of old trees this size that fell on houses — about 5 in 100, so far as we can tell. Now it is no longer an open-and-shut case about whether the tree will be standing tomorrow, and we are using statistical inference to help us with our thinking. We proceed to find a set of trees *that we consider similar to this one*, and study the variation in the outcomes of such trees. So far we have estimated that the *average* for this group of trees — the mean (proportion) that fell in the last big storm — is 5 percent. Averages are much more "stable" — that is, more similar to each other — than are individual cases.

Notice how we use the crucial concept of sameness: We assume that our tree is like the others we observed, or at least that it is not systematically different from most of them and it is more-or-less average.

How would our thinking be different if our data were that one tree in 10 had fallen instead of 5 in 100? This is a question in statistical inference.

How about if we investigate further and find that 4 of 40 *elms* fell, but only one of 60 *oaks*, and ours is an oak tree. Should we consider that oaks and elms have different chances of falling? Proceeding a bit further, we can think of the question as: Should we or should we not consider oaks and elms as different? This is the type of statistical inference called "hypothesis testing": We apply statistical procedures to help us decide whether to treat the two classes of trees as the same or different. If we should consider them the same, our worries about the tree falling are greater than if we consider them different with respect to the chance of damage.[1]

[1]It is because hypothesis testing focuses on this most basic of inferential processes — deciding "same" or "different" — that I believe it to be a more basic technique than estimating confidence intervals, which focus on the accuracy of estimates.

Notice that statistical inference was not necessary for accurate prediction when I asked the kids about the likelihood of a live tree falling on a day when there would be no storm. So it is with most situations we encounter. But when the assumption of constancy becomes shaky for one reason or another, as with the sick tree falling in a storm, we need a more refined form of thinking. We collect data on a large number of instances, inquire into whether the instances in which we are interested (our tree and the chance of it falling) are representative — that is, whether it resembles what we would get if we drew a sample randomly — and we then investigate the behavior of this large class of instances to see what light it throws on the instances(s) in which we are interested.

The procedure in this case — which we shall discuss in greater detail later on — is to ask: If oaks and elms are *not* different, how likely is it that only one of 60 oaks would fall whereas 4 of 40 elms would fall? Again, notice the assumption that our tree is "representative" of the other trees about which we have information — that it is not systematically different from most of them, but rather that it is more-or-less average. Our tree certainly was not chosen randomly from the set of trees we are considering. But for purposes of our analysis, we proceed *as if* it had been chosen randomly — because we deem it "representative."

This is the first of two roles that the concept of randomness plays in statistical thinking. Here is an example of the second use of the concept of randomness: We conduct an experiment — plant elm and oak trees at *randomly-selected* locations on a plot of land, and then try to blow them down with a wind-making machine. (The random selection of planting spots is important because some locations on a plot of ground have different growing characteristics than do others.) Some purists object that *only* this sort of experimental sampling is a valid subject of statistical inference; it can never be appropriate, they say, to simply *assume* on the basis of other knowledge that the tree is representative. I regard that purist view as a helpful discipline on our thinking. But accepting its conclusion — that one should not apply statistical inference except to randomly-drawn or randomly-constituted samples — would take from us a tool that has proven useful in a variety of activities.

As discussed earlier in this chapter, the data in some (probably most) scientific situations are so overwhelming that one can proceed without probabilistic inference. Historical examples include those shown above of Semmelweiss and puerperal fever, and John Snow and cholera.[2] But where there was lack of overwhelming evidence, the causation of many diseases long remained unclear for lack of statistical procedures. This led to superstitious beliefs and counter-productive behavior, such as quarantines against plague often were. Some effective practices also arose despite the lack of sound theory, however — the waxed costumes of doctors, and the burning of mattresses, despite the wrong theory about the causation of plague; see (Cipolla 1981).

[2] A peculiar perverseness associated with the knowledge of statistical inference is that very strong findings, which require little or no formal inference to demonstrate and which are so powerful that they can be shown with a simple graph or table, are very hard to publish in social science literature because they do not meet the tests of "rigor," and "elegance." Editors view them as detracting from the "technical level" of their journals. A good many of the greatest discoveries of the past would nowadays fall in this category of being difficult or impossible to publish.

So far I have spoken only of *predictability* and not of other elements of statistical knowledge such as *understanding* and *control*. This is simply because statistical *correlation* is the bed rock of most scientific understanding, and predictability. Later we will expand the discussion beyond predictability; it holds no sacred place here.

17.3 Where statistical inference becomes crucial

There was little role for statistical inference until about three centuries ago because there existed very few scientific data. When scientific data began to appear, the need emerged for statistical inference to improve the interpretation of the data. As we saw, statistical inference is not needed when the evidence is overwhelming. A thousand cholera cases at one well and zero at another obviously does not require a statistical test. Neither would 999 cases to one, or even 700 cases to 300, because our inbred and learned statistical senses can detect that the two situations are different. But probabilistic inference is needed when the number of cases is relatively small or where for other reasons the data are somewhat ambiguous.

For example, when working with the 17th century data on births and deaths, John Graunt — great statistician though he was — drew wrong conclusions about some matters because he lacked modern knowledge of statistical inference. For example, he found that in the rural parish of Romsey "there were born 15 Females for 16 Males, whereas in London there were 13 for 14, which shows, that London is somewhat more apt to produce Males, then the country" (p. 71). He suggests that the "curious" inquire into the causes of this phenomenon, apparently not recognizing — and at that time he had no way to test — that the difference might be due solely to chance. He also notices (p. 94) that the variations in deaths among years in Romsey were greater than in London, and he attempted to explain this apparent fact (which is just a statistical artifact) rather than understanding that this is almost inevitable because Romsey is so much smaller than London. Because we have available to us the modern understanding of variability, we can now reach sound conclusions on these matters.[3]

Summary statistics — such as the simple mean — are devices for reducing a large mass of data (inevitably confusing unless they are absolutely clear cut) to something one can manage to understand. And probabilistic inference is a device for determining whether patterns should be considered as facts or artifacts.

Here is another example that illustrates the state of early quantitative research in medicine:

> Exploring the effect of a common medicinal substance, Bőcker examined the effect of sasparilla on the nitrogenous and other constituents of the urine. An individual receiving a controlled diet was given a decoction of sasparilla for a period of twelve days, and the volume of urine passed daily was carefully measured. For a further twelve days that same individual, on the same diet, was given only distilled water,

[3] I (JLS) benefited from the discussion of this matter by Hald (1990, 93ff).

> and the daily quantity of urine was again determined. The first series of researches gave the following figures (in cubic centimeters): 1,467, 1,744, 1,665, 1,220, 1,161, 1,369, 1,675, 2,199, 887, 1,634, 943, and 2,093 (mean = 1,499); the second series: 1,263, 1,740, 1,538, 1,526, 1,387, 1,422, 1,754, 1,320, 1,809, 2,139, 1,574, and 1,114 (mean = 1,549). Much uncertainty surrounded the exactitude of these measurements, but this played little role in the ensuing discussion. The fundamental issue was not the quality of the experimental data but how inferences were drawn from those data (Coleman 1987, 207).

The experimenter Böcker had no reliable way of judging whether the data for the two groups were or were not meaningfully different, and therefore he arrived at the unsound conclusion that there was indeed a difference. (Gustav Radicke used this example as the basis for early work on statistical significance (Støvring 1999).)

Another example: Joseph Lister convinced the scientific world of the germ theory of infection, and the possibility of preventing death with a disinfectant, with these data: Prior to the use of antiseptics — 16 post-operative deaths in 35 amputations; subsequent to the use of antiseptics — 6 deaths in 40 amputations (Winslow 1980, 303). But how sure could one be that a difference of that size might not occur just by chance? No one then could say, nor did anyone inquire, apparently.

Here's another example of great scientists falling into error because of a too-primitive approach to data (Feller 1968, 1:69–70): Charles Darwin wanted to compare two sets of measured data, each containing 16 observations. At Darwin's request, Francis Galton compared the two sets of data by ranking each, and then comparing them pairwise. The a's were ahead 13 times. Without knowledge of the actual probabilities Galton concluded that the treatment was effective. But, assuming perfect randomness, the probability that the a's beat [the others] 13 times or more equals 3/16. This means that in three out of sixteen cases a perfectly ineffectual treatment would appear as good or better than the treatment classified as effective by Galton.

That is, Galton and Darwin reached an unsound conclusion. As Feller (1968, 1:70) says, "This shows that a quantitative analysis may be a valuable supplement to our rather shaky intuition".

Looking ahead, the key tool in situations like Graunt's and Böcker's and Lister's is creating *ceteris paribus* — making "everything else the same" — with random selection in experiments, or at least with statistical controls in non-experimental situations.

17.4 Conclusions

In all knowledge-seeking and decision-making, our aim is to peer into the unknown and reduce our uncertainty a bit. The two main concepts that we use — the two great concepts in all of

scientific knowledge-seeking, and perhaps in all practical thinking and decision-making — are a) continuity (or non-randomness) and the extent to which it applies in given situation, and b) random sampling, and the extent to which we can assume that our observations are indeed chosen by a random process.

18 Introduction to Statistical Inference

The usual goal of a statistical inference is a decision about which of two or more hypotheses a person will thereafter choose to believe and act upon. The strategy of such inference is to consider the behavior of a given universe in terms of the samples it is likely to produce, and if the observed sample is *not* a likely outcome of sampling from that universe, we then proceed as if the sample did not in fact come from that universe. (The previous sentence is a restatement in somewhat different form of the core of statistical analysis.)

18.1 Statistical inference and random sampling

Continuity and sameness is the fundamental concept in inference in general, as discussed in Chapter 17. Random sampling is the second great concept in inference, and it distinguishes probabilistic statistical inference from non-statistical inference as well as from non-probabilistic inference based on statistical data.

Let's begin the discussion with a simple though unrealistic situation. Your friend Arista a) looks into a cardboard carton, b) reaches in, c) pulls out her hand, and d) shows you a green ball. What might you reasonably infer?

You might at least be fairly sure that the green ball came from the carton, though you recognize that Arista might have had it concealed in her hand when she reached into the carton. But there is not much more you might reasonably conclude at this point except that there was at least one green ball in the carton to start with. There could be no more balls; there could be many green balls and no others; there could be a thousand red balls and just one green ball; and there could be one green ball, a hundred balls of different colors, and two pounds of mud — given that she looked in first, it is not improbable that she picked out the only green ball among other material of different sorts.

There is not much you could say with confidence about the probability of yourself reaching into the same carton with your eyes closed and pulling out a single green ball. To use other language (which some philosophers might say is not appropriate here as the situation is too specific), there is little basis for induction about the contents of the box. Nor is the situation very different if your friend reaches in three times in a row and hands you a green ball each time.

So far we have put our question rather vaguely. Let us frame a more precise inquiry: What do we predict about the next item(s) we might draw from the carton? If we assume — based on who-knows-what information or notions — that another ball will emerge, we could simply use the principle of sameness and (until we see a ball of another color) predict that the next ball will be green, whether one or three or 100 balls is (are) drawn.

But now what about if Arista pulls out nine green balls and one red ball? The principle of sameness cannot be applied as simply as before. Based on the last previous ball, the next one will be red. But taking into account *all* the balls we have seen, the next will "probably" be green. We have no solid basis on which to go further. There cannot be any "solution" to the "problem" of reaching a general conclusion on the basis of these specific pieces of evidence.

Now consider what you might conclude if you were told that a single green ball had been drawn *with a random sampling procedure* from a box containing nothing but balls. Knowledge that the sample was drawn randomly from a given universe is grounds for belief that one knows much more than if a sample were not drawn randomly. First, you would be sure — if you had reasonable basis to believe that the sampling really was random, which is not easy to guarantee — that the ball came from the box. Second, you would guess that the proportion of green balls is not very small, because if there are only a few green balls and many other-colored balls, it would be unusual — that is, the event would have a low probability — to draw a green ball. Not impossible, but unlikely. And *we can compute the probability of drawing a green ball* — or any other combination of colors — *for different assumed compositions within the box.* So the knowledge that the sampling process is random greatly increases our ability — or our confidence in our ability — to infer the contents of the box.

Let us note well the strategy of the previous paragraph: *Ask about the probability that one or more various possible contents of the box (the "universe") will produce the observed sample*, on the assumption that the sample was drawn randomly. *This is the central strategy of all statistical inference*, though I do not find it so stated elsewhere. We shall come back to this idea shortly.

There are several kinds of questions one might ask about the contents of the box. One general category includes questions about our best guesses of the box's contents — that is, questions of *estimation.* Another category includes questions about our *surety* of that description, and our surety that the contents are similar or different from the contents of other boxes; the consideration of surety follows after estimates are made. The estimation questions can be subtle and unexpected (Savage 1972, chap. 15), but do not cause major controversy about the foundations of statistics. So we can quickly move on to questions about the extent of surety in our estimations.

Consider your reaction if the sampling produces 10 green balls in a row, or 9 out of 10. If you had no other information (a very important assumption that we will leave aside for now), your best guess would be that the box contains all green balls, or a proportion of 9 of 10, in the two cases respectively. This estimation process seems natural enough.

You would be surprised if someone told you that instead of the box containing the proportion in the sample, it contained just *half* green balls. How surprised? Intuitively, the extent of your surprise would depend on the probability that a half-green "universe" would produce 10 or 9 green balls out of 10. This surprise is a key element in the logic of the hypothesis-testing branch of statistical inference.

We learn more about the likely contents of the box by asking about the probability that various *specific* populations of balls within the box would produce the particular sample that we received. That is, we can ask how likely a collection of 25 percent green balls is to produce (say) 9 of 10 green ones, and how likely collections of 50 percent, 75 percent, 90 percent (and any other collections of interest) are to produce the observed sample. That is, we ask about the *consistency* between any particular hypothesized collection within the box and the sample we observe. And it is reasonable to believe that those universes which have greater consistency with the observed sample — that is, those universes that are more likely to produce the observed sample — are more likely to be in the box than other universes. This (to repeat, as I shall repeat many times) is the basic strategy of statistical investigation. If we observe 9 of 10 green balls, we then determine that universes with (say) 9/10 and 10/10 green balls are more consistent with the observed evidence than are universes of 0/10 and 1/10 green balls. So by this process of considering specific universes that the box *might* contain, we make possible more specific inferences about the box's *probable* contents based on the sample evidence than we could without this process.

Please notice the role of the assessment of probabilities here: By one technical means or another (either simulation or formulas), we assess the probabilities that a particular universe will produce the observed sample, and other samples as well.

It is of the highest importance to recognize that without additional knowledge (or assumption) one cannot make any statements about the probability of the sample having come from *any particular universe*, on the basis of the sample evidence. (Better read that last sentence again.) We can only speak about the probability that a particular universe *will produce* the observed sample, a very different matter. This issue will arise again very sharply in the context of confidence intervals.

Let us generalize the steps in statistical inference:

1. Frame the original question as: **What is the chance of getting the observed sample x from population X?** That is, what is probability of (If x then X)?
2. Proceed to this question: **What kinds of samples does X produce, with which probability?** That is, what is the probability of this particular x coming from X? That is, what is p(x|X)?
3. Actually **investigate the behavior of X with respect to x** and other samples. One can do this in two ways:

1. Use the formulaic calculus of probability, perhaps resorting to Monte Carlo methods if an appropriate formula does not exist. Or,
2. Use resampling (in the larger sense), the domain of which equals (all Monte Carlo experimentation) minus (the use of Monte Carlo methods for approximations, investigation of complex functions in statistics and other theoretical mathematics, and uses elsewhere in science). Resampling in its more restricted sense includes the bootstrap, permutation tests, and other non-parametric methods.

4. **Interpretation of the probabilities** that result from step 3 in terms of

    i) acceptance or rejection of hypotheses, ii) surety of conclusions, or iii) inputs to decision theory.

Here is a short definition of statistical inference:

> The selection of a probabilistic model that might resemble the process you wish to investigate, the investigation of that model's behavior, and the interpretation of the results.

We will get even more specific about the procedure when we discuss the canonical procedures for hypothesis testing and for the finding of confidence intervals in the chapters on those subjects.

The discussion so far has been in the spirit of what is known as *hypothesis testing*. The result of a hypothesis test is a decision about whether or not one believes that the sample is likely to have been drawn randomly from the "benchmark universe" X. The logic is that if the probability of such a sample coming from that universe is low, we will then choose to believe the alternative — to wit, that the sample came from the universe that resembles the sample.

The underlying idea is that if an event would be very surprising if it really happened — as it would be very surprising if the dog had really eaten the homework (see Chapter 21) — we are inclined not to believe in that possibility. (This logic will be explored further in later chapters on hypothesis testing.)

We have so far assumed that our only relevant knowledge is the sample. And though we almost never lack *some* additional information, this can be a sensible way to proceed when we wish to *suppress* any other information or speculation. This suppression is controversial; those known as Bayesians or subjectivists want us to take into account all the information we have. But even they would not dispute suppressing information in certain cases — such as a teacher who does not want to know students' IQ scores because s/he might want avoid the possibility of unconsciously being affected by that score, or an employer who wants not to know the potential employee's ethnic or racial background even though the hiring process might be more "successful" on some metric, or a sports coach who refuses to pick the starting team each year until the players have competed for the positions.

Now consider a variant on the green-ball situation discussed above. Assume now that you are told that samples of balls are alternately drawn from one of two *specified* universes — two buckets of balls, one with 50 percent green balls and the other with 80 percent green balls. Now you are shown a sample of nine green and one red balls drawn from one of those buckets. On the basis of your sample you can then say how probable it is that the sample came *from one or the other universe*. You proceed by computing the probabilities (often called the *likelihoods* in this situation) that each of those two universes would individually produce the observed samples — probabilities that you could arrive at with resampling, with Pascal's Triangle, or with a table of binomial probabilities, or with the Normal approximation and the Z distribution, or with yet other devices. Those probabilities are .01 and .27, and the ratio of the two (0.1/.27) is a bit less than .04. That is, fair betting odds are about 1 to 27.

Let us consider a genetics problem on this model. Plant A produces 3/4 black seeds and 1/4 reds; plant B produces all reds. You get a red seed. Which plant would you guess produced it? You surely would guess plant B. Now, how about 9 reds and a black, from Plants A and C, the latter producing 50 percent reds on average?

To put the question more precisely: What betting odds would you give that the one red seed came from plant B? Let us reason this way: If you do this again and again, 4 of 5 of the red seeds you see will come from plant B. Therefore, reasonable (or "fair") odds are 4 to 1, because this is in accord with the ratios with which red seeds are produced by the two plants — 4/4 to 1/4.

How about the sample of 9 reds and a black, and plants A and C? It would make sense that the appropriate odds would be derived from the probabilities of the two plants producing that particular sample, probabilities which we computed above.

Now let us move to a bit more complex problem: Consider two buckets — bucket G with 2 red and 1 black balls, and bucket H with 100 red and 100 black balls. Someone flips a coin to decide which bucket will be drawn from, reaches into that bucket, and chooses two balls without replacing the first one before drawing the second. Both are red. What are the odds that the sample came from bucket G? Clearly, the answer should derive from the probabilities that the two buckets would produce the observed sample.

(Now just for fun, how about if the first ball drawn is thrown back after examining? What now are the appropriate odds?)

Let's restate the central issue. One can state the probability that a particular plant which produces *on average* 1 red and 3 black seeds will produce one red seed, or 5 reds among a sample of 10. But without further assumptions — such as the assumption above that the possibilities are limited to two specific universes — one cannot say how likely a given red seed is to have come from a given plant, even if we know that that plant produces only reds. (For example, it may have come from *other* plants producing only red seeds.)

When we limit the possibilities to two universes (or to a larger set of specified universes) we are able to put a probability on one hypothesis or another. But to repeat, in many or most

cases, one cannot reasonably assume it is *only* one or the other. And then we cannot state any odds that the sample came from a particular universe. This is a very difficult point to grasp, experience shows, but a crucial one. (It is the sort of subtle issue that makes statistics so difficult.)

The additional assumptions necessary to talk about the probability that the red seed came from a given plant are the stuff of statistical inference. And they must be combined with such "objective" probabilistic assessments as the probability that a 1-red-3-black plant will produce one red, or 5 reds among 10 seeds.

Now let us move one step further. Instead of stating as a fact under our control that there is a .5 chance of the sample being drawn from each of the two buckets in the problem above, let us assume that we do not *know* the probability of each bucket being picked, but instead we *estimate* a probability of .5 for each bucket, based on a variety of other information that all is uncertain. But though the facts are now different, the most reasonable estimate of the odds that the observed sample was drawn from one or the other bucket will not be different than before — because in both situations we were working with a "prior probability" of .5.

Now let us go a step further by allowing the universes from which the sample may have come to have different assumed probabilities as well as different compositions. That is, we now consider prior probabilities other than .5.

How do we decide which universe(s) to investigate for the probability of producing the observed sample, and of producing samples that are even less likely, in the sense of being more surprising? That judgment depends upon the purpose of your analysis, upon your point of view of how statistics ought to be done, and upon some other factors.

It should be noted that the logic described so far applies in exactly the same fashion whether we do our work estimating probabilities with the resampling method or with conventional methods. We can figure the probability of nine or more green chips from a universe of (say) p = .7 with either approach.

So far we have discussed the *comparison* of various hypotheses and possible universes. We must also consider where the consideration of the *reliability* of estimates comes in. This leads to the concept of *confidence limits*, which will be discussed in Chapter 26 and Chapter 27.

18.2 Samples Whose Observations May Have More Than Two Values

So far we have discussed samples and universes that we can characterize as proportions of elements which can have only one of two characteristics — green or other, in this case, which is equivalent to "1" or "0." This expositional choice has been solely for clarity. All the ideas discussed above pertain just as well to samples whose observations may have more than two values, and which may be either discrete or continuous.

18.3 Summary and conclusions

A statistical question asks about the probabilities of a sample having arisen from various source universes in light of the evidence of a sample. In every case, the statistical answer comes from considering the behavior of particular specified universes in relation to the sample evidence and to the behavior of other possible universes. That is, a statistical problem is an exercise in postulating universes of interest and interpreting the probabilistic distributions of results of those universes. The preceding sentence is the key operational idea in statistical inference.

Different sorts of realistic contexts call for different ways of framing the inquiry. For each of the established models there are types of problems which fit that model better than other models, and other types of problems for which the model is quite inappropriate.

Fundamental wisdom in statistics, as in all other contexts, is to employ a large tool kit rather than just applying only a hammer, screwdriver, or wrench no matter what the problem is at hand. (Philosopher Abraham Kaplan once stated Kaplan's Law of scientific method: Give a small boy a hammer and there is nothing that he will encounter that does not require pounding.) Studying the text of a poem statistically to infer whether Shakespeare or Bacon was the more likely author is quite different than inferring whether bioengineer Smythe can produce an increase in the proportion of calves, and both are different from decisions about whether to remove a basketball player from the game or to produce a new product.

Some key points: 1) In statistical inference as in all sound thinking, one's *purpose is central*. All judgments should be made relative to that purpose, and in light of costs and benefits. (This is the spirit of the Neyman-Pearson approach). 2) One cannot avoid making judgments; the process of statistical inference cannot ever be perfectly routinized or objectified. Even in science, fitting a model to experience requires judgment. 3) The best ways to infer are different in different situations — economics, psychology, history, business, medicine, engineering, physics, and so on. 4) Different tools must be used when the situations call for them — sequential vs. fixed sampling, Neyman-Pearson vs. Fisher, and so on. 5) In statistical inference it is wise not to argue about the proper conclusion when the data and procedures are ambiguous. Instead, whenever possible, one should go back and get more data, hence lessening the importance of the efficiency of statistical tests. In some cases one cannot easily get more data, or even conduct an experiment, as in biostatistics with cancer patients. And with respect to the past one cannot produce more historical data. But one can gather more and different kinds of data, e.g. the history of research on smoking and lung cancer.

19 Point Estimation

One of the great questions in statistical inference is: How big is it? This can mean — How long? How deep? How much time? At what angle?

This question about size may pertain to a single object, of which there are many measurements; an example is the location of a star in the heavens. Or the question may pertain to a varied set of elements and their measurements; examples include the effect of treatment with a given drug, and the incomes of the people of the United States in 1994.

From where the observer stands, having only the evidence of a sample in hand, it often is impossible to determine whether the data represent multiple observations of a single object, or single (or multiple) observations of multiple objects. For example, from crude measurements of weight you could not know whether one person is being weighed repeatedly, or several people have been weighed once. Hence all the following discussion of point estimation is the same for both of these situations.

The word "big" in the first sentence above is purposely vague, because there are many possible kinds of estimates that one might wish to make concerning a given object or collection. For a single object like a star, one surely will wish to make a best guess about its location. But about the effects of a drug treatment, or the incomes of a nation, there are many questions that one may wish to answer. The *average* effect or income is a frequent and important object of our interest. But one may also wish to know about the amount of *dispersion* in the distribution of treatment effects, or of incomes, or the symmetry of the distribution. And there are still other questions one may wish to answer.

Even if we focus on the average, the issue often is less clear cut than we may think at first. If we are to choose a single number to characterize the population (universe) from which a given set of data has been drawn, what should that representative number be for the case at hand? The answer must depend on the purpose with which we ask the question, of course. There are several main possibilities such as the mean, the median, and the mode.

Even if we confine our attention to the mean as our measure of the central tendency of a distribution, there are various ways of estimating it, each of them having a different rationale. The various methods of estimation often lead to the same estimate, especially if the distribution is symmetric (such as the distribution of errors you make in throwing darts at a dart board). But in an asymmetric case such as a distribution of incomes, the results may differ among the contending modes of estimation. So the entire topic is more messy than appears at first look. Though we will not inquire into the complexities, it is important that you understand that the

matter is not as simple as it may seem. (See Savage (1972), Chapter 15, for more discussion of this topic.)

19.1 Ways to estimate the mean

19.1.1 The Method of Moments

Since elementary school you have been taught to estimate the mean of a universe (or calculate the mean of a sample) by taking a simple arithmetic average. A fancy name for that process is "the method of moments." It is the equivalent of estimating the center of gravity of a pole by finding the place where it will balance on your finger. If the pole has the same size and density all along its length, that balance point will be halfway between the endpoints, and the point may be thought of as the arithmetic average of the distances from the balance point of all the one-centimeter segments of the pole.

Consider this example:

Example: Twenty-nine Out of Fifty People Polled Say They Will Vote For The Democrat. Who Will Win The Election? The Relationship Between The Sample Proportion and The Population Proportion in a Two-Outcome Universe.

You take a random sample of 50 people in Maryland and ask which party's candidate for governor they will vote for. Twenty-nine say they will vote for the Democrat. Let's say it is reasonable to assume in this case that people will vote exactly as they say they will. The statistical question then facing you is: What proportion of the voters in Maryland will vote for the Democrat in the general election?

Your intuitive best guess is that the proportion of the "universe" — which is composed of voters in the general election, in this case — will be the same as the proportion of the sample. That is, 58 percent = 29/50 is likely to be your guess about the proportion that will vote Democratic. Of course, your estimate may be too high or too low in this particular case, but in the long run — that is, if you take many samples like this one — on the average the sample mean will equal the universe (population) proportion, for reasons to be discussed later.

The sample mean seems to be the "natural" estimator of the population mean in this and many other cases. That is, it seems quite natural to say that the best estimate is the sample mean, and indeed it probably is best. But why? This is the problem of inverse probability that has bedeviled statisticians for two centuries.

If the only information that you have (or that seems relevant) is the evidence of the sample, then there would seem to be no basis for judging that the shape and location of the population differs to the "left" or "right" from that of the sample. That is often a strong argument.

Another way of saying much the same thing: If a sample has been drawn randomly, each single observation is a representative estimator of the mean; if you only have one observation, that

observation is your best guess about the center of the distribution (if you have no reason to believe that the distribution of the population is peculiar — such as not being symmetrical). And therefore the sum of 2, 3…n of such observations (divided by their number) should have that same property, based on basic principles.

But if you are on a ship at sea and a leaf comes raining down from the sky, your best guess about the location of the tree from which it comes is not directly above you, and if two leaves fall, the midpoint of them is not the best location guess, either; you know that trees don't grow at sea, and birds sometimes carry leaves out to sea.

We'll return to this subject when we discuss criteria of methods.

19.1.2 Expected Value and the Method of Moments

Consider this gamble: You and another person roll a die. If it falls with the "6" upwards you get $4, and otherwise you pay $1. If you play 120 times, at the end of the day you would expect to have (20 * $4 - 100 * $1 =) -$20 dollars. We say that -$20 is your "expected value," and your expected value per roll is (-$20 / 120 =) $.166 or the loss of 1/6 of a dollar. If you get $5 instead of $4, your expected value is $0.

This is exactly the same idea as the method of moments, and we even use the same term — "expected value," or "expectation" — for the outcome of a calculation of the mean of a distribution. We say that the expected value for the success of rolling a "6" with a single cast of a die is 1/6, and that the expected value of rolling a "6" or a "5" is (1/6 + 1/6 =) 2/6.

19.1.3 The Maximum Likelihood Principle

Another way of thinking about estimation of the population mean asks: Which population(s) would, among the possible populations, have the highest probability of producing the observed sample? This criterion frequently produces the same answer as the method of moments, but in some situations the estimates differ. Furthermore, the logic of the maximum-likelihood principle is important.

Consider that you draw without replacement six balls — 2 black and 4 white — from a bucket that contains twenty balls. What would you guess is the composition of the bucket from which they were drawn? Is it likely that those balls came from a bucket with 4 white and 16 black balls? Rather obviously not, because it would be most unusual to get all the 4 white balls in your draw. Indeed, we can estimate the probability of that happening with simulation or formula to be about .003.

How about a bucket with 2 black and 18 whites? The probability is much higher than with the previous bucket, but it still is low — about .075.

Let us now estimate the probabilities for all buckets across the range of probabilities. In Figure 19.1 we see that the bucket with the highest probability of producing the observed sample has the same proportions of black and white balls as does the sample. This is called the "maximum likelihood universe." Nor should this be very surprising, because that universe obviously has an equal chance of producing samples with proportions below and above that observed proportion — as was discussed in connection with the method of moments.

We should note, however, that the probability that even such a maximum-likelihood universe would produce *exactly* the observed sample is very low (though it has an even lower probability of producing any *other* sample).

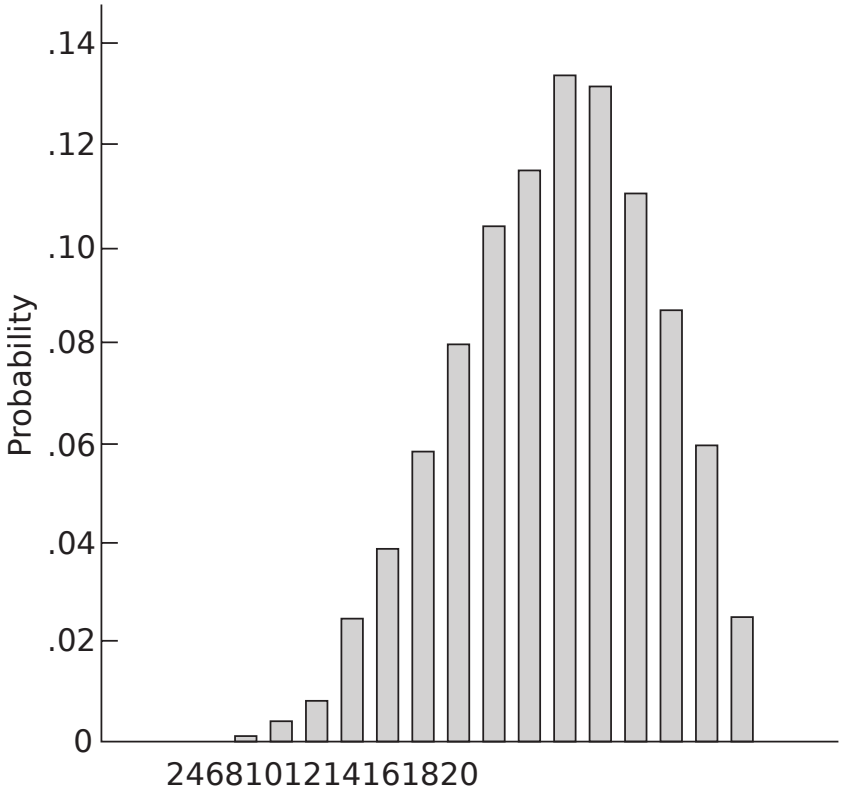


Figure 19.1: Number of White Balls in the Universe (N=20)

19.2 Choice of Estimation Method

When should you base your estimate on the method of moments, or of maximum likelihood, or still some other principle? There is no general answer. Sound estimation requires that you think long and hard about the purpose of your estimation, and fit the method to the purpose. I am well aware that this is a very vague statement. But though it may be an uncomfortable

idea to live with, guidance to sound statistical method must be vague because it requires sound judgment and deep knowledge of the particular set of facts about the situation at hand.

19.3 Criteria of estimates

How should one judge the soundness of the process that produces an estimate? General criteria include *representativeness* and *accuracy*. But these are pretty vague; we'll have to get more specific.

19.3.1 Unbiasedness

Concerning representativeness: We want a procedure that will not be systematically in error in one direction or another. In technical terms, we want an "unbiased estimate," if possible. "Unbiased" in this case does not mean "friendly" or "unprejudiced," but rather implies that on the average — that is, in the long run, after taking repeated samples — estimates that are too high will about balance (in percentage terms) those that are too low. The mean of the universe (or the proportion, if we are speaking of two-valued "binomial situations") is a frequent object of our interest. And the sample mean is (in most cases) an unbiased estimate of the population mean.

Let's now see an informal proof that the mean of a randomlydrawn sample is an "unbiased" estimator of the population mean. That is, the errors of the sample means will cancel out after repeated samples because the mean of a large number of sample means approaches the population mean. A second "law" to be informally proven is that the size of the inaccuracy of a sample proportion is largest when the population proportion is near 50 percent, and smallest when it approaches zero percent or 100 percent.

The statement that the sample mean is an unbiased estimate of the population mean holds for *many but not all kinds* of samples — proportions of two-outcome (Democrat-Republican) events (as in this case) and also the means of many measured-data universes (heights, speeds, and so on) that we will come to later.

But, you object, I have only *said* that this is so; I haven't proven it. Quite right. Now we will go beyond this simple assertion, though we won't reach the level of formal proof. This discussion applies to conventional analytic statistical theory as well as to the resampling approach.

We want to know why the mean of a repeated sample — or the proportion, in the case of a binomial universe — tends to equal the mean of the universe (or the proportion of a binomial sample). Consider a population of one thousand voters. Split the population into random sub-populations of 500 voters each; let's call these sub-populations by the name "samples." Almost inevitably, the proportions voting Democratic in the samples will *not exactly* equal the "true" proportions in the population. (Why not? Well, why should they split evenly? There is no general reason why they should.) But if the sample proportions do *not* equal the population

proportion, we can say that the extent of the difference between the two sample proportions and the population proportion will be *identical but in the opposite direction.* If the population proportion is 600/1000 = 60 percent, and one sample's proportion is 340/500 = 68 percent, then the other sample's proportion must be (600-340 = 260)/500 = 52 percent. So if in the very long run you would choose each of these two samples about half the time (as you would if you selected between the two samples randomly) the average of the sample proportions would be (68 percent + 52 percent)/2 = 60 percent. This shows that on the average the sample proportion is a fair and unbiased estimate of the population proportion — if the sample is half the size of the population.

If we now sub-divide each of our two samples of 500 (each of which was half the population size) into equal-size subsamples of 250 each, the same argument will hold for the proportions of the samples of 250 with respect to the sample of 500: The proportion of a 250-voter sample is an unbiased estimate of the proportion of the 500-voter sample from which it is drawn. It seems inductively reasonable, then, that if the proportion of a 250-voter sample is an unbiased estimate of the 500-voter sample from which it is drawn, and the proportion of a 500-voter sample is an unbiased estimate of the 1000-voter population, then the proportion of a 250-voter sample should be an unbiased estimate of the population proportion. And if so, this argument should hold for samples of 1/2 x 250 = 125, and so on — in fact for *any* size sample.

The argument given above is not a rigorous formal proof. But I doubt that the non-mathematician needs, or will benefit from, a more formal proof of this proposition. You are more likely to be persuaded if you demonstrate this proposition to yourself experimentally in the following manner:

- **Step 1.** Let "1-6" = Democrat, "7-10" = Republican
- **Step 2.** Choose a sample of, say, ten random numbers, and record the proportion Democrat (the sample proportion).
- **Step 3.** Repeat step 2 a thousand times.
- **Step 4.** Compute the mean of the sample proportions, and compare it to the population proportion of 60 percent. This result should be close enough to reassure you that *on the average* the sample proportion is an "unbiased" estimate of the population proportion, though in any particular sample it may be substantially off in either direction.

19.3.2 Efficiency

We want an estimate to be accurate, in the sense that it is as close to the "actual" value of the parameter as possible. Sometimes it is possible to get more accuracy at the cost of biasing the estimate. More than that does not need to be said here.

19.3.3 Maximum Likelihood

Knowing that a particular value is the most likely of all values may be of importance in itself. For example, a person betting on one horse in a horse race is interested in his/her estimate of the winner having the highest possible probability, and is not the slightest bit interested in getting *nearly* the right horse. Maximum likelihood estimates are of particular interest in such situations.

See (Savage 1972, chap. 15), for many other criteria of estimators.

19.4 Criteria of the Criteria

What should we look for in choosing criteria? Logically, this question should precede the above list of criteria.

Savage (1972, chap. 15) has urged that we should always think in terms of the *consequences* of choosing criteria, in light of our purposes in making the estimate. I believe that he is making an important point. But it often is very hard work to think the matter through all the way to the consequences of the criteria chosen. And in most cases, such fine inquiry is not needed, in the sense that the estimating procedure chosen will be the same no matter what consequences are considered.[1]

19.5 Estimation of accuracy of the point estimate

So far we have discussed how to make a point estimate, and criteria of good estimators. We also are interested in estimating the accuracy of that estimate. That subject — which is harder to grapple with — is discussed in Chapter 26 and Chapter 27 on confidence intervals.

Most important: One cannot sensibly talk about the accuracy of probabilities in the abstract, without reference to some set of facts. In the abstract, the notion of accuracy loses any meaning, and invites confusion and argument.

19.6 Uses of the mean

Let's consider when the use of a device such as the mean is valuable, in the context of the data on marksmen in Table 19.1.[2]. If we wish to compare marksman A versus marksman B,

[1] Here I shall merely mention that the method of moments and the method of maximum likelihood serve most of our needs, and often agree in their conclusions; furthermore, we often know when the former may be inappropriate.

[2] This discussion follows (Gnedenko, Aleksandr, and Khinchin 1962, chap. 8).

we can immediately see that marksman A hit the bullseye (80 shots for 3 points each time) as many times as marksman B hit *either* the bullseye or simply got in the black (30 shots for 3 points and 50 shots for 2 points), and A hit the black (2 points) as many times as B just got in the white (1 point). From these two comparisons covering all the shots, in both of which comparisons A does better, it is immediately obvious that marksman A is better than marksman B. We can say that A's score *dominates* B's score.

Table 19.1: Score percentages by marksman

Score	# occurrences	Probability
Marksman A		
1	0	0
2	20	.2
3	80	.8
Marksman B		
1	20	.2
2	50	.5
3	30	.3
Marksman C		
1	10	.1
2	60	.6
3	30	.3

When we turn to comparing marksman C to marksman D, however, we cannot say that one "dominates" the other as we could with the comparison of marksmen A and B. Therefore, we turn to a summarizing device. One such device that is useful here is the mean. For marksman C the mean score is $(40 * 1) + (10 * 2) + (50 * 3) = 210$, while for marksman D the mean score is $(10 * 1) + (60 * 2) + (30 * 3) = 220$. Hence we can say that D is better than C even though D's score does not dominate C's score in the bullseye category.

Another use of the mean (Gnedenko, Aleksandr, and Khinchin 1962, 68) is shown in the estimation of the number of matches that we need to start fires for an operation carried out 20 times in a day (Table 19.2). Let's say that the number of cases where s/he needs 1, 2 … 5 matches to start a fire are as follows (along with their probabilities) based on the last 100 fires started:

Table 19.2: Number of matches needed to start a fire

Number of Matches	Number of Cases	Probabilities
1	7	.16
2	16	.16
3	55	.55

Number of Matches	Number of Cases	Probabilities
4	21	.21
5	1	.01

If you know that the operator will be lighting twenty fires, you can estimate the number of matches that s/he will need by multiplying the mean number of matches (which turns out be $1 * .07 + 2 * 0.16 + 3 * 0.55 + 4 * 0.21 + 5 * 0.01 = 2.93$) in the observed experience by 20. Here you are using the mean as an indication of a representative case.

It is common for writers to immediately produce the data in the forms of percentages or probabilities. But I think it is important to include in our discussion the absolute numbers, because this is what one must begin with in practice. And keeping the absolute numbers in mind is likely to avoid some confusions that arise if one immediately goes to percentages or to probabilities.

Still another use for the mean is when you have a set of observations with error in them. The mean of the observations probably is your best guess about which is the "right" one. Furthermore, the distance you are likely to be off the mark is less if you select the mean of the observations. An example might be a series of witnesses giving the police their guesses about the height of a man who overturned an outhouse. The mean probably is the best estimate to give to police officers as a description of the perpetrator (though it would be helpful to give the range of the observations as well).

We use the mean so often, in so many different circumstances, that we become used to it and never think about its nature. So let's do so a bit now.

Different statistical ideas are appropriate for business and engineering decisions, biometrics, econometrics, scientific explanation (the philosophers' case), and other fields. So nothing said here holds everywhere and always.

One might ask: What is the "meaning" of a mean? But that is not a helpful question. Rather, we should ask about the *uses* of a mean. Usually a mean is used to summarize a set of data. As we saw with marksmen C and D, it often is difficult to look at a table of data and obtain an overall idea of how big or how small the observations are; the mean (or other measurements) can help. Or if you wish to compare two sets of data where the distributions of observations overlap each other, comparing the means of the two distributions can often help you better understand the matter.

Another complication is the confusion between *description* and *estimation*, which makes it difficult to decide where to place the topic of descriptive statistics in a textbook. For example, compare the mean income of all men in the U. S., as measured by the decennial census. This mean of the universe can have a very different meaning from the mean of a sample of men with respect to the same characteristic. The sample mean is a point estimate, a statistical device, whereas the mean of the universe is a description. The use of the mean as an estimator

is fraught with complications. Still, maybe it is no more complicated than deciding what describer to use for a population. This entire matter is much more complex than it appears at first glance.

When the sample size approaches in size the entire population — when the sample becomes closer and closer to being the *same* as the population — the two issues blend. What does that tell us? Anything? What is the relationship between a baseball player's average for two weeks, and his/her lifetime average? This is subtle stuff — rivaling the subtleness of arguments about inference versus probability, and about the nature of confidence limits (see Chapter 26 and Chapter 27). Maybe the only solid answer is to try to stay super-clear on what you are doing for what purpose, and to ask continually what job you want the statistic (or describer) to do for you.

The issue of the relationship of sample size to population size arises here. If the sample size equals or approaches the population size, the very notion of estimation loses its meaning.

The notion of "best estimator" makes no sense in some situations, including the following: a) You draw one black ball from a bucket. You cannot put confidence intervals around your estimate of the proportion of black balls, except to say that the proportion is somewhere between 1 and 0. No one would proceed without bringing in more information. That is, when there is almost no information, you simply cannot make much of an estimate — and the resampling method breaks down, too. It does not help much to shift the discussion to the models of the buckets, because then the issue is the unknown population of the buckets, in which case we need to bring in our general knowledge. b) When the sample size equals or is close to the population size, as discussed in this section, the data are a description rather than an estimate, because the sample is getting to be much the same as the universe; that is, if there are twelve people in your family, and you randomly take a sample of the amount of sugar used by eight members of the family, the results of the sample cannot be very different than if you compute the amount for all twelve family members. In such a case, the interpretation of the mean becomes complex.

Underlying all estimation is the assumption of continuation, which follows from random sampling — that there is no reason to expect the next sample to be different from the present one in any *particular* fashion, mean or variation. But we do expect it to be different in *some* fashion because of sampling variability.

19.7 Conclusion

A *Newsweek* article says, "According to a recent reader's survey in Bride's magazine, the average blowout [wedding] will set you back about $16,000" (Feb 15, 1993, p. 67). That use of the mean (I assume) for the average, rather than the median, could cost the parents of some brides a pretty penny. It could be that the cost for the average *person* — that is, the median expenditure — might be a lot less than $16,000. (A few million dollar weddings could have

a huge effect on a survey mean.) An inappropriate standard of comparison might enter into some family discussions as a result of this article, and cause higher outlays than otherwise. This chapter helps one understand the nature of such estimates.

20 Framing Statistical Questions

20.1 Introduction

Chapter 3 - Chapter 15 discussed problems in probability theory. That is, we have been estimating the probability of a composite event *resulting from a system in which we know the probabilities of the simple events* — the "parameters" of the situation.

Then Chapter 17 - Chapter 19 discussed the underlying philosophy of statistical inference.

Now we turn to inferential-statistical problems. Up until now, we have been estimating the complex probabilities of *known* universes — the topic of *probability*. Now as we turn to problems in *statistics*, we seek to learn the characteristics of an unknown system — the basic probabilities of its simple events and parameters. (Here we note again, however, that in the process of dealing with them, all statistical-inferential problems eventually are converted into problems of pure probability). To assess the characteristics of the system in such problems, we employ the characteristics of the sample(s) that have been drawn from it.

For further discussion on the distinction between inferential statistics and probability theory, see Chapter 2 - Chapter 3.

This chapter begins the topic of *hypothesis testing*. The issue is: whether to adjudge that a particular sample (or samples) come(s) from a particular universe. A two-outcome yes-no universe is discussed first. Then we move on to "measured-data" universes, which are more complex than yes-no outcomes because the variables can take on many values, and because we ask somewhat more complex questions about the relationships of the samples to the universes. This topic is continued in subsequent chapters.

In a typical hypothesis-testing problem presented in this chapter, one sample of hospital patients is treated with a new drug and a second sample is not treated but rather given a "placebo." After obtaining results from the samples, the "null" or "test" or "benchmark" hypothesis would be that the resulting drug and placebo samples are drawn from the same universe. This device of the null hypothesis is the equivalent of stating that the drug had no effect on the patients. It is a special intellectual strategy developed to handle such statistical questions.

We start with the scientific question: Does the medicine have an effect? We then translate it into a testable statistical question: How likely is it that the sample means come from the same universe? This process of question-translation is the crucial step in hypothesis-testing and

inferential statistics. The chapter then explains how to solve these problems using resampling methods after you have formulated the proper statistical question.

Though the examples in the chapter mostly focus on tests of hypotheses, the procedures also apply to confidence intervals, which will be discussed later.

20.2 Translating scientific questions into probabilistic and statistical questions

The first step in using probability and statistics is to translate the *scientific* question into a *statistical* question. Once you know exactly which prob-stats question you want to ask — that is, exactly which probability you want to determine — the rest of the work is relatively easy (though subtle). The stage at which you are most likely to make mistakes is in stating the question you want to answer in probabilistic terms.

Though this translation is difficult, it involves no mathematics. Rather, this step requires only hard thought. You cannot beg off by saying, "I have no brain for math!" The need is for a brain that will do clear thinking, rather than a brain especially talented in mathematics. A person who uses conventional methods can avoid this hard thinking by simply grabbing the formula for some test without understanding why s/he chooses that test. But resampling pushes you to do this thinking explicitly.

This crucial process of translating from a pre-statistical question to a statistical question takes place in all statistical inference. But its nature comes out most sharply with respect to testing hypotheses, so most of what will be said about it will be in that context.

20.3 The three types of questions

Let's consider the natures of conceptual, operational, and statistical questions.

20.3.1 The Scientific Question

A study for either scientific or decision-making purposes properly begins with a general question about the nature of the world — that is, a conceptual or theoretical question. One must then transform this question into an operational-empirical form that one can study scientifically. Thence comes the translation into a technical-statistical question.

The scientific-conceptual-theoretical question can be an issue of theory, or a policy choice, or the result of curiosity at large.

Examples include: Can a bioengineer increase the chance of female calves being born? Is copper becoming less scarce? Are the prices of liquor systematically different in states where the liquor stores are publicly owned compared to states where they are privately owned? Does a new formulation of pig rations lead to faster hog growth? Was the rate of unemployment higher last month than the long-run average, or was the higher figure likely to be the result of sampling error? What are the margins of probable error for an unemployment survey?

20.3.2 The Operational-Empirical Question

The operational-empirical question is framed in measurable quantities in a meaningful design. Examples include: How likely is this state of affairs (say, the new pig-food formulation) to cause an event such as was observed (say, the observed increase in hog growth)? How likely is it that the mean unemployment rate of a sample taken from the universe of interest (say, the labor force, with an unemployment rate of 10 percent) will be between 11 percent and 12 percent? What is the probability of getting three girls in the first four children if the probability of a girl is .48? How unlikely is it to get nine females out of ten calves in an experiment on your farm? Did the price of copper fall between 1800 and the present? These questions are in the form of empirical questions, which have already been transformed by operationalizing from scientific-conceptual questions.

20.3.3 The Statistical Question

At this point one must decide whether the conceptual-scientific question is of the form of either a) or b):

a. A test about whether some sample will frequently happen by chance rather than being very surprising — a test of the "significance" of a hypothesis. Such hypothesis testing takes the following form: How likely is a given "universe" to produce *some* sample like x? This leads to interpretation about: How likely is a given universe to be the cause of *this observed* sample?
b. A question about the accuracy of the estimate of a parameter of the population based upon sample evidence (an inquiry about "confidence intervals"). This sort of question is considered by some (but not by me) to be a question in estimation — that is, one's best guess about (say) the magnitude and probable error of the mean or median of a population. This is the form of a question about confidence limits — how likely is the mean to be between x and y?

Notice that the statistical question is framed as a question in probability.

20.4 Illustrative translations

The best way to explain how to translate a scientific question into a statistical question is to illustrate the process.

20.4.1 Illustration A — beliefs about smoking

Were doctors' beliefs as of 1964 about the harmfulness of cigarette smoking (and doctors' own smoking behavior) affected by the *social* groups among whom the doctors live (Simon 1967)? That was the *theoretical* question. We decided to define the doctors' *reference groups* as the *states* in which they live, because data about doctors and smoking were available state by state (*Modern Medicine*, 1964). We could then translate this question into an operational and testable *scientific* hypothesis by asking this question: Do doctors in tobacco-economy states differ from doctors in other states in their smoking, and in their beliefs about smoking?

Which numbers would help us answer this question, and how do we interpret those numbers? We now were ready to ask the *statistical* question: Do doctors in tobacco-economy states "belong to the same universe" (with respect to smoking) as do other doctors? That is, do doctors in tobacco-economy states have the same characteristics — at least, those characteristics we are interested in, smoking in this case — as do other doctors? Later we shall see that the way to proceed is to consider the statistical hypothesis that these doctors do indeed belong to that same universe; that hypothesis and the universe will be called "benchmark hypothesis" and "benchmark universe" respectively — or in more conventional usage, the "null hypothesis."

If the tobacco-economy doctors do indeed belong to the benchmark universe — that is, if the benchmark hypothesis is correct — then there is a 49/50 chance that doctors in some state *other than* the state in which tobacco is most important will have the highest rate of cigarette smoking. But in fact we observe that the state in which tobacco accounts for the largest proportion of the state's income — North Carolina — had (as of 1964) a higher proportion of doctors who smoked than any other state. (Furthermore, a lower proportion of doctors in North Carolina than in any other state said that they *believed* that smoking is a health hazard.)

Of course, it is possible that it was just *chance* that North Carolina doctors smoked most, but the chance is only 1 in 50 if the benchmark hypothesis is correct. Obviously, *some* state had to have the highest rate, and the chance for any other state was also 1 in 50. But, because our original *scientific* hypothesis was that North Carolina doctors' smoking rate would be highest, and we then observed that it was highest even though the chance was only 1 in 50, the observation became interesting and meaningful to us. It means that the chances are strong that there was a connection between the importance of tobacco in the economy of a state and the rate of cigarette smoking among doctors living there (as of 1964).

To consider this problem from another direction, it would be rare for North Carolina to have the highest smoking rate for doctors if there were no special reason for it; in fact, it would

occur only once in fifty times. But, if there *were* a special reason — and we hypothesize that the tobacco economy provides the reason — then it would *not* seem unusual or rare for North Carolina to have the highest rate; therefore we choose to believe in the not-so-unusual phenomenon, that the tobacco economy caused doctors to smoke cigarettes.

Like many (most? all?) actual situations, the cigarettes and doctors' smoking issue is a rather messy business. Did I have a clear-cut, theoretically-derived prediction before I began? Maybe I did a bit of "data dredging" — that is, maybe I started with a vague expectation, and only arrived at my sharp hypothesis after I saw the data. This would weaken the probabilistic interpretation of the test of significance — but this is something that a scientific investigator does not like to do because it weakens his/her claim for attention and chance of publication. On the other hand, if one were a Bayesian, one could claim that one had a prior probability that the observed effect would occur, and the observed data strengthens that prior; but this procedure would not seem proper to many other investigators. The only wholly satisfactory conclusion is to obtain more data — but as of 1993, there does not seem to have been another data set collected since 1964, and collecting a set by myself is not feasible.

This clearly is a case of statistical inference that one could argue about, though perhaps it is true that *all* cases where the data are sufficiently ambiguous as to require a test of significance are also sufficiently ambiguous that they are properly subject to argument.

For some decades the hypothetico-deductive framework was the leading point of view in empirical science. It insisted that the empirical and statistical investigation should be preceded by theory, and only propositions suggested by the theory should be tested. Investigators were not supposed to go back and forth from data to theory to testing. It is now clear that this is an ivory-tower irrelevance, and no one lived by the hypothetico-deductive strictures anyway — just pretended to. Furthermore, there is no sound reason to feel constrained by it, though it strengthens your conclusions if you had theoretical reason in advance to expect the finding you obtained.

20.4.2 Illustration B — is it a cure?

Does medicine CCC cure some particular cancer? That's the scientific question. So you give the medicine to six patients who have the cancer and you do not give it to six similar patients who have the cancer. Your sample contains only twelve people because it is not feasible for you to obtain a larger sample. Five of six "medicine" patients get well, two of six "no medicine" patients get well. Does the medicine cure the cancer? That is, if future cancer patients take the medicine, will their rate of recovery be higher than if they did not take the medicine?

One way to translate the scientific question into a statistical question is to ask: Do the "medicine" patients *belong to the same universe* as the "no medicine" patients? That is, we ask whether "medicine" patients still have the *same* chances of getting well from the cancer as do the "no medicine" patients, or whether the medicine has bettered the chances of those who took it and thus removed them from the original universe, with its original chances of getting

well. The original universe, to which the "no medicine" patients must still belong, is the benchmark universe. Shortly we shall see that we proceed by comparing the observed results against the benchmark *hypothesis* that the "medicine" patients still belong to the benchmark *universe* — that is, they still have the same chance of getting well as the "no medicine" patients.

We want to know whether or not the medicine does any good. This question is the same as asking whether patients who take medicine are still in the same population (universe) as "no medicine" patients, or whether they now belong to a different population in which patients have higher chances of getting well. To recapitulate our translations, we move from asking: Does the medicine cure the cancer? to, Do "medicine" patients have the same chance of getting well as "no medicine" patients?; and finally, to: Do "medicine" patients belong to the same universe (population) as "no medicine" patients? Remember that "population" in this sense does not refer to the population at large, but rather to a group of cancer sufferers (perhaps an infinitely large group) who have given chances of getting well, on the average. Groups with different chances of getting well are called "different populations" (universes). Shortly we shall see how to *answer* this statistical question. We must keep in mind that our ultimate concern in cases like this one is to *predict future results* of the medicine, that is, to predict whether use of the medicine will lead to a higher recovery rate than would be observed without the medicine.

20.4.3 Illustration C — a better method for teaching reading

Is method Alpha a better method of teaching reading than method Beta? That is, will method Alpha produce a higher average reading score in the future than will method Beta? Twenty children taught to read with method Alpha have an average reading score of 79, whereas children taught with method Beta have an average score of 84. To translate this *scientific* question into a *statistical* question we ask: Do children taught with method Alpha come from the same universe (population) as children taught with method Beta? Again, "universe" (population) does *not* mean the town or social group the children come from, and indeed the experiment will make sense only if the children *do* come from the same population, in that sense of "population." What we want to know is whether or not the children belong to the same *statistical* population (universe), *defined according to their reading ability, after they have studied* with method Alpha or method Beta.

20.4.4 Illustration D — better fertilizer

If one plot of ground is treated with fertilizer, and another similar plot is not treated, the benchmark (null) hypothesis is that the corn raised on the treated plot is no different than the corn raised on the untreated lot — that is, that the corn from the treated plot comes from ("belongs to") the same universe as the corn from the untreated plot. If our statistical test makes it seem very unlikely that a universe like that from which the untreated-plot corn comes would *also* produce corn such as came from the treated plot, then we are willing to believe

that the fertilizer has an effect. For a psychological example, substitute the words "group of children" for "plot," "special training" for "fertilizer," and "I.Q. score" for "corn."

There is nothing sacred about the benchmark (null) hypothesis of "no difference." You could just as well test the benchmark hypothesis that the corn comes from a universe that averages 110 bushels per acre, if you have reason to be especially interested in knowing whether or not the fertilizer produces more than 110 bushels per acre. But in many cases it is reasonable to test the probability that a sample comes from the population that does not receive the special treatment of medicine, fertilizer, or training.

20.5 Generalizing from sample to universe

So far we have discussed the scientific question and the statistical question. Remember that there is always a generalization question, too: Do the statistical results from this particular sample of, say, rats apply to a universe of humans? This question can be answered only with wisdom, common sense, and general knowledge, and not with probability statistics.

Translating from a scientific question into a statistical question is mostly a matter of asking the probability that some given benchmark universe (population) will produce one or more observed samples. Notice that we must (at least for general scientific testing purposes) ask about a *given* universe whose composition we assume to be *known*, rather than about a *range* of universes, or about a universe whose properties are unknown. In fact, there is really only one question that probability statistics can answer: Given some particular benchmark universe of some stated composition, what is the probability that an observed sample would come from it? (Please notice the subtle but all-important difference between the words "would come" in the previous sentence, and the word "came.") A variation of this question is: Given two (or more) samples, what is the probability that they would come from the *same* universe — that is, that the same universe would produce both of them? In this latter case, the relevant benchmark universe is implicitly the universe whose composition is the two samples combined.

The necessity for stating the characteristics of the universe in question becomes obvious when you think about it for a moment. Probability-statistical testing adds up to comparing a sample with a particular benchmark universe, and asking whether there probably is a difference between the sample and the universe. To carry out this comparison, we ask *how likely* it is that the benchmark universe would produce a sample like the observed sample.

But in order to find out whether or not a universe could produce a given sample, we must ask whether or not some *particular* universe — with stated characteristics — could produce the sample. There is no doubt that *some* universe could produce the sample by a random process; in fact, some universe did. The only sensible question, then, is whether or not a *particular* universe, with stated (or known) characteristics, is likely to produce such a sample. In the case of the medicine, the universe with which we compare the sample who took the medicine is the benchmark universe to which that sample would belong if the medicine had had no

effect. This comparison leads to the benchmark (null) hypothesis that the sample comes from a population in which the medicine (or other experimental treatment) seems to have *no effect*. It is to avoid confusion inherent in the term "null hypothesis" that I replace it with the term "benchmark hypothesis."

The concept of the benchmark (null) hypothesis is not easy to grasp. The best way to learn its meaning is to see how it is used in practice. For example, we say we are willing to believe that the medicine has an effect if it seems very unlikely from the number who get well that the patients given the medicine still belong to the same benchmark universe as the patients given no medicine at all — that is, if the benchmark hypothesis is unlikely.

20.6 The steps in statistical inference

These are the steps in conducting statistical inference

- **Step 1.** Frame a question in the form of: *What is the chance of getting the observed sample x from some specified population X?* For example, what is the probability of getting a sample of 9 females and one male from a population where the probability of getting a single female is .48?
- **Step 2.** Reframe the question in the form of: *What kinds of samples does population X produce, with which probabilities?* That is, what is the probability of the observed sample x (9 females in 10 calves), given that a population is X (composed of 48 percent females)? Or in notation, what is $P(x|X)$?
- **Step 3.** Actually *investigate the behavior* of S with respect to S and other samples. This can be done in two ways:

1. Use the calculus of probability (the formulaic method), perhaps resorting to the Monte Carlo method if an appropriate formula does not exist. Or
2. Resampling (in the larger sense), which equals the Monte Carlo method minus its use for approximations, investigation of complex functions in statistics and other theoretical mathematics, and non-resampling uses elsewhere in science. Resampling in the more restricted sense includes bootstrap, permutation, and other non-parametric methods. More about the resampling procedure follows in the paragraphs to come, and then in later chapters in the book.

- **Step 4.** Interpret the probabilities that result from step 3 in terms of acceptance or rejection of hypotheses, surety of conclusions, and as inputs to decision theory.[1]

The following short definition of statistical inference summarizes the previous four steps:

[1]These steps are discussed in more philosophic depth in my (JLS's) online book on the philosophy of statistics and resampling.

> Statistical inference equals the selection of a probabilistic model to resemble the process you wish to investigate, the investigation of that model's behavior, and the interpretation of the results.

Stating the steps to be followed in a procedure is an operational definition of the procedure. My belief in the clarifying power of this device (the operational definition) is embodied in the set of steps given in Chapter 15 for the various aspects of statistical inference. A canonical question-and-answer procedure for testing hypotheses will be found in Chapter 25, and one for confidence intervals will be found in Chapter 26.

20.7 Summary

We define resampling to include problems in inferential statistics as well as problems in probability as follows: *Using the entire set of data you have in hand, or using the given data-generating mechanism (such as a die) that is a model of the process you wish to understand, produce new samples of simulated data, and examine the results of those samples.* That's it in a nutshell. In some cases, it may also be appropriate to amplify this procedure with additional assumptions.

Problems in pure probability may at first seem different in nature than problems in statistical inference. But the same logic as stated in this definition applies to both varieties of problems. The difference is that in probability problems the "model" is known in advance — say, the model implicit in a deck of poker cards plus a game's rules for dealing and counting the results — rather than the model being assumed to be best estimated by the observed data, as in resampling statistics.

The hardest job in using probability statistics, and the most important, is to *translate* the scientific question into a form to which statistics can give a sensible answer. You must translate scientific questions into the appropriate form for *statistical operations*, so that you know which operations to perform. This is the part of the job that requires hard, clear thinking — though it is non-mathematical thinking — and it is the part that someone else usually cannot easily do for you.

Once you know exactly which probability-statistical question you want to ask — that is, exactly which probability you want to determine — the rest of the work is relatively easy. The stage at which you are most likely to make mistakes is in stating the question you want to answer in probabilistic terms. Though this step is hard, *it involves no mathematics.* This step requires only *hard, clear thinking.* You cannot beg off by saying "I have no brain for math!" To flub this step is to admit that you have no brain for clear thinking, rather than no brain for mathematics.

21 Hypothesis-Testing with Counted Data, Part 1

21.1 Introduction

The first task in inferential statistics is to make one or more *point estimates* — that is, to make one or more statements about *how much* there is of something we are interested in — including especially the mean and the dispersion. (That work goes under the label "estimation" and is discussed in Chapter 19.) Frequently the next step, after making such quantitative estimation of the universe from which a sample has been drawn, is to consider whether two or more samples are different from each other, or whether the single sample is different from a specified value; this work goes under the label "hypothesis testing." We ask: Did something happen? Or: Is there a difference between two universes? These are yes-no questions.

In other cases, the next step is to inquire into the reliability of the estimates; this goes under the label "confidence intervals." (Some writers include assessing reliability under the rubric of estimation, but I judge it better not to do so).

So: Having reviewed how to convert hypothesis-testing problems into statistically testable questions in Chapter 20, we now must ask: How does one employ resampling methods to make the statistical test? As is always the case when using resampling techniques, there is no unique series of steps by which to proceed. The crucial criterion in assessing the model is whether it accurately simulates the actual event. With hypothesis-testing problems, any number of models may be correct. Generally speaking, though, the model that makes fullest use of the quantitative information available from the data is the best model.

When attempting to deduce the characteristics of a universe from sample data, or when asking whether a sample was drawn from a particular universe, a crucial issue is whether a "one-tailed test" or a "two-tailed test" should be applied. That is, in examining the results of our resampling experiment based on the benchmark universe, do we examine both ends of the frequency distribution, or just one? If there is strong reason to believe *a priori* that the difference between the benchmark (null) universe and the sample will be in a given direction — for example if you hypothesize that the sample mean will be *smaller* than the mean of the benchmark universe — you should then employ a *one-tailed test*. If you do *not* have strong basis for such a prediction, use the *two-tailed* test. As an example, when a scientist tests a new medication, his/her hypothesis would be that the number of patients who get well will be

higher in the treated group than in the control group. Thus, s/he applies the one-tailed test. See the text below for more detail on one- and two-tailed tests.

Some language first:

Hypothesis: In inferential statistics, a statement or claim about a universe that can be tested and that you wish to investigate.

Testing: The process of investigating the validity of a hypothesis.

Benchmark (or null) hypothesis: A particular hypothesis chosen for convenience when testing hypotheses in inferential statistics. For example, we could test the hypothesis that there is *no difference* between a sample and a given universe, or between two samples, or that a parameter is less than or greater than a certain value. The benchmark universe refers to this hypothesis. (The concept of the benchmark or null hypothesis was discussed in Chapter 9 and Chapter 20.)

Soon we will begin the actual statistical testing of various sorts of hypotheses about samples and populations.

But, before we get there, we will take a short technical detour.

21.1.1 Building strings for labels and messages

> Note 39: Notebook: Building strings for labels
>
> - Download notebook
> - Interact

As the examples in this book proceed, we will use more code techniques to write the examples in a more concise and efficient way.

One task that we often have, is to build up helpful strings to use as labels on plots, or to print out as messages. These strings will often mix numbers and text. For example, we may want to print out a helpful message such as: `Simulation using 10000 trials`, where the number 10000 in the message comes from some variable, such as `n_trials`. Let's set that variable now:

```
n_trials = 10000
```

Building the useful string above would involve taking the string `'Simulation using '`, then appending a string to represent the number 10,000 — as in:

```
# Convert the number (integer) to a string representing the number.
str(n_trials)
```

```
'10000'
```

Finally we need to append another string to the result — `' trials.'`. So our task will be to *concatenate* (stick together) these three strings.

There are several ways to concatenate strings in Python. For example, Python interprets `+`, between strings, to mean *concatenate*. One way to make a new string that concatenates the strings `'resampling '`, `'is '` and `'better'` is to use `+`, like this:

```
# + between strings means "concatenate".
'resampling ' + 'is ' + 'better'
```

```
'resampling is better'
```

We could always insert a number as a string, by converting the number to a string, and concatenating, like this:

```
'resampling ' + 'is ' + str(100) + ' times ' + 'better'
```

```
'resampling is 100 times better'
```

However, as you can see, this starts looking rather ugly and verbose. It's easy to forget to append spaces to the strings to concatenate, and end up with messages like:

```
'resampling' + 'is' + str(100) + 'times' + 'better'
```

```
'resamplingis100timesbetter'
```

One of the most flexible ways to solve problems like this, is to use Python "f" (Format) strings. Format strings start with an `f` and can include values inside the string, enclosed in curly brackets. This is best explained by example:

```
# Our first Python format string.
# Notice the "f" prefix before the quotes, to tell Python this is a Format
# string, and may include values to interpolate, inside curly brackets.
f'resampling is {100} times better'
```

```
'resampling is 100 times better'
```

The Format string expects Python values to appear inside the string, enclosed in curly brackets. When Python see a Format ("f" string) like this, it gets the value named or typed inside the curly brackets, converts it to a string, and inserts that string into the result at the relevant place.

Format strings take a little getting used to, but once you are used to them, they are a flexible and concise way of assembling useful messages.

For example, to create the string we started this section with, we could write:

```
f'Simulation using {n_trials} trials.'
```

```
'Simulation using 10000 trials.'
```

Don't forget the `f` prefix to the string, if you do want to insert values like this, otherwise you'll get a standard (not-Format) string, and Python won't insert the value.

```
# Oops!  We forgot the "f" prefix to the string.  Python won't put in
# (interpolate) the string representation of the value.
'Simulation using {n_trials} trials.'
```

```
'Simulation using {n_trials} trials.'
```

> End of notebook: Building strings for labels
>
> `building_strings` starts at Note 39.

Now we begin with statistical testing for hypotheses on samples and populations.

21.2 Should a single sample of counted data be considered different from a benchmark universe?

21.2.1 Example: Does irradiation affect the sex ratio in fruit flies?

Where the Benchmark Universe Mean (in this case, the Proportion) is Known, is the Mean (Proportion) of the Population Affected by the Treatment?)

You think you have developed a technique for irradiating the genes of fruit flies so that the sex ratio of the offspring will *not* be half males and half females. In the first twenty cases you treat, there are fourteen males and six females. Does this experimental result confirm that the irradiation does work?

First convert the scientific question — whether or not the treatment affects the sex distribution — into a probability-statistical question: Is the observed sample likely to have come from a benchmark universe in which the sex ratio is one male to one female? The benchmark (null) hypothesis, then, is that the treatment makes no difference and the sample comes from the one-male-to-one-female universe. Therefore, we investigate *how likely a one-to-one universe is to produce a distribution of fourteen or more of just one sex.*

A coin has a one-to-one (one out of two) chance of coming up tails. Therefore, we might flip a coin in groups of twenty flips, and count the number of tails in each twenty flips. Or we can use a random number table. The following steps will produce a sound estimate:

- **Step 1.** Let tails = male, heads = female.
- **Step 2.** Flip twenty coins and count the number of males. If 14 or more males occur, record "yes." Also, if 6 or fewer males occur, record "yes" because this means we have gotten 14 or more females. Otherwise, record "no."
- **Step 3.** Repeat step 2 perhaps 100 times.
- **Step 4.** Calculate the proportion "yes" in the 100 trials. This proportion estimates the probability that a fruit-fly population with a propensity to produce 50 percent males will by chance produce as many as 14 or as few as 6 males in a sample of 20 flies.

Table 21.1: Results from 25 random trials for Fruitfly problem

Trial no	# of tails	>=14 or <=6
1	12	No
2	12	No
3	8	No
4	11	No
5	8	No
6	10	No
7	11	No
8	6	Yes
9	6	Yes
10	10	No
11	11	No
12	12	No
13	7	No
14	15	Yes
15	13	No

Trial no	# of tails	>=14 or <=6
16	9	No
17	9	No
18	10	No
19	10	No
20	9	No
21	12	No
22	11	No
23	4	Yes
24	16	Yes
25	7	No

Table 21.1 shows the results obtained in 25 trials of twenty flips each. In two of the twenty-five trials (8 percent) there were fourteen or more tails, which we call "males," and in three of the 25 trials (12 percent) there six or fewer tails, meaning there were fourteen or more heads ("females"). We can therefore estimate that, even if the treatment does *not* affect the sex and the births over a long period really are one to one, five out of twenty-five times (20 percent) we would get fourteen or more of one sex or the other. Therefore, finding fourteen males out of 20 births is not overwhelming evidence that the treatment has any effect, even though the result is suggestive.

How accurate is the estimate? Seventy-five more trials were made, and of the 100 trials nine contained fourteen or more "males" (9 percent), and 8 trials contained fourteen or more "females" (8 percent), a total of 17 percent. So the first twenty-five trials gave a fairly reliable indication. As a matter of fact, analytically-based computation (not explained here) shows that the probability of getting fourteen or more females out of twenty births is .057 and, of course, the same for fourteen or more males from a one-to-one universe, implying a total probability of .114 of getting fourteen or more males *or* females.

Now let us obtain larger and more accurate simulation samples with the computer. The key step in the Python notebook below represents male fruit flies with the string `'male'` and female fruit flies with the string `'female'`. The `rnd.choice` function is then used to generate 20 of these strings with an equal probability that either string is selected. This simulates randomly choosing 20 fruit flies on the benchmark assumption — the "null hypothesis" — that each fruit fly has an equal chance of being a male or female. Now we want to discover the chances of getting more than 13 (i.e., 14 or more) males or more than 13 females under these conditions. So we use `np.sum` to count the number of males in each random sample and then store this value in the `scores` array of this number for each sample. We repeat these steps 10,000 times.

After ten thousand samples have been drawn, we count (`sum`) how often there were more than 13 males and then count the number of times there were fewer than 7 males (because if there were fewer than 7 males there must have been more than 13 females). When we add the two

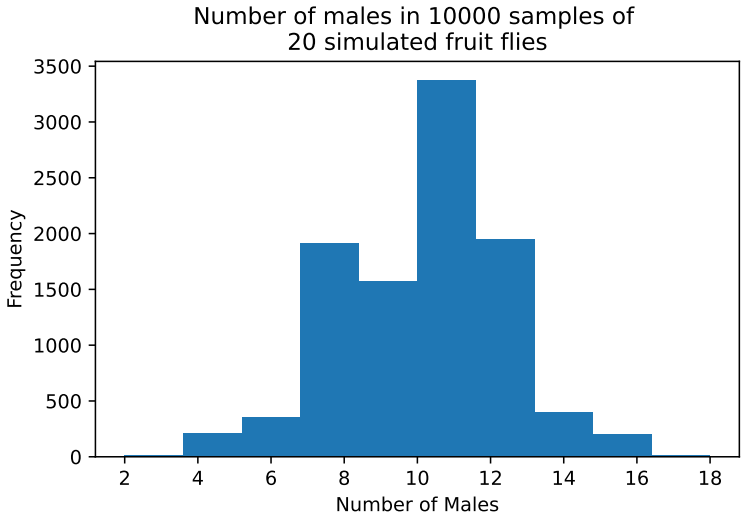
results together we have the probability that the results obtained from the sample of irradiated fruit flies would be obtained from a random sample of fruit flies.

> **Note 40: Notebook: Fruit fly simulation**
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

# set up the random number generator
rnd = np.random.default_rng()
```

```
# Set the number of trials
n_trials = 10000

# set the sample size for each trial
sample_size = 20

# An empty array to store the trials
scores = np.zeros(n_trials)

# Do 10000 trials
for i in range(n_trials):

    # Generate 20 simulated fruit flies, where each has an equal chance of
    # being male or female
    a = rnd.choice(['male', 'female'],
                   size=sample_size,
                   p=[0.5, 0.5],
                   replace = True)

    # count the number of males in the sample
    b = np.sum(a == 'male')

    # store the result of this trial
    scores[i] = b

# Produce a histogram of the trial results
plt.title(f"Number of males in {n_trials} samples of \n{sample_size} simulated fruit flies")
plt.hist(scores)
```

```
plt.xlabel('Number of Males')
plt.ylabel('Frequency')
```



In the histogram above, we see that in about 12 percent of the trials, the number of males was 14 or more, or 6 or fewer. Or instead of reading the results from the histogram, we can calculate the result by tacking on the following commands to the above program:

```
# Determine the number of trials in which we had 14 or more males.
j = np.sum(scores >= 14)

# Determine the number of trials in which we had 6 or fewer males.
k = np.sum(scores <= 6)

# Add the two results together.
m = j + k

# Convert to a proportion.
mm = m / n_trials

# Print the results.
print(mm)
```

```
0.1191
```

> **i** End of notebook: Fruit fly simulation
>
> `fruit_fly` starts at Note 40.

Notice that the strength of the evidence for the effectiveness of the radiation treatment depends upon the original question: whether or not the treatment had *any* effect on the sex of the fruit fly, which is a two-tailed question. If there were reason to believe at the start that the treatment could increase *only* the number of *males*, then we would focus our attention on the result that in only two of the twenty-five trials were fourteen or more males. There would then be only a $2/25 = 0.08$ probability of getting the observed results by chance if the treatment really has no effect, rather than the weaker odds against obtaining fourteen or more of *either* males or females.

Therefore, whether you decide to figure the odds of just fourteen or more males (what is called a "one-tail test") or the odds for fourteen or more males *plus* fourteen or more females (a "two-tail test"), depends upon your advance knowledge of the subject. If you have no reason to believe that the treatment will have an effect *only* in the direction of creating more males and if you figure the odds for the one-tail test anyway, then you will be kidding yourself. Theory comes to bear here. If you have a strong hypothesis, deduced from a strong theory, that there will be more males, then you should figure one-tail odds, but if you have no such theory you should figure the weaker two-tail odds.[1]

In the case of the next problem concerning calves, we shall see that a one-tail test is appropriate because we have no interest in producing more male calves. Before leaving this example, let us review our intellectual strategy in handling the problem. First we observe a result (14 males in 20 flies) which differs from the proportion of the benchmark population (50 percent males). Because we have treated this sample with irradiation and observed a result that differs from the untreated benchmark-population's mean, we speculate that the irradiation caused the sample to differ from the untreated population. We wish to check on whether this speculation is correct.

When asking whether this speculation is correct, we are implicitly asking whether future irradiation would also produce a proportion of males higher than 50 percent. That is, we are implicitly asking whether irradiated flies would produce more samples with male proportions as high as 14/20 than would occur by chance in the absence of irradiation.

If samples as far away as 14/20 from the benchmark population mean of 10/20 would occur frequently by chance, then we would not be impressed with that experimental evidence as proof that irradiation does affect the sex ratio. Hence we set up a model that will tell us the frequency with which samples of 14 or more males out of 20 births would be observed by chance. Carrying out the resampling procedure tells us that perhaps a tenth of the time such samples would be observed by chance. That is not extremely frequent, but it is not infrequent

[1] If you are very knowledgeable, you may do some in-between figuring (with what is known as "Bayesian analysis"), but leave this alone unless you know well what you are doing.

either. Hence we would probably conclude that the evidence is provocative enough to justify further experimentation, but not so strong that we should immediately believe in the truth of this speculation.

The logic of attaching meaning to the probabilistic outcome of a test of a hypothesis is discussed in Chapter 22. There also is more about the concept of the level of significance in Chapter 22.

Because of the great importance of this sort of case, which brings out the basic principles particularly clearly, let us consider another example:

21.2.2 Example: Does a treatment increase the female calf rate?

What is the probability that among 10 calves born, 9 or more will be female?

Let's consider this question in the context of a set of queries for performing statistical inference that will be discussed further in Chapter 25.

The question: (From Hodges Jr and Lehmann (1970)): Female calves are more valuable than males. A bio-engineer claims to be able to cause more females to be born than the expected 50 percent rate. He conducts his procedure, and nine females are born out of the next 10 pregnancies among the treated cows. Should you believe his claim? That is, what is the probability of a result this (or more) surprising occurring by chance if his procedure has no effect? In this problem, we assume that on average 100 of 206 births are female, in contrast to the 50-50 benchmark universe in the previous problem.

What is the purpose of the work?: Female calves are more valuable than male calves.

Statistical inference?: Yes.

Confidence interval or Test of hypothesis?: Test of hypothesis.

Will you state the costs and benefits of various outcomes, or a loss function?: Yes. One need only say that the benefits are very large, and if the results are promising, it is worth gathering more data to confirm results.

How many samples of data are part of the hypothesis test?: One.

What is the size of the first sample about which you wish to make significance statements?: Ten.

What comparison(s) to make?: Compare the sample to the benchmark universe.

***What is the benchmark universe*: that embodies the null hypothesis?** 100/206 female.

Which symbols for the observed entities?: Balls in bucket, or numbers.

What values or ranges of values?: We could write numbers 1 through 206 on pieces of paper, and take numbers 1-100 as "male" and 101-206 as "female". Or we could use some other mechanism to give us a 100/206 chance of any one calf being female.

Finite or infinite universe?: Infinite.

Which sample(s) do you wish to compare to which, or to the null universe (and perhaps to the alternative universe)?: Ten calves.

What procedure to produce the sample entities?: Sampling with replacement.

Simple (single step) or complex (multiple "if" drawings)?: Can think of it either way.

What to record as the outcome of each resample trial?: The proportion (or number) of females.

What is the criterion to be used in the test?: The probability that in a sample of ten calves, nine (or more) females would be drawn by chance from the benchmark universe of 100/206 females.

"One tail" or "two tail" test?: One tail, because the farmer is only interested in females. Finding a large proportion of males would not be of interest; it would not cause rejecting the null hypothesis.

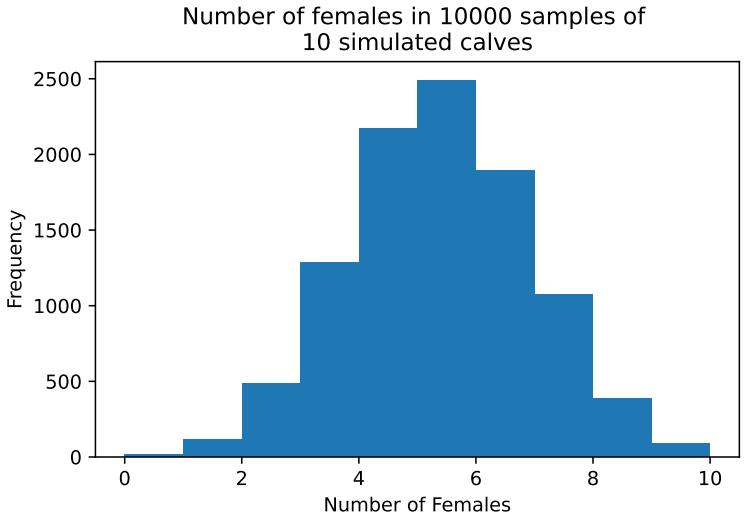
The actual computation of probability may be done in several ways, as discussed earlier for four children and for ten cows. Conventional methods are discussed for comparison in Chapter 25. Here is the resampling solution in Python.

> **Note 41: Notebook: Female calf numbers simulation**
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Set the number of trials.
n_trials = 10000

# Set the size of each sample.
sample_size = 10

# Probability of any one calf being female.
```

```
p_female = 100 / 206

# An array to store the results.
scores = np.zeros(n_trials)

# For 10000 repeats.
for i in range(n_trials):

    a = rnd.choice(['female', 'male'],
                   p=[p_female, 1 - p_female],
                   size=sample_size)
    b = np.sum(a == 'female')

    # Store the result of the current trial.
    scores[i] = b

# Plot a histogram of the scores.
plt.title(f"Number of females in {n_trials} samples of \n{sample_size} simulated calves")
plt.hist(scores)
plt.xlabel('Number of Females')
plt.ylabel('Frequency')

# Count the number of scores that were greater than or equal to 9.
k = np.sum(scores >= 9)

# Express as a proportion.
kk = k / n_trials

# Show the proportion.
print("Probability of 9 or 10 females occurring by chance:", kk)
```

```
Probability of 9 or 10 females occurring by chance: 0.009
```



We read from the result in variable `kk` that the probability of 9 or 10 females occurring by chance is a bit more than one percent.

> End of notebook: Female calf numbers simulation
>
> `female_calves` starts at Note 41.

21.2.3 Example: A public-opinion poll

Is the Proportion of a Population Greater Than a Given Value?

A municipal official wants to determine whether a majority of the town's residents are for or against the awarding of a high-speed internet contract, and she asks you to take a poll. You judge that the voter registration records are a fair representation of the universe in which the politician was interested, and you therefore decided to interview a random selection of registered voters. Of a sample of fifty people who expressed opinions, thirty said "yes" they were for the plan and twenty said "no," they were against it. How conclusively do the results show that the people in town want this internet contract?

Now comes some necessary subtle thinking in the interpretation of what seems like a simple problem. Notice that our aim in the analysis is to avoid the mistake of saying that the town favors the plan when in fact it does not favor the plan. Our chance of making this mistake is greatest when the voters are evenly split, so we choose as the benchmark (null) hypothesis that 50 percent of the town does not want the plan. This statement really means that "50 percent or more do not want the plan." We could assess the probability of obtaining our result from a population that is split (say) 52-48 against, but such a probability would necessarily be even smaller, and we are primarily interested in assessing the maximum probability of being

wrong. If the maximum probability of error turns out to be inconsequential, then we need not worry about less likely errors.

This problem is very much like the one-group fruit fly irradiation problem above. The only difference is that now we are comparing the observed sample against an arbitrary value of 50 percent (because that is the break-point in a situation where the majority decides) whereas in Section 21.2.1 we compared the observed sample against the normal population proportion (also 50 percent, because that is the normal proportion of males). But it really does not matter *why* we are comparing the observed sample to the figure of 50 percent; the procedure is the same in both cases. (Please notice that there is nothing special about the 50 percent figure; the same procedure would be followed for 20 percent or 85 percent.)

In brief, we a) take two pieces of paper, write "Yes" on one and "No" on the other, put them in a bucket b) draw a piece of paper from the bucket, record whether it was "Yes" or "No", replace, and repeat 50 times c) count the number of "yeses" and "noes" in the fifty draws, c) repeat for perhaps a hundred trials, then d) count the proportion of the trials in which a 50-50 universe would produce thirty or more "yes" answers.

In operational steps, the procedure is as follows:

- **Step 1.** "1-5" = no, "6-0" = yes.
- **Step 2.** In 50 random numbers, count the "yeses," and record "false positive" if 30 or more "yeses."
- **Step 3.** Repeat step 2 perhaps 100 times.
- **Step 4.** Calculate the proportion of experimental trials showing "false positive." This estimates the probability that as many as 30 "yeses" would be observed by chance in a sample of 50 people *if* half (or more) are really against the plan.

Table 21.2: Results from 20 random trials for contract poll problem

Trial no	# of "Noes"	# of "Yeses"	>= 30 "Yeses"
1	25	25	
2	23	27	
3	18	32	+
4	33	17	
5	32	18	
6	23	27	
7	28	22	
8	28	22	
9	22	28	
10	30	20	
11	22	28	
12	28	22	
13	26	24	

Trial no	# of “Noes”	# of “Yeses”	>= 30 “Yeses”
14	20	30	+
15	27	23	
16	19	31	+
17	26	24	
18	27	23	
19	20	30	+
20	24	26	

In Table 21.2, we see the results of twenty trials; 4 of 20 times (20 percent), 30 or more “yeses” were observed by chance. So our “significance level” or “probability value” is 20 percent, which is normally too high to feel confident that our poll results are reliable. This is the probability that as many as thirty of fifty people would say “yes” by chance if the population were “really” split evenly. (If the population were split so that *more* than 50 percent were against the plan, the probability would be even *less* that the observed results would occur by chance. In this sense, the benchmark hypothesis is conservative). On the other hand, if we had been counting the number of times there are 30 or more “No” votes that, in our setup, have the same odds as to 30 or more “Yes” votes, there would have been three. This indicates how samples can vary just by chance.

Taken together, the evidence suggests that the official would be wise not to place very much confidence in the poll results, but rather ought to act with caution or else take a larger sample of voters.

> **Note 42: Notebook: Contract poll simulation**
>
> - Download notebook
> - Interact

This Python notebook generates samples of 50 simulated voters on the assumption that only 50 percent are in favor of the contract. Then it counts (`sums`) the number of samples where over 29 (30 or more) of the 50 respondents said they were in favor of the contract. (That is, we use a “one-tailed test.”) The result in the `kk` variable is the chance of a “false positive,” that is, 30 or more people saying they favor a contract when support for the proposal is actually split evenly down the middle.

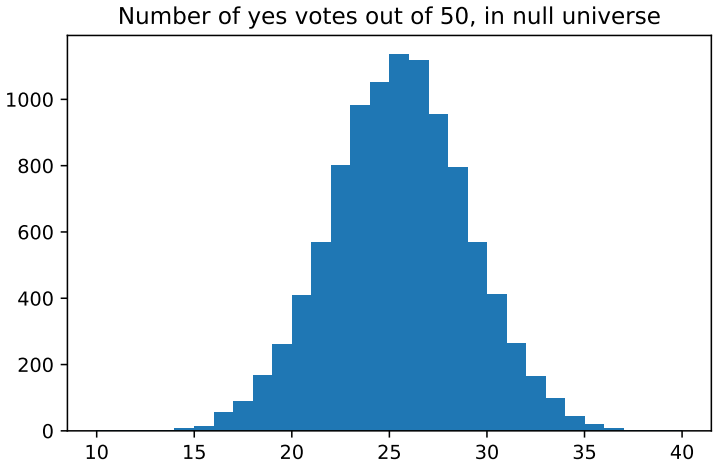
```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()
```

```
# We will do 10,000 iterations.
n = 10_000
```

> **Note 43: Underscores in Python integers**
>
> Notice the line above: `n = 10_000`. This is an alternative way of writing the familiar `n = 10000`. Python allows underscores among the digits when we type an integer — it will treat them as decoration, and ignore them in resolving the number we intend. This means that all these are equivalent to Python:
>
> ```
> 10000
> ```
>
> ```
> 10000
> ```
>
> ```
> 10_000
> ```
>
> ```
> 10000
> ```
>
> or even:
>
> ```
> 1_00_00
> ```
>
> ```
> 10000
> ```
>
> For the same reason, we could also write 1000 as:
>
> ```
> 1_000
> ```
>
> ```
> 1000
> ```
>
> The underscores make no difference to the resulting number in Python; it is up to us whether we use them. They can make it easier for us humans to read the value.

```
# Make an array of integers to store the "Yes" counts.
yeses = np.zeros(n, dtype=int)

for i in range(n):
    answers = rnd.choice(['No', 'Yes'], size=50)
    yeses[i] = np.sum(answers == 'Yes')

# Produce a histogram of the trial results.
```

```
# Use integer bins for histogram, from 10 through 40.
plt.hist(yeses, bins=range(10, 41))
plt.title('Number of yes votes out of 50, in null universe')
```



In the histogram above, we see that about 11 percent of our trials had 30 or more voters in favor, despite the fact that they were drawn from a population that was split 50-50. Python will calculate this proportion directly if we add the following commands to the above:

```
k = np.sum(yeses >= 30)
kk = k / n
print('Proportion >= 30:', np.round(kk, 2))
```

```
Proportion >= 30: 0.1
```

> **i** End of notebook: Contract poll simulation
>
> `contract_poll` starts at Note 42.

The section above discusses testing hypotheses about a single sample of counted data relative to a benchmark universe. This section discusses the issue of whether *two* samples with counted data should be considered the same or different.

21.2.4 Example: Did the Trump-Clinton poll indicate that Trump would win?

> **Note 44: Notebook: Trump/Clinton poll simulation**
>
> - Download notebook
> - Interact

What is the probability that a sample outcome such as actually observed (840 Trump, 660 Clinton) would occur by chance if Clinton is "really" ahead — that is, if Clinton has 50 percent (or more) of the support? To restate in sharper statistical language: What is the probability that the observed sample or one even more favorable to Trump would occur if the universe has a mean of 50 percent or below?

Here is a procedure that responds to that question:

1. Create a benchmark universe with one ball marked "Trump" and another marked "Clinton"
2. Draw a ball, record its marking, and replace. (We sample with replacement to simulate the practically-infinite population of U. S. voters.)
3. Repeat step 2 1500 times and count the number of "Trump"s. If 840 or greater, record "Y"; otherwise, record "N."
4. Repeat steps 3 and 4 perhaps 1000 or 10,000 times, and count the number of "Y"s. The outcome estimates the probability that 840 or more Trump choices would occur if the universe is "really" half or more in favor of Clinton.

Before we come to the simulation, we need some new code to tune our histograms (see Section 12.15.2). We are going to set the bins for the histogram using advanced ranges.

> **Note 45: Advanced ranges**
>
> So far (Section 5.9) we have used `np.arange` to make regular sequences of integers. For example, to make an array of the sequential integers from 3 through 12, we could use:
>
> ```
> np.arange(3, 13)
> ```
>
> ```
> array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
> ```
>
> Sometimes we want to be able to specify a step size — the gap between the numbers in the sequence. In the sequence above, the gap (step) between each number is 1. We might want some other step size. To create a sequence of integers from 3 through 33 in steps of 5, we could write:
>
> ```
> np.arange(3, 34, step=5)
> ```

```
array([ 3,  8, 13, 18, 23, 28, 33])
```

Read this as "give me the sequence (range) of numbers, starting at 3, up to but not including 34, in steps of 5.
So far we have used integers as the start, stop and step values, but we could also use floating point values. For example, to get a sequence of values starting at 0.1 up to and including 0.9, in steps of 0.2:

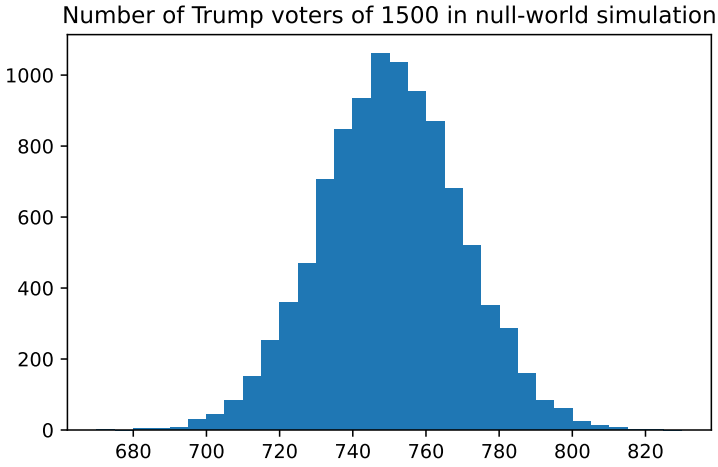
```
np.arange(0.1, 1, step=0.2)
```

```
array([0.1, 0.3, 0.5, 0.7, 0.9])
```

With that background, we can proceed with the Python implementation of the simulation procedure.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Number of repeats we will run.
n = 10_000

# Make an integer array to store the counts.
trumps = np.zeros(n, dtype=int)

for i in range(n):
    votes = rnd.choice(['Trump', 'Clinton'], size=1500)
    trumps[i] = np.sum(votes == 'Trump')

# Integer bins from 670 through 830 in steps of 5.
plt.hist(trumps, bins=range(670, 831, 5))
plt.title('Number of Trump voters of 1500 in null-world simulation')

# How often >= 840 Trump votes in random draw?
k = np.sum(trumps >= 840)
# As a proportion of simulated resamples.
kk = k / n

print('Proportion voting for Trump:', kk)
```

```
Proportion voting for Trump: 0.0
```



The value for `kk` is our estimate of the probability that Trump's "victory" in the sample would occur by chance if he really were behind. In this case, our probability estimate is less than 1 in 10,000 (< 0.0001).

> **i** End of notebook: Trump/Clinton poll simulation
>
> `trump_clinton` starts at Note 44.

21.2.5 Example: Comparison of possible cancer cure to placebo

Do Two Binomial Populations Differ in Their Proportions.

Section 21.2.1 used an observed sample of male and female fruitflies to test the benchmark (null) hypothesis that the flies came from a universe with a one-to-one sex ratio, and the poll data problem also compared results to a 50-50 hypothesis. The calves problem also compared the results to a single benchmark universe — a proportion of 100/206 females. Now we want to compare *two samples with each other*, rather than comparing one sample with a hypothesized universe. That is, in this example we are not comparing one sample to a benchmark universe, but rather asking whether *both* samples come from the *same* universe. The universe from which both samples come, *if* both belong to the same universe, may be thought of as the benchmark universe, in this case.

The scientific question is whether pill P cures a rare cancer. A researcher gave pill P to six patients selected randomly from a group of twelve cancer patients; of the six, five got well. He gave an inactive placebo to the other six patients, and two of them got well. Does the evidence justify a conclusion that the pill has a curative effect?

(An identical statistical example would serve for an experiment on methods of teaching reading to children. In such a situation the researcher would respond to inconclusive results by running the experiment on more subjects, but in cases like the cancer-pill example the researcher often cannot obtain more subjects.)

We can answer the stated question by *combining* the two samples and testing both samples against the resulting combined universe. In this case, the universe is twelve subjects, seven (5 + 2) of whom got well. How likely would such a universe produce two samples as far apart as five of six, and two of six, patients who get well? In other words, how often will two samples of six subjects, each drawn from a universe in which 7/12 of the patients get well, be as far apart as 5 - 2 = 3 patients in favor of the sample designated "pill"? This is obviously a one-tail test, for we have no reason to believe that the pill group might do *less* well than the placebo group.

We might construct a twelve-sided die, seven of whose sides are marked "get well." Or put 12 pieces of paper in a bucket, seven with "get well" and five with "not well". Or we would use pairs of numbers from the random-number table, with numbers "01-07" corresponding to get well, numbers "08-12" corresponding to "not get well," and all other numbers omitted. (If you wish to save time, you can work out a system that uses more numbers and skips fewer, but that is up to you.) Designate the first six subjects "pill" and the next six subjects "placebo."

The specific procedure might be as follows:

- **Step 1.** Write "get well" on seven pieces of paper, "not well" on another five. Put the 12 pieces of paper into a bucket.
- **Step 2.** Select two groups, "pill" and "placebo", each with six random draws (with replacement) from the 12 pieces of paper.
- **Step 3.** Record how many "get well" in each group.
- **Step 4.** Subtract the result in group "placebo" from that in group "pill" (the difference may be negative).
- **Step 5.** Repeat steps 1-4 perhaps 100 times.
- **Step 6.** Compute the proportion of trials in which the pill does better by three or more cases.

Table 21.3: Results from 25 random trials for pill/placebo

Trial no	# of pill cures	# of placebo cures	Difference
1	2	3	-1
2	4	3	1
3	5	2	3
4	3	3	0
5	5	2	3
6	4	4	0
7	3	3	0

Trial no	# of pill cures	# of placebo cures	Difference
8	3	3	0
9	3	3	0
10	4	5	-1
11	4	5	-1
12	3	4	-1
13	0	3	-3
14	5	4	1
15	3	3	0
16	5	3	2
17	5	1	4
18	3	4	-1
19	4	2	2
20	2	4	-2
21	2	6	-4
22	5	5	0
23	4	5	-1
24	3	3	0
25	4	5	-1

In the trials shown in Table 21.3, in three cases (12 percent) the difference between the randomly-drawn groups is three cases or greater. Apparently it is *somewhat* unusual — it happens 12 percent of the time — for this universe to generate "pill" samples in which the number of recoveries exceeds the number in the "placebo" samples by three or more. Therefore the answer to the scientific question, based on these samples, is that there is *some* reason to think that the medicine does have a favorable effect. But the investigator might sensibly await more data before reaching a firm conclusion about the pill's efficiency, given the 12 percent probability.

> **Note 46: Notebook: Cures for pill vs placebo**
>
> - Download notebook
> - Interact

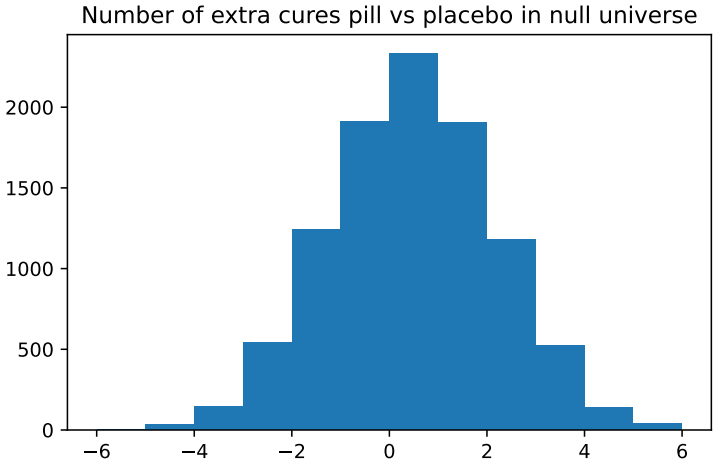
Now for a Python solution. Again, the benchmark hypothesis is that pill P has no effect, and we ask how often, on this assumption, the results that were obtained from the actual test of the pill would occur by chance.

Given that in the test 7 of 12 patients overall got well, the benchmark hypothesis assumes 7/12 to be the chances of any random patient being cured. We generate two similar samples of 6 patients, both taken from the same universe composed of the combined samples — the bootstrap procedure. We count (`sum`) the number who are "get well" in each sample. Then we

subtract the number who got well in the "pill" sample from the number who got well in the "no-pill" sample. We record the resulting difference for each trial in the variable `pill_betters`.

In the actual test, 3 more patients got well in the sample given the pill than in the sample given the placebo. We therefore count how many of the trials yield results where the difference between the sample given the pill and the sample not given the pill was greater than 2 (equal to or greater than 3). This result is the probability that the results derived from the actual test would be obtained from random samples drawn from a population which has a constant cure rate, pill or no pill.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# The bucket with the pieces of paper.
options = np.repeat(['get well', 'not well'], [7, 5])

n = 10_000

pill_betters = np.zeros(n, dtype=int)

for i in range(n):
    pill = rnd.choice(options, size=6)
    pill_cures = np.sum(pill == 'get well')
    placebo = rnd.choice(options, size=6)
    placebo_cures = np.sum(placebo == 'get well')
    pill_betters[i] = pill_cures - placebo_cures

plt.hist(pill_betters, bins=range(-6, 7))
plt.title('Number of extra cures pill vs placebo in null universe')
```



Recall our actual observed results: In the medicine group, three more patients were cured than in the placebo group. From the histogram, we see that in only about 8 percent of the simulated trials did the "medicine" group do as well or better. The results seem to suggest — but by no means conclusively — that the medicine's performance is not due to chance. Further study would probably be warranted. The following commands added to the above program will calculate this proportion directly:

```
# How many trials gave an advantage of 3 or greater to the pill?
k = np.sum(pill_betters >= 3)
# Convert to a proportion.
kk = k / n
# Print the result.
print('Proportion with advantage of 3 or more for pill:',
      np.round(kk, 2))
```

```
Proportion with advantage of 3 or more for pill: 0.07
```

> End of notebook: Cures for pill vs placebo
>
> `pill_placebo` starts at Note 46.

As I (JLS) wrote when I first proposed this bootstrap method in 1969, this method is not the standard way of handling the problem; it is not even analogous to the standard analytic difference-of-proportions method (though since then it has become widely accepted). Though the method shown is quite direct and satisfactory, there are also *many other* resampling methods that one might construct to solve the same problem. By all means, invent your own

statistics rather than simply trying to copy the methods described here; the examples given here only illustrate the process of inventing statistics rather than offering solutions for all classes of problems.

21.2.6 Example: Did attitudes about marijuana change?

Consider two polls, each asking 1500 Americans about marijuana legalization. One poll, taken in 1980, found 52 percent of respondents in favor of decriminalization; the other, taken in 1985, found 46 percent in favor of decriminalization (Wonnacott and Wonnacott 1990, 275). Our null (benchmark) hypothesis is that both samples came from the same universe (the universe made up of the total of the two sets of observations). If so, let us then ask how likely would be two polls to produce results as different as were observed? Hence we construct a universe with a mean of 49 percent (the mean of the two polls of 52 percent and 46 percent), and repeatedly draw pairs of samples of size 1500 from it.

To see how the construction of the appropriate question is much more challenging intellectually than is the actual mathematics, let us consider another possibility suggested by a student: What about considering the universe to be the earlier poll with a mean of 52 percent, and then asking the probability that the later poll of 1500 people with a mean of 46 percent would come from it? Indeed, on first thought that procedure seems reasonable.

Upon reflection — and it takes considerable thought on these matters to get them right — that would *not* be an appropriate procedure. The student's suggested procedure would be the same as assuming that we had long-run solid knowledge of the universe, as if based on millions of observations, and then asking about the probability of a particular sample drawn from it. That does not correspond to the facts.

The only way to find the approach you eventually consider best — and there is no guarantee that it is indeed *correct* — is by close reference to the particular facts of the case.

21.2.7 Example: Infarction and cholesterol: Framingham study

It is so important to understand the logic of hypothesis tests, and of the resampling method of doing them, that we will now tackle another problem similar to the preceding one.

This will be the first of several problems that use data from the famous Framingham study (drawn from Kahn and Sempos (1989)) concerning the development of myocardial infarction 16 years after the Framingham study began, for men ages 35- 44 with serum cholesterol above 250, compared to those with serum cholesterol below 250. The raw data are shown in Table 21.4. The data are from (Shurtleff 1970), cited in (Kahn and Sempos 1989, 12:61, Table 3-8). Kahn and Sempos divided the cases into "high" and "low" cholesterol.

Table 21.4: Development of Myocardial Infarction in Men Aged 35-44 After 16 Years

Serum Cholesterol	Developed MI	Didn't Develop MI	Total
> 250	10	125	135
<= 250	21	449	470

The statistical logic properly begins by asking: How likely is that the two observed groups "really" came from the same "population" with respect to infarction rates? That is, we start with this question: How sure should one be that there is a difference in myocardial infarction rates between the high and low-cholesterol groups? Operationally, we address this issue by asking how likely it is that two groups as different in disease rates as the observed groups would be produced by the same "statistical universe."

Key step: We assume that the relevant "benchmark" or "null hypothesis" population (universe) is the composite of the two observed groups. That is, if there were no "true" difference in infarction rates between the two serum-cholesterol groups, and the observed disease differences occurred just because of sampling variation, the most reasonable representation of the population from which they came is the composite of the two observed groups.

Therefore, we compose a hypothetical "benchmark" universe containing (135 + 470 =) 605 men at risk, and designate (10 + 21 =) 31 of them as infarction cases. We want to determine how likely it is that a universe like this one would produce — just by chance — two groups that differ as much as do the actually observed groups. That is, how often would random sampling from this universe produce one sub-sample of 135 men containing a large enough number of infarctions, and the other sub-sample of 470 men producing few enough infarctions, that the difference in occurrence rates would be as high as the observed difference of .029? (10/135 = .074, and 21/470 = .045, and .074 - .045 = .029).

So far, everything that has been said applies both to the conventional formulaic method and to the "new statistics" resampling method. But the logic is seldom explained to the reader of a piece of research — if indeed the researcher her/ himself grasps what the formula is doing. And if one just grabs for a formula with a prayer that it is the right one, one need never analyze the statistical logic of the problem at hand.

Now we tackle this problem with a method that you would think of yourself if you began with the following mind-set: How can I simulate the mechanism whose operation I wish to understand? These steps will do the job:

- **Step 1:** Fill a bucket with 605 balls, 31 red (infarction) and the rest (605 — 31 = 574) green (no infarction).
- **Step 2:** Draw a sample of 135 (simulating the high serum-cholesterol group), one ball at a time and throwing it back after it is drawn to keep the simulated probability of an infarction the same throughout the sample; record the number of reds. Then do the same with another sample of 470 (the low serum-cholesterol group).

- **Step 3:** Calculate the difference in infarction rates for the two simulated groups, and compare it to the actual difference of .029; if the simulated difference is that large, record "Yes" for this trial; if not, record "No."
- **Step 4:** Repeat steps 2 and 3 until a total of (say) 400 or 1000 trials have been completed. Compute the frequency with which the simulated groups produce a difference as great as actually observed. This frequency is an estimate of the probability that a difference as great as actually observed in Framingham would occur even if serum cholesterol has no effect upon myocardial infarction.

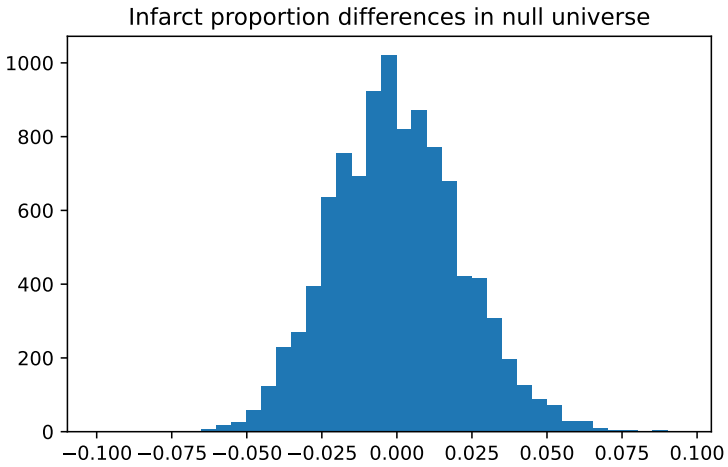
The procedure above can be carried out with balls in a bucket in a few hours. Yet it is natural to seek the added convenience of the computer to draw the samples. Here is the Python program:

> Note 47: Notebook: Framingham heart data
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

n = 10_000
men = np.repeat(['infarction', 'no infarction'], [31, 574])

n_high = 135  # Number of men with high cholesterol
n_low = 470  # Number of men with low cholesterol

infarct_differences = np.zeros(n)

for i in range(n):
    highs = rnd.choice(men, size=n_high)
    lows = rnd.choice(men, size=n_low)
    high_infarcts = np.sum(highs == 'infarction')
    low_infarcts = np.sum(lows == 'infarction')
    high_prop = high_infarcts / n_high
    low_prop = low_infarcts / n_low
    infarct_differences[i] = high_prop - low_prop

# Set the histogram bin edges to the sequence starting at -0.1, up to (not
# including) 0.1, in steps of 0.005.
plt.hist(infarct_differences, bins=np.arange(-0.1, 0.1, 0.005))
```

```
plt.title('Infarct proportion differences in null universe')

# How often was the resampled difference >= the observed difference?
k = np.sum(infarct_differences >= 0.029)
# Convert this result to a proportion
kk = k / n

print('Proportion of trials with difference >= observed:',
      np.round(kk, 2))
```

```
Proportion of trials with difference >= observed: 0.09
```



The results of the test using this program may be seen in the histogram. We find — perhaps surprisingly — that a difference as large as observed would occur by chance around 10 percent of the time. (If we were not guided by the theoretical expectation that high serum cholesterol produces heart disease, we might include the 10 percent difference going in the other direction, giving a 20 percent chance). Even a ten percent chance is sufficient to call into question the conclusion that high serum cholesterol is dangerous. At a minimum, this statistical result should call for more research before taking any strong action clinically or otherwise.

> ℹ End of notebook: Framingham heart data
>
> `framingham_hearts` starts at Note 47.

Where should one look to determine which procedures should be used to deal with a problem such as set forth above? Unlike the formulaic approach, the basic source is not a manual

which sets forth a menu of formulas together with sets of rules about when they are appropriate. Rather, you consult your own understanding about what is happening in (say) the Framingham situation, and the question that needs to be answered, and then you construct a "model" that is as faithful to the facts as is possible. The bucket-sampling described above is such a model for the case at hand.

To connect up what we have done with the conventional approach, one could apply a z test (conceptually similar to the t test, but applicable to yes-no data; it is the Normal-distribution approximation to the large binomial distribution). Do so, we find that the results are much the same as the resampling result — an eleven percent probability.

Someone may ask: Why do a resampling test when you can use a standard device such as a z or t test? The great advantage of resampling is that it avoids using the wrong method. The researcher is more likely to arrive at sound conclusions with resampling because s/he can understand what s/he is doing, instead of blindly grabbing a formula which may be in error.

The textbook from which the problem is drawn is an excellent one; the difficulty of its presentation is an inescapable consequence of the formulaic approach to probability and statistics. The body of complex algebra and tables that only a rare expert understands down to the foundations constitutes an impenetrable wall to understanding. Yet without such understanding, there can be only rote practice, which leads to frustration and error.

21.2.8 Example: Is one pig ration more effective than the other?

Testing For a Difference in Means With a Two-by-Two Classification.

Each of two new types of ration is fed to twelve pigs. A farmer wants to know whether ration A or ration B is better.[2] The weight gains in pounds for pigs fed on rations A and B are:

A: 31, 34, 29, 26, 32, 35, 38, 34, 31, 29, 32, 31

B: 26, 24, 28, 29, 30, 29, 31, 29, 32, 26, 28, 32

The statistical question may be framed as follows: should one consider that the pigs fed on the different rations come from the same universe with respect to weight gains?

In the actual experiment, 9 of the 12 pigs who were fed ration A also were in the top half of weight gains. How likely is it that one group of 12 randomly-chosen pigs would contain 9 of the 12 top weight gainers?

One approach to the problem is to divide the pigs into two groups — the twelve with the highest weight gains, and the twelve with the lowest weight gains — and examine whether an unusually large number of high-weight-gain pigs were fed on one or the other of the rations.

We can make this test by ordering and grouping the twenty four pigs:

[2]The data for this example are based on Dixon and Massey (1983, 124), who offer an orthodox method of handling the problem with a t-test.

- **High-weight group:** 38 (ration A), 35 (A), 34 (A), 34 (A), 32 (B), 32 (A), 32 (A), 32 (B), 31 (A), 31 (B), 31 (A), 31 (A)
- **Low-weight group:** 30 (B), 29 (A), 29 (A), 29 (B), 29 (B), 29 (B), 28 (B), 28 (B), 26 (A), 26 (B), 26 (B), 24 (B).

Among the twelve high-weight-gain pigs, nine were fed on ration A. We ask: Is this further from an even split than we are likely to get by chance? Let us take twelve red and twelve black cards, shuffle them, and deal out twelve cards (the other twelve need not be dealt out). Count the proportion of the hands in which one ration comes up nine or more times in the first twelve cards, to reflect ration A's appearance nine times among the highest twelve weight gains. More specifically:

- **Step 1.** Constitute a deck of twelve red and twelve black cards, and shuffle.
- **Step 2.** Deal out twelve cards, count the number red, and record "yes" if there are nine or more of *either* red or black.
- **Step 3.** Repeat step 2 perhaps fifty times.
- **Step 4.** Compute the proportion "yes." This proportion estimates the probability sought.

Table 21.5: Results from 25 random trials for pig rations

Trial no	# red	# black	>=9 red or black
1	6	6	
2	9	3	+
3	9	3	+
4	7	5	
5	7	5	
6	8	4	
7	5	7	
8	8	4	
9	7	5	
10	10	2	+
11	6	6	
12	7	5	
13	7	5	
14	5	7	
15	7	5	
16	5	7	
17	7	5	
18	4	8	
19	6	6	
20	9	3	+
21	6	6	

Trial no	# red	# black	>=9 red or black
22	8	4	
23	7	5	
24	7	5	
25	9	3	+

Table 21.5 shows the results of 25 trials. In five (marked by + signs) of the 25 (that is, 20 percent of the trials) there were nine or more either red or black cards in the first twelve cards. Again the results suggest that it would be *slightly unusual* for the results to favor one ration or the other so strongly just by chance if they come from the same universe.

Now the Python procedure to answer the question:

> **Note 48: Notebook: Weight gain on pig rations**
>
> - Download notebook
> - Interact

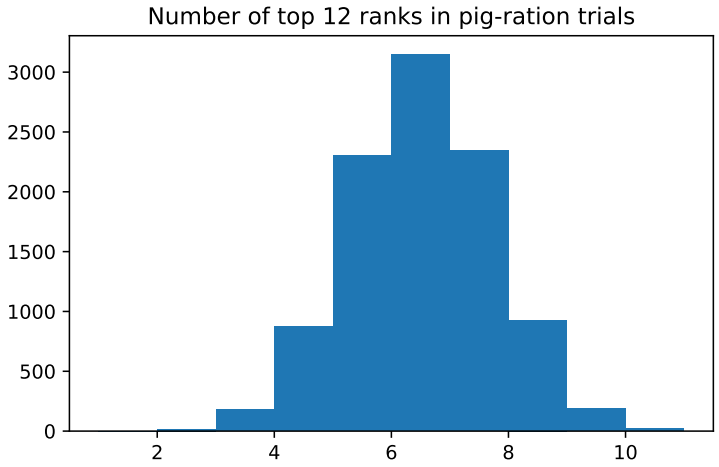
The `ranks = np.arange(1, 25)` statement creates an array of numbers 1 through 24, which will represent the rankings of weight gains for each of the 24 pigs. We repeat the following procedure for 10000 trials. First we shuffle the elements of array `ranks` so that the rank numbers for weight gains are randomized and placed in array `shuffled`. We then select the first 12 elements of `shuffled` and place them in `first_12`; this represents the rankings of a randomly-selected group of 12 pigs. We next count (`sum`) in `n_top` the number of pigs whose rankings for weight gain were in the top half — that is, a rank of less than 13. We record that number in `top_ranks`, and then continue the loop, until we finish our `n` trials.

Since we did not know beforehand the direction of the effect of ration A on weight gain, we want to count the times that *either more than 8* of the random selection of 12 pigs were in the top half of the rankings, *or that fewer than 4* of these pigs were in the top half of the weight gain rankings — (The latter is the same as counting the number of times that more than 8 of the 12 *non-selected* random pigs were in the top half in weight gain.)

We do so with the final two `sum` statements. By adding the two results `n_gte_9` and `n_lte_3` together, we have the number of times out of 10,000 that differences in weight gains in two groups as dramatic as those obtained in the actual experiment would occur by chance.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Constitute the set of the weight gain rank orders. ranks is now a vector
```

```
# consisting of the numbers 1 - 24, in that order.
ranks = np.arange(1, 25)

n = 10_000

top_ranks = np.zeros(n, dtype=int)

for i in range(n):
    # Shuffle the ranks of the weight gains.
    shuffled = rnd.permuted(ranks)
    # Take the first 12 ranks.
    first_12 = shuffled[:12]
    # Determine how many of these randomly selected 12 ranks are less than
    # 12 (i.e. 1-12), put that result in n_top.
    n_top = np.sum(first_12 <= 12)
    # Keep track of each trial result in top_ranks
    top_ranks[i] = n_top

plt.hist(top_ranks, bins=np.arange(1, 12))
plt.title('Number of top 12 ranks in pig-ration trials')
```



We see from the histogram that, in about 3 percent of the trials, either more than 8 or fewer than 4 top half ranks (1-12) made it into the random group of twelve that we selected. Python will calculate this for us as follows:

```
# Determine how many of the trials yielded 9 or more top ranks.
n_gte_9 = np.sum(top_ranks >= 9)
# Determine how many trials yielded 3 or fewer of the top ranks.
# If there were 3 or fewer, then 9 or more of the top ranks must
# have been in the other group (not selected).
n_lte_3 = np.sum(top_ranks <= 3)
# Add the two together.
n_both = n_gte_9 + n_lte_3
# Convert to a proportion.
prop_both = n_both / n

print('Trial proportion >=9 top ranks in either group:',
      np.round(prop_both, 2))
```

```
Trial proportion >=9 top ranks in either group: 0.04
```

The decisions that are warranted on the basis of the results depend upon one's purpose. If writing a scientific paper on the merits of ration A is the ultimate purpose, it would be sensible to test another batch of pigs to get further evidence. (Or you could proceed to employ another sort of test for a slightly more precise evaluation.) But if the goal is a decision on which type of ration to buy for a small farm and they are the same price, just go ahead and buy ration A because, even if it is no better than ration B, you have strong evidence that it is *no worse*.

> **i** End of notebook: Weight gain on pig rations
>
> `pig_rations` starts at Note 48.

21.2.9 Example: Do planet densities differ?

Consider the five planets known to the ancient world.

Mosteller and Rourke (1973, 17–19) ask us to compare the densities of the three planets *farther* from the sun than is the earth (Mars, density 0.71; Jupiter, 0.24; and Saturn, 0.12) against the densities of the planets *closer* to the sun than is the earth (Mercury, 0.68; Venus, 0.94).

The average density of the distant planets is .357, of the closer planets is .81. Is this difference (.353) statistically surprising, or is it likely to occur in a chance ordering of these planets?

We can answer this question with a permutation test; such sampling without replacement makes sense here because we are considering the entire set of planets, rather than a sample drawn from a larger population of planets (the word "population" is used here, rather than "universe," to avoid confusion.) And because the number of objects is so small, one could

examine all possible arrangements (permutations), and see how many have (say) differences in mean densities between the two groups as large as observed.

Another method that Mosteller and Rourke suggest is by a comparison of the density *ranks* of the two sets, where Saturn has rank 1 and Venus has rank 5. This might have a scientific advantage if the sample data are dominated by a single "outlier," whose domination is removed when we rank the data.

We see that the sum of the ranks for the "closer" set is 3+5=8. We can then ask: If the ranks were assigned at random, how likely is it that a set of two planets would have a sum as large as 8? Again, because the sample is small, we can examine all the possible permutations, as Mosteller and Rourke do in their Table 3-1 (Mosteller and Rourke 1973, 56) (Substitute "Closer" for "B," "Further" for "A"). In two of the ten permutations, a sum of ranks as great as 8 is observed, so the probability of a result as great as observed happening by chance is 20 percent, using these data. (We could just as well consider the difference in mean ranks between the two groups: (8/2 - 7/3 = 10 / 6 = 1.67).

To illuminate the logic of this test, consider comparing the heights of two samples of trees. If sample A has the five tallest trees, and sample B has the five shortest trees, the difference in mean ranks will be (6+7+8+9+10=) 40 — (1+2+3+4+5=) 15, the largest possible difference. If the groups are less sharply differentiated — for example, if sample A has #3 and sample B has #8 — the difference in ranks will be less than the maximum of 40, as you can quickly verify.

The method we have just used is called a Mann-Whitney test, though that label is usually applied when the data are too many to examine all the possible permutations; in that case one conventionally uses a table prepared by formula. In the case where there are too many for a complete permutation test, our resampling algorithm is as follows (though we'll continue with the planets example):

1. Compute the mean ranks of the two groups.
2. Calculate the difference between the means computed in step 1.
3. Create a bucket containing the ranks from 1 to the number of observations (5, in the case of the planets)
4. Shuffle the ranks.
5. Since we are working with the ranked data, we must draw without replacement, because there can only be one #3, one #7, and so on. So draw the number of observations corresponding to the number of observations — 2 "Closer" and 3 "Further."
6. Compute the mean ranks of the two simulated groups of planets.
7. Calculate the difference between the means computed in step 5 and record.
8. Repeat steps 4 through 7 perhaps 1000 times.
9. Count how often the shuffled difference in ranks exceeds the observed difference from step 2 (1.67).

Note 49: Notebook: Planet densities and distance

- Download notebook
- Interact

```
import numpy as np

rnd = np.random.default_rng()

# Steps 1 and 2.
actual_mean_diff = 8 / 2 - 7 / 3

# Step 3
ranks = np.arange(1, 6)

n = 10_000

mean_differences = np.zeros(n)

for i in range(n):
    # Step 4
    shuffled = rnd.permuted(ranks)
    # Step 5
    closer = shuffled[:2]  # First 2
    further = shuffled[2:] # Last 3
    # Step 6
    mean_close = np.mean(closer)
    mean_far = np.mean(further)
    # Step 7
    mean_differences[i] = mean_close - mean_far

# Step 9
k = np.sum(mean_differences >= actual_mean_diff)
prob = k / n

print('Proportion of trials with mean difference >= 1.67:',
      np.round(prob, 2))
```

```
Proportion of trials with mean difference >= 1.67: 0.19
```

Interpretation: 19 percent of the time, random shufflings produced a difference in ranks as great as or greater than observed. Hence, on the strength of this evidence, we should *not* conclude

that there is a statistically surprising difference in densities between the further planets and the closer planets.

> End of notebook: Planet densities and distance
>
> `planet_densities` starts at Note 49.

21.3 Conclusion

This chapter has begun the actual work of testing hypotheses. The next chapter continues with discussion of somewhat more complex problems with counted data — more complex to think about, but no more difficult to actually treat mathematically with resampling simulation. If you have understood the general logic of the procedures used up until this point, you are in command of all the necessary conceptual knowledge to construct your own tests to answer any statistical question. A lot more practice, working on a variety of problems, obviously would help. But the key elements are simple: 1) Model the real situation accurately, 2) experiment with the model, and 3) compare the results of the model with the observed results.

22 The Concept of Statistical Significance in Testing Hypotheses

This chapter offers an interpretation of the meaning of the concept of statistical significance and the term "significant" in connection with the logic of significance tests. It also discusses the concept of "level of significance."

22.1 The logic of hypothesis tests

Let's address the logic of hypothesis tests by considering a variety of examples in everyday thinking:

Consider the nine-year-old who tells the teacher that the dog ate the homework. Why does the teacher not accept the child's excuse? Clearly it is because the event would be too "unusual." But why do we think that way?

Let's speculate that you survey a million adults, and only three report that they have ever heard of a real case where a dog ate somebody's homework. You are a teacher, and a student comes in without homework and says that a dog ate the homework. It could have happened — your survey reports that it really has happened in three lifetimes out of a million. But the event happens *only very infrequently*. Therefore, you probably conclude that because the event is so unlikely, something else must have happened — and the likeliest alternative is that the student did not do the homework. The logic is that if an event seems very unlikely, it would therefore surprise us greatly if it were to actually happen, and therefore we assume that there must be a better explanation. This is why we look askance at unlikely coincidences when they are to someone's benefit.

The same line of reasoning was the logic of John Arbuthnot's hypothesis test (1710) about the ratio of births by sex in the first published hypothesis test, though his extension of logic to God's design as an alternative hypothesis goes beyond the standard modern framework. It is also the implicit logic in the research on puerperal fever, cholera, and beri-beri, the data for which were shown in Chapter 17, though no explicit mention was made of probability in those cases.

Two students sat next to each other at an ACT college-entrance examination in Kentucky in 1987. Out of 219 questions, 211 of the answers were identical, including many that were wrong. Student A was a high school athlete in Kentucky who had failed two previous SAT exams, and

Student B thought he saw Student A copying from him. Should one believe that Student A cheated? (*The Washington Post*, April 19, 1992, p. D2.)

You say to yourself: It would be most unlikely that the two test-takers would answer that many questions identically by chance — and we can compute how unlikely that event would be. Because that event is so unlikely, we therefore conclude that one or both cheated. And indeed, the testing service invalidated the athlete's exam. On the other hand, if all the questions that were answered identically were *correct*, the result might not be unreasonable. If we knew in how many cases they made the *same mistakes*, the inquiry would have been clearer, but the newspaper did not contain those details.

The court is hearing a murder case. There is no eye-witness, and the evidence consists of such facts as the height and weight and age of the person charged, and other circumstantial evidence. Only one person in 50 million has such characteristics, and you find such a person. Will you convict the person, or will you believe that the evidence was just a coincidence? Of course the evidence *might* have occurred by bad luck, but the probability is very, very small (1 in 50 million). Will you therefore conclude that because the chance is so small, it is reasonable to assume that the person charged committed the crime?

Sometimes the unusual really happens — the court errs by judging that the wrong person did it, and that person goes to prison or even is executed. The best we can do is to make the criterion strict: "Beyond a reasonable doubt." (People ask: What probability does that criterion represent? But the court will not provide a numerical answer.)

Somebody says to you: I am going to deal out five cards and it will be a royal flush — ten, jack, queen, king, and ace of the same suit. The person deals the cards and lo and behold! the royal flush appears. Do you think the occurrence happened just by chance? No, you are likely to be very dubious that it happened by chance. Therefore, you believe there must be some other explanation — that the person fixed the cards, for example.

Note: You don't attach the same meaning to any *other* permutation (say 3, 6, 7, 7, and king of various suits), even though that permutation is just as rare — unless the person announced exactly that permutation in advance.

Indeed, even if the person says *nothing*, you will be surprised at a royal flush, because this hand has *meaning*, whereas another given set of five cards do not have any special meaning.

You see six Volvos in one home's driveway, and you conclude that it is a Volvo club meeting, or a Volvo salesperson's meeting. Why? Because it is unlikely that six people not connected formally by Volvo ownership would be friends of the same person.

Two important points complicate the concept of statistical significance:

1. With a large enough sample, *every* treatment or variable will seem different from every other. Two faces of even a good die (say, "1" and "2") will produce different results in the very very long run.

2. Statistical significance does not imply economic or social significance. Two faces of a die may be statistically different in a huge sample of throws, but a 1/10,000 difference between them is too small to make an economic difference in betting. Statistical significance is only a *filter* . If it appears, one should then proceed to decide whether there is *substantive* significance.

Interpreting statistical significance is sometimes complex, especially when the interpretation depends heavily upon your prior expectations — as it often does. For example, how should a basketball coach decide whether or not to bench a player for poor performance after a series of missed shots at the basket?

Consider Coach John Thompson who, after Charles Smith missed 10 of 12 shots in the 1989 Georgetown-Notre Dame NCAA game, took Smith out of the game for a time (*The Washington Post*, March 20, 1989, p. C1). The scientific or decision problem is: Should the coach consider that Smith is not now a 47 percent shooter as he normally is, and therefore the coach should bench him? The statistical question is: How likely is a shooter with a 47 percent average to produce 10 of 12 misses? The key issue in the statistical question concerns the total number of shot attempts we should consider.

Would Coach Thompson take Smith out of the game after he missed *one* shot? Clearly not. Why not? Because one "expects" Smith to miss a shot half the time, and missing one shot therefore does not seem unusual.

How about after Smith misses two shots in a row? For the same reason the coach still would not bench him, because this event happens "often" — more specifically, about once in every sequence of four shots.

How about after 9 misses out of ten shots? Notice the difference between this case and 9 females among ten calves. In the case of the calves, we expected half females because the experiment is a single isolated trial. The event *considered by itself* has a small enough probability that it seems *unexpected* rather than expected. ("Unexpected" seems to be closely related to "happens seldom" or "unusual" in our psychology.) And an event that happens seldom seems to call for explanation, and also seems to promise that it will yield itself to explanation by some unusual concatenation of forces. That is, unusual events lead us to think that they have unusual causes; that is the nub of the matter. (But on the other hand, one can sometimes benefit by paying attention to unusual events, as scientists know when they investigate outliers.)

In basketball shooting, we expect 47 percent of Smith's *individual* shots to be successful, and we also expect that average for each set of shots. But we also expect *some* sets of shots to be far from that average because we observe many sets; such variation is inevitable. So when we see a single set of 9 misses in ten shots, we are not very surprised.

But how about 29 misses in 30 shots? At some point, one must start to pay attention. (And of course we would pay more attention if beforehand, and never at any other time, the player said, "I can't see the basket today. My eyes are dim.")

So, how should one proceed? Perhaps proceed the same way as with a coin that keeps coming down heads a very large proportion of the throws, over a long series of tosses: At *some* point you examine it to see if it has two heads. But if your investigation is negative, in the absence of an indication *other than the behavior in question*, you continue to believe that there is no explanation and you assume that the event is "chance" and *should not be acted upon*. In the same way, a coach might *ask* a player if there is an explanation for the many misses. But if the player answers "no," the coach should not bench him. (There are difficulties here with truth-telling, of course, but let that go for now.)

The key point for the basketball case and other repetitive situations is not to judge that there is an unusual explanation from the behavior of a *single sample alone*, just as with a short sequence of stock-price changes.

We all need to learn that "irregular" (a good word here) sequences are less unusual than they seem to the naked intuition. A streak of 10 out of 12 misses for a 47 percent shooter occurs about 3 percent of the time. That is, about every 33 shots Smith takes, he will begin a sequence of 12 shots that will end with 3 or fewer baskets — perhaps once in every couple of games. This does not seem "very" unusual, perhaps. And if the coach *treats* each such case as unusual, he will be losing some of the services of a better player than he replaces him with.

In brief, how hard one should search for an explanation should depend on the probability of the event. But one should (almost) assume the absence of an explanation unless one actually finds it.

Bayesian analysis (Chapter 31) could be brought to bear upon the matter, bringing in your prior probabilities based on the knowledge of research that has shown that there is no such thing as a "hot hand" in basketball (see Chapter 14), together with some sort of cost-benefit error-loss calculation comparing Smith and the next best available player.

22.2 The concept of statistical significance

"Significance level" is a common term in probability statistics. It corresponds roughly to the probability that the assumed benchmark universe could give rise to a sample as extreme as the observed sample by chance. The results of Example 16-1 would be phrased as follows: The hypothesis that the radiation treatment affects the sex of the fruit fly offspring is accepted as true at the probability level of .16 (sometimes stated as the 16 percent level of significance). (A more common way of expressing this idea would be to say that the hypothesis is *not rejected* at the .16 probability level or the 16 percent level of significance. But "not rejected" and "accepted" really do mean much the same thing, despite some arguments to the contrary.) This kind of statistical work is called hypothesis testing.

The question of *which* significance level should be considered "significant" is difficult. How great must a coincidence be before you refuse to believe that it is only a coincidence? It has been conventional in social science to say that if the probability that something happens by

chance is less than 5 percent, it is significant. But sometimes the stiffer standard of 1 percent is used. Actually, *any* fixed cut-off significance level is arbitrary. (And even the whole notion of saying that a hypothesis "is true" or "is not true" is sometimes not useful.) Whether a one-tailed or two-tailed test is used will influence your significance level, and this is why care must be taken in making that choice.

23 The Statistics of Hypothesis-Testing with Counted Data, Part 2

Here's the bad-news-good-news message again: The bad news is that the subject of inferential statistics is extremely difficult — not because it is complex but rather because it is subtle. The cause of the difficulty is that the world around us is difficult to understand, and spoon-fed mathematical simplifications which you manipulate mechanically simply mislead you into thinking you understand that about which you have not got a clue.

The good news is that you — and that means *you*, even if you say you are "no good at mathematics" — can understand these problems with a layperson's hard thinking, even if you have no mathematical background beyond arithmetic and you think that you have no mathematical capability. That's because the difficulty lies in such matters as pin-pointing the right question, and understanding how to interpret your results.

The problems in the previous chapter were tough enough. But this chapter considers problems with additional complications, such as when there are more than two groups, or paired comparisons for the same units of observation.

But first, we need another addition to our Python vocabulary.

23.1 Logical operators

> Note 50: Notebook: Logical operators
>
> - Download notebook
> - Interact

This section continues our programme of expanding the range of Python features that you can use to clear code. As we introduce each feature, we will use them in the following examples.

As motivation, we are about to do some simulations where we are interested in the number of some particular type of observations in each trial. For example, let's do 10 coin tosses with `rnd.choice`:

```
import numpy as np
rnd = np.random.default_rng()
```

```
# For each element, heads of tails is equally likely.
coins = rnd.choice(['heads', 'tails'], size=10)
coins
```

```
array(['tails', 'tails', 'tails', 'tails', 'tails', 'heads', 'tails',
       'tails', 'tails', 'heads'], dtype='<U5')
```

Let us now say that we are interested to record if the trial had *either* 2 or fewer "heads" *or* two or fewer "tails".

We could write it like this:

```
if np.sum(coins == 'heads') <= 2:
    print('Trial is of interest')
```

```
Trial is of interest
```

```
if np.sum(coins == 'tails') <= 2:
    print('Trial is of interest')
```

It is a little repetitive to have to repeat the code identical code to print the same message for either of the two cases, and it would be even more repetitive if there were more lines of identical code to run for each of the two cases.

Python solves this problem with the `or` operator, like this:

```
if np.sum(coins == 'heads') <= 2 or np.sum(coins == 'tails') <= 2:
    print('Trial is of interest')
```

```
Trial is of interest
```

> **Note 51: What is an operator?**
>
> Above, we called `or` an *operator*. An operator, for our purposes, is a special character , or a word, that sits between two values, and that tells Python how to combine these values.

For example `+` is an operator. When `+` sits between two numbers in code, Python interprets this to mean "take the two numbers on either side, and make a new number that is the result of adding the two numbers":

```
# + is an operator that, between two numbers, means "add the numbers".
1 + 3
```

```
4
```

`+`, `-`, `/` and `*` are all examples of operators that do *arithmetic* on the numbers to either side — they are *arithmetic* operators.

```
# * is an operator that, between two numbers, means "multiply the numbers".
2 * 4
```

```
8
```

We are about to use the operator `or`. `or` is a *logical* operator. It is a logical operator because it does not operate on *numbers* (as arithmetic operators do), but on *logical* (*Boolean*) values — values that can be either `True` or `False`.

For example, here we use `or` to combine a `True` value (on the left) with a `False` value (on the right). It gives a result — `True`.

```
True or False
```

```
True
```

`or` applies a very simple rule: if *either* the left-hand (LH) *or* the right-hand (RH) values are `True`, then `or` evaluates to `True`. Only if *neither* of the LH and RH values are `True`, does it return `False`.

```
# Both LH and RH are True, return True.
print('True or True result:', True or True)
```

```
True or True result: True
```

```
# Only LH is True, return True.
print('True or False result:', True or False)
```

```
True or False result: True
```

```
# Only RH is True, return True.
print('False or True result:', False or True)
```

```
False or True result: True
```

```
# Neither LH nor RH are True, return False.
print('False or False result:', False or False)
```

```
False or False result: False
```

Now let's go back to the `if` statement above. The conditional part of the header line is:

```
np.sum(coins == 'heads') <= 2 or np.sum(coins == 'tails') <= 2
```

```
np.True_
```

This will be `True` *either* when there there are two or fewer "heads", *or* when there are two or fewer tails. Therefore, when we use this conditional in an `if` statement, we make the *body* of the `if` statement run only if either of the two conditions are `True`.

```
if np.sum(coins == 'heads') <= 2 or np.sum(coins == 'tails') <= 2:
    print('Trial is of interest')
```

```
Trial is of interest
```

While we are here, Python has another very useful logical operator: `and`.

`and` takes the LH and RH values, and returns `True` only if *both* values are `True`.

```
# Both LH and RH are True, return True.
print('True and True result:', True and True)
```

```
True and True result: True
```

```
# Only LH is True, return False.
print('True and False result:', True and False)
```

```
True and False result: False
```

```
# Only RH is True, return False.
print('False and True result:', False and True)
```

```
False and True result: False
```

```
# Neither LH nor RH are True, return False.
print('False and False result:', False and False)
```

```
False and False result: False
```

We could, for example, ask whether the number of heads is >=3 *and* <=7 (is in the range 3 through 7).

```
if np.sum(coins == 'heads') >= 3 and np.sum(coins == 'heads') <= 7:
    print('Trial is of interest')
```

i Python interval comparison

In fact, Python has a special shortcut syntax called *interval comparison* for that last question — whether a number is within a range. It looks like this:

```
# Asks whether thee number of heads is >= 3 *and* <= 7.
3 <= np.sum(coins == 'heads') <= 7
```

```
np.False_
```

Notice the value at one end of the range the left (here, the lower value), then the comparison operator, then the value to compare, then another comparison operator, followed by the value at the other end of the range on the right.
The interval comparison above is a shortcut for the more verbose version we would need when using `and`:

```
# Also asks whether thee number of heads is >= 3 *and* <= 7.
3 <= np.sum(coins == 'heads') and np.sum(coins == 'heads') <= 7
```

```
np.False_
```

End of notebook: Logical operators

`logical_operators` starts at Note 50.

23.2 Comparisons among more than two samples of counted data

23.2.1 Example: Do Any of Four Treatments Affect Sex Ratio in Fruit Flies?

This is an example of the general problem — **when the benchmark universe proportion is known, is the proportion of the binomial population affected by any of the treatments?**

Suppose that, instead of experimenting with just one type of radiation treatment on the flies (as in Section 21.2.1), you try *four* different treatments, which we shall label A, B, C, and D. Treatment A produces fourteen males and six females, but treatments B, C, and D produce ten, eleven, and ten males, respectively. It is immediately obvious that there is no reason to think that treatment B, C, or D affects the sex ratio. But what about treatment A?

A frequent and dangerous mistake made by young scientists is to scrounge around in the data for the most extreme result, and then treat it as if it were the only result. In the context of this example, it would be fallacious to think that the probability of the fourteen-males-to-six females split observed for treatment A is the same as the probability that we figured for a *single* experiment in the example Section 21.2.1. Instead, we must consider that our benchmark universe is composed of *four sets* of twenty trials, each trial having a 50-50 probability of being male. We can consider that our previous trials 1-4 in Section 21.2.1 constitute a *single* new trial, and each subsequent set of four previous trials constitute another new trial. We then ask how likely a new trial of our sets of twenty flips is to produce *one* set with fourteen or more of one or the other sex.

Let us make the procedure explicit, starting at the procedure from Section 21.2.1. Again, we will check for 14 or more males, *or* 6 or fewer males (meaning, 14 or more females).

- **Step 1.** Let tails = male, heads = female.
- **Step 2.** Flip twenty coins and count the number of tails (males). Call this the count for group A. Repeat three more times to get counts for groups B, C and D.
- **Step 5.** If *any* of the group counts for A, or B or C or D is 14 ore more then record "Yes", or if *any* of the group counts A, B, C, D are 6 or less (meaning >= 14 females), record "yes". If neither is true, record "No".
- **Step 4.** Repeat steps 2 and 3 perhaps 100 times.
- **Step 5.** Calculate the proportion of "yes" results in the 100 trials. This proportion estimates the probability that a fruit fly population with a proportion of 50 percent

males will produce as many as 14 males, or as many as 14 females, in at least one of four groups of 20 flies.

We begin the trials with data as in Table 23.1. In two of the six simulation trials, one or more one samples (groups) shows 14 or more males. Without even concerning ourselves about whether we should be looking at males or females, or just males, or needing to do more trials, we can see that it would be *very common indeed* to have one of four treatments show fourteen or more of one sex just by chance. This discovery clearly indicates that a result that would be fairly unusual (five in 25) for a single sample alone is commonplace in one of four observed samples.

Table 23.1: Results from 6 random trials for Fruitfly 4-group problem

Trial no	Count for group A	Count for group B	Count for group C	Count for group D	Any >=14 or any <=6
1	12	12	8	11	No
2	8	10	11	6	No
3	6	10	11	12	No
4	7	15	13	9	Yes
5	9	10	10	9	No
6	12	11	4	16	Yes

Note 52: Notebook: Fruit fly simulation of four groups

- Download notebook
- Interact

A key point of the notebook here is that each trial consists of *four groups* of 20 randomly generated hypothetical fruit flies. And if we consider 10,000 trials, we will be examining 40,000 sets of 20 fruit flies.

In each trial we generate 4 random samples (groups) of 20 fruit flies, and for each, we count the number of males ("males"s) and then check whether that group has more than 13 of either sex (actually, more than 13 "males"s or less than 7 "males"). If it does, then we change a variable called `unusual` to 1, which informs us that for this sample, at least 1 group of 20 fruit flies had results as unusual as the results from the fruit flies exposed to the four treatments.

After the 10,000 runs are made, we count the number of trials where one sample had a group of fruit flies with 14 or more of either sex, and show the results.

```
import numpy as np
rnd = np.random.default_rng()
```

```
n_iters = 10_000

# Make array to store results for each trial.
results = np.zeros(n_iters)

for i in range(n_iters):
    # unusual indicates whether we have obtained any trial group with more
    # than 13 of either sex. We start at 0 (= no).
    unusual = 0
    # Repeat the following steps 4 times to constitute 4 trial groups
    # (representing treatments A, B, C, and D) of 20 flies each.
    for j in range(4):
        flies = rnd.choice(['male', 'female'], size=20)
        n_males = np.sum(flies == 'male')
        if n_males >= 14 or n_males <= 6:
            unusual = 1
    # unusual now tells us whether we got a result as extreme as that
    # observed (unusual == 1 if we did, unusual == 0 if not). We must
    # keep track of this result in the results variable, for each experiment.
    results[i] = unusual

# The number of trials for which at least one of the four tests
# exceeded 13 males or 13 females.
k = np.sum(results)
kk = k / n_iters

print('Proportion of trials with one or more group >=14 or <=6 :', kk)
```

```
Proportion of trials with one or more group >=14 or <=6 : 0.3808
```

> **i** End of notebook: Fruit fly simulation of four groups
>
> `fruit_fly4` starts at Note 52.

In one set of 10,000 trials, there were more than 13 males or more than 13 females 38 percent of the time — clearly not an unusual occurrence.

23.2.2 Example: Do Four Psychological Treatments Differ in Effectiveness?

Do Several Two-Outcome Samples Differ Among Themselves in Their Proportions?

Consider four different psychological treatments designed to rehabilitate young offenders. Instead of a numerical test score, there is only a "yes" or a "no" answer as to whether the young person has kept their record clean or has gotten into trouble again. Call a clean record "success". Label the treatments P, R, S, and T, each of which is administered to a separate group of twenty young offenders. The number of "success" outcomes per group has been: P, 17; R, 10; S, 10; T, 7. Is it improbable that all four groups come from the same universe?

This problem is like the placebo vs. cancer-cure problem, but now there are more than two samples. It is also like the four-sample irradiated-fruit flies example (Section 23.2.1), except that now we are *not* asking whether any or some of the samples differ from a *given universe* (50-50 sex ratio in that case). Rather, we are now asking whether there are differences *among* the samples themselves. Please keep in mind that we are still dealing with two-outcome (success-or-failure, yes-or-no, well-or-sick) problems. The outcomes fall into categories, to which we give labels ("success" or "failure", "well" or "sick"). Later we shall take up problems that are similar except that the outcomes are "quantitative" — in that the outcomes are numbers rather than labels.

If all four groups were drawn from the same universe, that universe has an estimated success rate of 17/20 + 10/20 + 10/20 + 7/20 = 44/80 = 55/100 = 55%, because the observed data *taken as a whole* constitute our best guess as to the nature of the universe from which they come — again, *if* they all come from the same universe. (Please think this matter over a bit, because it is important and subtle. It may help you to notice the absence of any *other* information about the universe from which they have all come, if they have come from the same universe.)

Therefore, select twenty two-digit numbers for each group from the random-number table, marking "success" for each number in the range 1 through 55 and "failure" for each number 56 through 100. Conduct a number of such trials. Then count the proportion of times that the difference between the highest and lowest groups is larger than the widest observed difference, the difference between P and T (17-7 = 10). In Table 23.2, none of the first six trials shows anywhere near as large a difference as the observed range of 10, suggesting that it would be rare for four treatments that are "really" similar to show so great a difference. There is thus reason to believe that P and T differ in their effects.

Table 23.2: Results of Six Random Trials for Problem "offenders"

Trial	P	R	S	T	Largest Minus Smallest
1	11	9	8	12	4
2	10	10	12	12	2
3	9	12	8	12	4
4	9	11	12	10	3
5	10	10	11	12	1
6	11	11	9	11	2

The strategy of the Python solution to "offenders" is similar to the strategy for previous problems in this chapter. The benchmark (null) hypothesis is that the treatments do not differ in their effects observed, and we estimate the probability that the observed results would occur by chance using the benchmark universe. The only new twist is that we must instruct the computer to find the groups with the highest and the lowest numbers of success rates.

Using Python, we generate four "treatments," each represented by 20 labels. We draw these 20 labels from the choices "success" or "failure", with a 55% chance of getting "success" and a 45% chance of "failure". Follow along in the program for the rest of the procedure:

> **Note 53: Notebook: Simulation of offender strategies**
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

# set up the random number generator
rnd = np.random.default_rng()

# Set the number of trials
n_trials = 10_000

# Set the sample size for each trial
sample_size = 20

# An empty array to store the trial results.
scores = np.zeros(n_trials)

# Do 10000 trials
for i in range(n_trials):
    # The first treatment group
    a = rnd.choice(['success', 'failure'], p=[0.55, 0.45], size=20)
    # Count successes.
    a_count = np.sum(a == 'success')
    # Second, third and fourth treatment groups, and successes.
    b = rnd.choice(['success', 'failure'], p=[0.55, 0.45], size=20)
    b_count = np.sum(b == 'success')
    c = rnd.choice(['success', 'failure'], p=[0.55, 0.45], size=20)
    c_count = np.sum(c == 'success')
    d = rnd.choice(['success', 'failure'], p=[0.55, 0.45], size=20)
    d_count = np.sum(d == 'success')
```

```
    # Now find all the pairwise differences in successes among the groups.
    a_minus_b = a_count - b_count
    a_minus_c = a_count - c_count
    a_minus_d = a_count - d_count
    b_minus_c = b_count - c_count
    b_minus_d = b_count - d_count
    c_minus_d = c_count - d_count

    # Concatenate, or join, all the differences in a single array "diffs".
    diffs = np.array([a_minus_b, a_minus_c, a_minus_d,
                      b_minus_c, b_minus_d, c_minus_d])
    # Since we are interested only in the magnitude of the difference, not its
    # direction, we take the absolute value of all the differences (we remove
    # any minus signs, making all values positive).
    abs_diffs = np.abs(diffs)
    # Find the largest of all the differences
    max_abs_diff = np.max(abs_diffs)
    # Keep score of the largest
    scores[i] = max_abs_diff

    # End a trial, go back and repeat until all 10000 are complete.

# How many of the trials yielded a maximum difference greater than the
# observed maximum difference?
k = np.sum(scores >= 10)
# Convert to a proportion
kk = k / n_trials

print('Proportion >= 10 was', kk)
```

```
Proportion >= 10 was 0.0121
```

> **i** End of notebook: Simulation of offender strategies
>
> `offenders` starts at Note 53.

Only one percent of the experiments with randomly generated treatments from a common success rate of .55 produced differences in excess of the observed maximum difference (10).

An alternative approach to this problem would be to deal with *each* result's departure from the mean, rather than the largest difference among the pairs. Once again, we want to deal with

absolute departures, since we are interested only in magnitude of difference. We could take the absolute value of the differences, as above, but we will try something different here. *Squaring* the differences also renders them all positive: this is a common approach in statistics.

> **Note 54: Notebook: Simulation of offender strategies using squared differences**
>
> - Download notebook
> - Interact

The first step is to examine our data and calculate this measure: The mean is 11, the differences (call *deviations from the mean*) are 6, 1, 1, and 4, the squared deviations are 36, 1, 1, and 16, and their sum is 54. In Python:

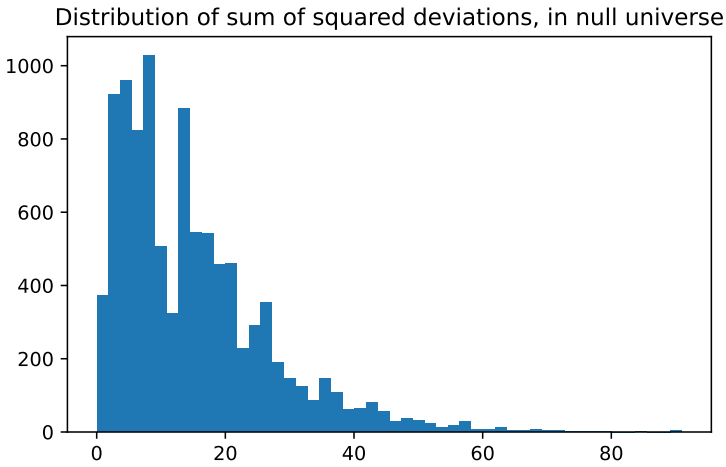
```
# The actual scores for each treatment.
treat_scores = np.array([17, 10, 10, 7])
# The mean.
m = np.mean(treat_scores)
# The four deviations from the mean
deviations = treat_scores - m
# Squared deviations.
sq_deviations = deviations ** 2
# Sum of squared deviations.
actual_sum_sq_deviations = np.sum(sq_deviations)
# Show the result.
actual_sum_sq_deviations
```

```
np.float64(54.0)
```

Our experiment will be, as before, to constitute four groups of 20 at random from a universe with a 55 percent rehabilitation rate. We then calculate this same measure for the random groups. If it is frequently larger than 54, then we conclude that a uniform cure rate of 55 percent could easily have produced the observed results. The program that follows also generates the four treatments by using a `for` loop, rather than spelling out the `rnd.choice` command 4 times as above. In Python:

```
import numpy as np
import matplotlib.pyplot as plt

# Set up the random number generator.
rnd = np.random.default_rng()

# Set the number of trials.
```

```
n_trials = 10000

# Set the sample size for each trial.
sample_size = 20

# An empty array to store the trials.
scores = np.zeros(n_trials)

# Do 10000 trials
for i in range(n_trials):
    # Repeat the following steps 4 times to constitute 4 groups of 20 and
    # count their success rates.
    group_scores = np.zeros(4)
    for group_no in range(4):  # for 0, 1, 2, 3.
        # A treatment group
        group = rnd.choice(['success', 'failure'], p=[0.55, 0.45], size=20)
        # Count successes.
        group_count = np.sum(group == 'success')
        # Store result
        group_scores[group_no] = group_count
        # End the procedure for one group of 20, go back and repeat until all 4
        # are done.
    # Calculate the mean
    m = np.mean(group_scores)
    # Calculate the deviations of the scores from the mean of the scores.
    deviations = group_scores - m
    # Square them, making them them positive.
    sq_deviations = deviations ** 2
    # Sum up the squared deviations.
    sum_sq_deviations = np.sum(sq_deviations)
    # Keep track of the result for each trial.
    scores[i] = sum_sq_deviations

    # End a trial, go back and repeat until all 10000 are complete.

# Produce a histogram of the trial results.
plt.hist(scores, bins=50)
plt.title('Distribution of sum of squared deviations, in null universe')
```



From this histogram, we see that in only about 1 percent of the cases did our trial sum of squared differences equal or exceed 54, confirming our conclusion that this is an unusual result. We can have Python calculate this proportion:

```
# How many of the trials yielded a maximum difference greater than the
# observed maximum difference?
k = np.sum(scores >= actual_sum_sq_deviations)
# Convert to a proportion
kk = k / n_trials

print('Proportion >=', actual_sum_sq_deviations, 'was:', kk)
```

```
Proportion >= 54.0 was: 0.0112
```

The conventional way to approach this problem would be with what is known as a chi-squared test.

> **i** End of notebook: Simulation of offender strategies using squared differences
>
> `offenders_squared` starts at Note 54.

23.2.3 Example: Three-way Comparison

In a national election poll of 750 respondents in May, 1992, George Bush got 36 percent of the preferences (270 voters), Ross Perot got 30 percent (225 voters), and Bill Clinton got 28 percent

(210 voters), with 45 undecided (*Wall Street Journal*, October 29, 1992, A16). Assuming that the poll was representative of actual voting, how likely is it that Bush was actually *behind* and just came out ahead in this poll by chance? Or to put it differently, what was the probability that Bush actually had a plurality of support, rather than that his apparent advantage was a matter of sampling variability? We test this by constructing a universe in which Bush is slightly behind (in practice, just equal), and then drawing samples to see how likely it is that those samples will show Bush ahead.

We must first find that universe — among all possible universes that yield a conclusion contrary to the conclusion shown by the data, and one in which we are interested — that has the highest probability of producing the observed sample. With a two-person race the universe is obvious: a universe that is evenly split except for a single vote against "our" candidate who is now in the lead, i.e. in practice a 50-50 universe. In that simple case we then ask the probability that *that* universe would produce a sample as far out in the direction of the conclusion drawn from the observed sample as the observed sample.

With a three-person race, however, the decision is not obvious (and if this problem becomes too murky for you, skip over it; it is included here more for fun than anything else). And there is no standard method for handling this problem in conventional statistics (a solution in terms of a confidence interval was first offered in 1992, and that one is very complicated and not very satisfactory to me (JLS)). But the sort of thinking that we must labor to accomplish is also required for any conventional solution; the difficulty is inherent in the problem, rather than being inherent in resampling, and resampling will be at least as simple and understandable as any formulaic approach.

Before we start to think about this problem, let us simplify by ignoring the 45 undecided voters, and adjusting the poll percentages accordingly. Of the remaining voters, Bush got 270 / 705 = 38.3%, Perot had 225 / 705 = 31.9%, and Clinton had 210 / 705 = 29.8%. Bush's lead over Perot, in voters with a declared preference, was therefore 38.3 - 31.9 = 6.4%.

The relevant universe is (or so I think) a universe that is 35 Bush — 35 Perot — 30 Clinton (for a race where the poll indicates a 38.3-31.9-29.8% split); the 35-35-30 universe is of interest because it is the universe that is closest to the observed sample that does not provide a win for Bush; it is roughly analogous to the 50-50 split in the two-person race, though a clear-cut argument would require a lot more discussion. A universe that is split 34-34-32, or any of the other possible universes, is less likely to produce a 36-30-28 sample (such as was observed) than is a 35-35-30 universe.[1]. (In technical terms, it might be a "maximum likelihood universe" that we are looking for.) For completeness, we might also try a 36-36-28 universe to see if that produces a result very different than the 35-35-30 universe.

[1] We are interested in choosing the benchmark universe that a) has Bush with equal or lower percentage votes than another candidate and b) has the largest probability of giving rise to the observed vote proportions in the sample. We can show by trying all possible options that the 35, 35, 30 universe has the highest probability in giving rise to the (38.3%, 31.9%, 29.8%) sample percentages (270, 225, 210 vote counts). See this Python notebook for the simulation.

Among those universes where Bush is behind (or equal), a universe that is split 50-50-0 (with just one extra vote for the closest opponent to Bush) would be the most likely to produce a 6 percent difference between the top two candidates by chance, but we are not prepared to believe that the voters are split in such a fashion. This assumption shows that we are bringing some judgments to bear from outside the observed data.

For now, the point is not *how* to discover the appropriate benchmark hypothesis, but rather its *criterion* — which is, I repeat, that universe (among all possible universes) that yields a conclusion contrary to the conclusion shown by the data (and in which we are interested) and that (among such universes that yield such a conclusion) has the highest probability of producing the observed sample.

Let's go through the logic again: 1) Bush apparently has a 6.4% percent lead over the second-place candidate. 2) We ask if the second-place candidate might be ahead if *all voters* were polled, rather than just this sample. We test that by setting up a universe in which the second-place candidate is infinitesimally ahead (in practice, we make the two top candidates equal in our hypothetical universe). And we make the third-place candidate somewhere close to the top two candidates. 3) We then draw samples from this universe and observe how often the result is a 6.4% percent lead for the top candidate (who starts off just below or equal in the universe).

From here on, the procedure is straightforward: Determine how likely that universe is to produce a sample as far (or further) away in the direction of "our" candidate winning. (One could do something like this even if the candidate of interest were not now in the lead.)

This problem teaches again that one must think explicitly about the choice of a benchmark hypothesis. The grounds for the choice of the benchmark hypothesis should precede the notebook, or should be included as an extended commentary within the notebook.

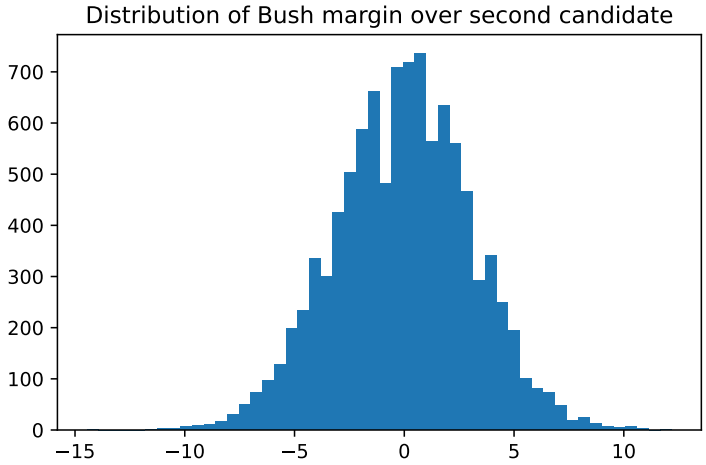
This Python code embodies the previous line of thought.

> **i** Note 55: Notebook: Simulation of Bush / Clinton polling
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

# set up the random number generator
rnd = np.random.default_rng()

# Set the number of trials.
n_trials = 10000
```

```
# Number of voters who expressed a preference.
n_voters = 705

# Benchmark proportions.
bench_ps = [0.35, 0.35, 0.30]

# An empty array to store the trials.
scores = np.zeros(n_trials)

# Do 10000 trials
for i in range(n_trials):
    # Take a sample of 705 votes, with replacement.
    samp = rnd.choice(['Bush', 'Perot', 'Clinton'],
                      p=bench_ps,
                      size=n_voters)
    # Count the Bush voters, etc.
    n_bush = np.sum(samp == 'Bush')
    n_perot = np.sum(samp == 'Perot')
    n_clinton = np.sum(samp == 'Clinton')
    # Join Perot & Clinton votes in an array.
    others = np.array([n_perot, n_clinton])
    # Find the larger of the other two.
    n_second = np.max(others)
    # Find Bush's margin over 2nd.
    lead = n_bush - n_second
    # Convert vote difference to percent lead.
    pct_lead = lead / n_voters * 100
    # Store the result.
    scores[i] = pct_lead

plt.hist(scores, bins=50)
plt.title('Distribution of Bush margin over second candidate')

# Compare to the observed margin in the sample of 705 corresponding to a 6.4
# percent margin by Bush over 2nd place finisher.
k = np.sum(scores >= 6.4)
kk = k / n_trials

print('Proportion of trials where Bush margin >= 6.4%:', kk)
```

```
Proportion of trials where Bush margin >= 6.4%: 0.0182
```



ℹ End of notebook: Simulation of Bush / Clinton polling

`bush_clinton` starts at Note 55.

When we run this program with a 36-36-28 split, we also get a similar result — around 2.2 percent (try it — edit `bench_ps` in the notebook and run it again).

Our main result is that our 35-35-30 analysis shows a probability of only 1.8 percent that Bush would score a 6.4 percentage point "victory" in the sample, by chance, if the universe were split as specified. So Bush could feel reasonably confident that at the time the poll was taken, he was ahead of the other two candidates.

23.3 Paired Comparisons With Counted Data

23.3.1 Example: the pig rations again, but comparing pairs of pigs

This is a **Paired-Comparison Test**.

To illustrate how several different procedures can reasonably be used to deal with a given problem, here is another way to decide whether pig ration A is "really" better: We can assume that the order of the pig scores listed *within each ration group* is random — perhaps the order of the stalls the pigs were kept in, or their alphabetical-name order, or any other random order *not related to their weights*. Match the first pig eating ration A with the first pig eating ration B, and also match the second pigs, the third pigs, and so forth. Then count the number of matched pairs on which ration A does better. On nine of twelve pairings ration A does better, that is, $31.0 > 26.0$, $34.0 > 24.0$, and so forth.

Now we can ask: *If* the two rations are equally good, how often will one ration exceed the other nine or more times out of twelve, just by chance? This is the same as asking how often either heads *or* tails will come up nine or more times in twelve tosses. (This is a "two-tailed" test because, as far as we know, either ration may be as good as or better than the other.) Once we have decided to treat the problem in this manner, it is quite similar to Section 21.2.1 (the first fruitfly irradiation problem). We ask how likely it is that the outcome will be as far away as the observed outcome (9 "heads" of 12) from 6 of 12 (which is what we expect to get by chance in this case if the two rations are similar).

So we conduct perhaps fifty trials as in Table 23.3, where an asterisk denotes nine or more heads or tails.

- **Step 1.** Let odd numbers equal "A better" and even numbers equal "B better."
- **Step 2.** Examine 12 random digits and check whether 9 or more, or 3 or less, are odd. If so, record "yes," otherwise "no."
- **Step 3.** Repeat step 2 fifty times.
- **Step 4.** Compute the proportion "yes," which estimates the probability sought.

The results are shown in Table 23.3.

In 8 of 50 simulation trials, one or the other ration had nine or more tosses in its favor. Therefore, we estimate the probability to be .16 (eight of fifty) that samples this different would be generated by chance if the samples came from the same universe.

Table 23.3: Results from fifty simulation trials of the paired pigs problem

Trial	"Heads" or "Odds" Ration A	"Tails" or "Evens" Ration B	Trial	"Heads" or "Odds" Ration A	"Tails" or "Evens" Ration B
1	6	6	26	6	6
2	4	8	27	5	7
3	6	6	28	7	5
4	7	5	29	4	8
5	3	9	30	6	6
6	5	7	**31**	9	3
7	8	4	**32**	2	10
8	6	6	33	7	5
9	7	5	34	5	7
10	9	3	35	6	6
11	7	5	36	8	4
12	3	9	37	6	6
13	5	7	38	4	8
14	6	6	39	5	7
15	6	6	40	8	4

Trial	"Heads" or "Odds" Ration A	"Tails" or "Evens" Ration B	Trial	"Heads" or "Odds" Ration A	"Tails" or "Evens" Ration B
16	8	4	41	5	7
17	5	7	42	6	6
18	9	3	43	5	7
19	6	6	44	7	5
20	7	5	45	6	6
21	4	8	46	4	8
22	10	2	47	5	7
23	6	6	48	5	7
24	5	7	49	8	4
25	3	9	50	7	5

Now for a Python program and results. This notebook is different from the example at Section 21.2.8 in that it compares the weight-gain results of *pairs* of pigs, instead of simply looking at the *rankings* for weight gains.

The key to the pigs pair notebook is the `rnd.choice` line. If we assume that ration A does not have an effect on weight gain (which is the "benchmark" or "null" hypothesis), then the results of the actual experiment would be no different than if we randomly `rnd.choice` "A" and "B" and treat an "A" as a larger weight gain for the ration A pig, and a "B" as a larger weight gain for the ration B pig. Both events have a .5 chance of occurring for each pair of pigs because if the rations had no effect on weight gain (the null hypothesis), ration A pigs would have larger weight gains about half of the time. The next step is to count (`sum`) the number of times that the weight gains of one group (call it the group fed with ration A) were larger than the weight gains of the other (call it the group fed with ration B). The complete program follows:

> **Note 56: Notebook: Paired test for pig rations**
>
> - Download notebook
> - Interact

```
import numpy as np
rnd = np.random.default_rng()

n_iters = 10_000

# Make array to store results for each trial.
results = np.zeros(n_iters)
```

```
# Do 10000 trials
for i in range(n_iters):
    # Generate randomly 12 "A" or "B"s.  Each of the 12 values represents
    # one "pairing" where "A" = ration A "wins," "B" = ration B "wins.".
    wins = rnd.choice(['A', 'B'], size=12)
    # Count the number of "pairings" where ration A won, put the result in
    # "n_a_wins".
    n_a_wins = np.sum(wins == 'A')
    # Keep track of the result in results.
    results[i] = n_a_wins
    # End the trial, go back and repeat until all 10000 trials are complete.
# Determine how often we got 9 or more "wins" for ration A.
high = np.sum(results >= 9)
# Determine how often we got 3 or fewer "wins" for ration A.
low = np.sum(results <= 3)
# Add the two together
k = high + low
#  Convert to a proportion
kk = k / n_iters
# Print the result.
print('Proportion >=9 or <=3:', kk)
```

```
Proportion >=9 or <=3: 0.1391
```

> **i** End of notebook: Paired test for pig rations
>
> `pig_pairs` starts at Note 56.

Notice how we proceeded in examples Section 21.2.8 and Section 23.3.1. The data were originally quantitative — weight gains in pounds for each pig. But for simplicity we classified the data into simpler counted-data formats. The first format (Section 21.2.8) was a rank order, from highest to lowest. The second format (Section 23.3.1) was simply higher-lower, obtained by randomly pairing the observations (using alphabetical letter, or pig's stall number, or whatever was the cause of the order in which the data were presented to be random). Classifying the data in either of these ways loses some information and makes the subsequent tests somewhat cruder than more refined analysis could provide (as we shall see in the following chapter), but the loss of efficiency is not crucial in many such cases. We shall see how to deal directly with the quantitative data in Chapter 24.

23.3.2 Example: merged firms compared to two non-merged groups

In a study by Simon, Mokhtari, and Simon (1996), a set of 33 advertising agencies that merged over a period of years were each compared to entities within two groups (each also of 33 firms) that did not merge; one non-merging group contained firms of roughly the same size as the final merged entities, and the other non-merging group contained pairs of non-merging firms whose total size was roughly the same as the total size of the merging entities.

The idea behind the matching was that each pair of merged firms was compared against

1. a pair of contemporaneous firms that were roughly the same size as the merging firms *before* the merger, and
2. a single firm that was roughly the same size as the merged entity *after* the merger.

Here (Table 23.4) are the data (provided by the authors):

Table 23.4: Revenue Growth In Year 1 Following Merger

Set #	Merged	Match1	Match2
1	-0.200	0.026	0.000
2	-0.348	-0.125	0.080
3	0.075	0.063	-0.023
4	0.126	-0.042	0.165
5	-0.102	0.080	0.278
6	0.038	0.149	0.430
7	0.116	0.152	0.143
8	-0.098	0.038	0.040
9	0.021	0.077	0.011
10	-0.017	0.284	0.189
11	-0.365	0.139	0.039
12	0.088	0.039	0.095
13	-0.263	0.056	0.045
14	-0.049	0.054	0.008
15	0.011	0.048	0.095
16	0.010	0.198	0.061
17	0.071	0.421	-0.025
18	0.002	0.074	0.053
19	0.005	-0.007	0.050
20	-0.054	0.172	0.110
21	0.023	0.028	-0.022
22	0.060	0.049	0.167
23	-0.060	0.026	0.021
24	-0.089	-0.059	0.077

Set #	Merged	Match1	Match2
25	-0.025	-0.018	0.060
26	0.076	0.013	0.035
27	-0.002	-0.045	0.054
28	-0.220	0.343	0.043
29	0.382	0.221	0.116
30	-0.007	0.255	0.237
31	-0.163	0.011	0.190
32	0.192	0.150	0.152
33	0.061	0.170	0.094

Comparisons were made in several years before and after the mergings to see whether the merged entities did better or worse than the non-merging entities they were matched with by the researchers, but for simplicity we may focus on just one of the more important years in which they were compared — say, the revenue growth rates in the year after the merger.

Here are those average revenue growth rates for the three groups:

Table 23.5: Year's revenue growth

	Mean revenue growth
Merged	-0.021
Match1	0.092
Match2	0.093

We could do a general test to determine whether there are differences among the means of the three groups, as was done in the "Differences Among 4 Pig Rations" problem (Section 24.0.1). However, we note that there may be considerable variation from one matched set to another — variation which can obscure the overall results if we resample from a large general bucket.

Therefore, we use the following resampling procedure that maintains the separation between matched sets by converting each observation into a rank (1, 2 or 3) within the matched set.

Here (Table 23.6) are those ranks, where 1 = worst, 3 = best:

Table 23.6: Revenue growth ranked within matched set (1=best)

Set #	Merged	Match1	Match2
1	1	3	2
2	1	2	3
3	3	2	1

Set #	Merged	Match1	Match2
4	2	1	3
5	1	2	3
6	1	2	3
7	1	3	2
8	1	2	3
9	2	3	1
10	1	3	2
11	1	3	2
12	2	1	3
13	1	3	2
14	1	3	2
15	1	2	3
16	1	3	2
17	2	3	1
18	1	3	2
19	2	1	3
20	1	3	2
21	2	3	1
22	2	1	3
23	1	3	2
24	1	2	3
25	1	2	3
26	3	1	2
27	2	1	3
28	1	3	2
29	3	2	1
30	1	3	2
31	1	2	3
32	3	1	2
33	1	3	2

These (Table 23.7) are the average ranks for the three groups (1 = worst, 3 = best):

Table 23.7: Mean rank of year revenue growth

	Mean rank
Merged	1.49
Match1	2.27
Match2	2.24

Is it possible that the merged group received such a low (poor) average ranking just by chance? The null hypothesis is that the ranks within each set were assigned randomly, and that "merged" came out so poorly just by chance. The following procedure simulates random assignment of ranks to the "merged" group:

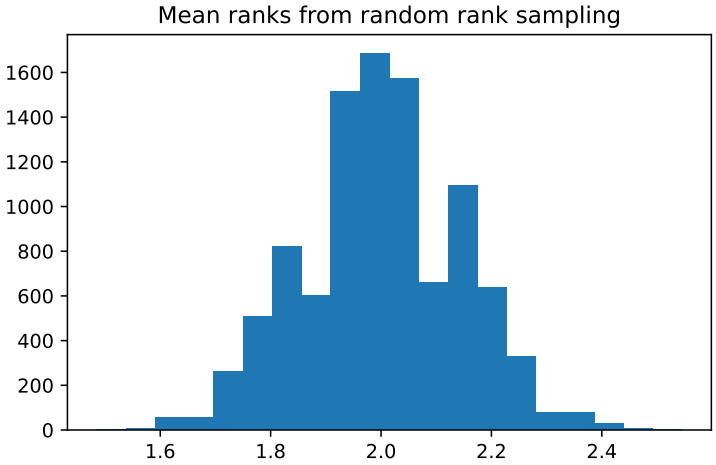
1. Randomly select 33 integers between "1" and "3" (inclusive).
2. Find the average rank & record.
3. Repeat steps 1 and 2, say, 10,000 times.
4. Find out how often the average rank is as low as 1.48.

Here's a Python notebook to apply those steps:

> **Note 57:** Notebook: Merger rank test
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

n_iters = 10_000

# Make array to store results for each trial.
results = np.zeros(n_iters)

for i in range(n_iters):
    ranks = rnd.choice([1, 2, 3], size=33)
    mean_rank = np.mean(ranks)
    results[i] = mean_rank

plt.hist(results, bins=20)
plt.title('Mean ranks from random rank sampling')

k = np.sum(results <= 1.48)
kk = k / n_iters

print('Proportion of mean ranks <= 1.48:', kk)
```

```
Proportion of mean ranks <= 1.48: 0.0
```



End of notebook: Merger rank test

`merger_ranks` starts at Note 57.

Interpretation: 10000 random selections of 33 ranks produced an average as low as the observed average in approximately 0 percent of the trials. Therefore we rule out chance as an explanation for the poor ranking of the merged firms.

Exactly the same technique might be used in experimental medical studies wherein subjects in an experimental group are matched with two different entities that receive placebos or control treatments.

For example, there have been several recent three-way tests of treatments for depression: drug therapy versus cognitive therapy versus combined drug and cognitive therapy. If we are interested in the combined drug-therapy treatment in particular, comparing it to standard existing treatments, we can proceed in the same fashion as in the merger problem.

We might just as well consider the real data from the merger as hypothetical data for a proposed test in 33 triplets of people that have been matched within triplet by sex, age, and years of education. The three treatments were to be chosen randomly within each triplet.

Assume that we now switch scales from the merger data, so that #1 = best and #3 = worst, and that the outcomes on a series of tests were ranked from best (#1) to worst (#3) within each triplet. Assume that the combined drug-and-therapy regime has the best average rank. How sure can we be that the observed result would not occur by chance? Here are the data from the merger study, seen here as Table 23.8:

Table 23.8: Ranked therapies within matched-patient triplets (1=best)

Triplet #	Therapy only	Combined	Drug only
1	1	3	2
2	1	2	3
3	3	2	1
4	2	1	3
5	1	2	3
6	1	2	3
7	1	3	2
8	1	2	3
9	2	3	1
10	1	3	2
11	1	3	2
12	2	1	3
13	1	3	2
14	1	3	2
15	1	2	3
16	1	3	2
17	2	3	1
18	1	3	2
19	2	1	3
20	1	3	2
21	2	3	1
22	2	1	3
23	1	3	2
24	1	2	3
25	1	2	3
26	3	1	2
27	2	1	3
28	1	3	2
29	3	2	1
30	1	3	2
31	1	2	3
32	3	1	2
33	1	3	2

These (Table 23.9) are the average ranks for the three groups (1 = best, 3 = worst):

Table 23.9: Mean rank of therapy in matched triplets

	Mean rank
Therapy only	1.49
Combined	2.27
Drug only	2.24

In these hypothetical data, the average rank for the drug and therapy regime is 1.48. Is it likely that the regimes do not "really" differ with respect to effectiveness, and that the drug and therapy regime came out with the best rank just by the luck of the draw? We test by asking, "If there is no difference, what is the probability that the treatment of interest will get an average rank this good, just by chance?"

We proceed exactly as with the solution for the merger problem (see above).

In the above problems, we did not concern ourselves with chance outcomes for the other therapies (or the matched firms) because they were not our primary focus. If, in actual fact, one of them had done exceptionally well or poorly, we would have paid little notice because their performance was not the object of the study. We needed, therefore, only to guard against the possibility that chance good luck for *our therapy of interest* might have led us to a hasty conclusion.

Suppose now that we are not interested primarily in the combined drug-therapy treatment, and that we have three treatments being tested, all on equal footing. It is no longer sufficient to ask the question "What is the probability that the combined therapy could come out this well just by chance?" We must now ask "What is the probability that *any* of the therapies could have come out this well by chance?" (Perhaps you can guess that this probability will be higher than the probability that our *chosen therapy* will do so well by chance.)

Here is a resampling procedure that will answer this question:

> **Note 58: Notebook: Minimum average rank**
>
> - Download notebook
> - Interact

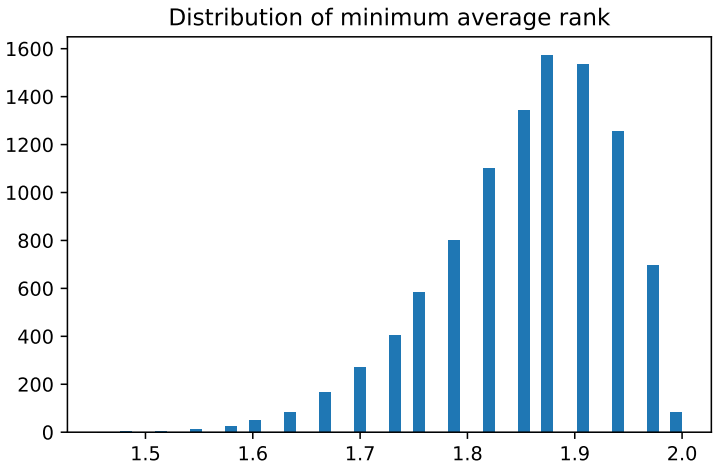
1. Put the numbers "1", "2" and "3" (corresponding to ranks) in a bucket
2. Shuffle the numbers and deal them out to three locations that correspond to treatments (call the locations "t1," "t2," and "t3")
3. Repeat step two another 32 times (for a total of 33 repetitions, for 33 matched triplets)
4. Find the average rank for each location (treatment).
5. Record the minimum (best) score.
6. Repeat steps 2-4, say, 10,000 times.

7. Find out how often the minimum average rank for any treatment is as low as 1.48.

```
import numpy as np
import matplotlib.pyplot as plt

# set up the random number generator
rnd = np.random.default_rng()

# Set the number of trials
n_iters = 10_000

# An empty array to store the trial results.
results = np.zeros(n_iters)

# Step 1 above.
bucket = np.array([1, 2, 3])

n_rows = 33

# Do 10000 trials (step 6).
for i in range(n_iters):
    # Prepare arrays to store shuffled row results.
    t1 = np.zeros(n_rows)
    t2 = np.zeros(n_rows)
    t3 = np.zeros(n_rows)
    # Step 3 (for all 33 rows).
    for row_no in range(n_rows):
        # Step 2.
        shuffled = rnd.permuted(bucket)
        # Step 2 continued.
        t1[row_no] = shuffled[0]
        t2[row_no] = shuffled[1]
        t3[row_no] = shuffled[2]
    # Step 4.
    m_t1 = np.mean(t1)
    m_t2 = np.mean(t2)
    m_t3 = np.mean(t3)
    # Step 5.
    means = np.array([m_t1, m_t2, m_t3])
    min_mean = np.min(means)
    results[i] = min_mean

plt.hist(results, bins=50)
```

```
plt.title('Distribution of minimum average rank')

# Step 7.
k = np.sum(results <= 1.48)
kk = k / n_iters

print('Proportion minimum average rank <= 1.48:', kk)
```

```
Proportion minimum average rank <= 1.48: 0.0001
```



ℹ End of notebook: Minimum average rank

`minimum_average_rank` starts at Note 58.

Interpretation: We did 10,000 random shufflings of 33 ranks, apportioned to three "treatments". Of these, about 0 percent produced, for the best treatment in the three, an average as low as the observed average, therefore we rule out chance as an explanation for the success of the combined therapy.

An interesting feature of the mergers (or depression treatment) problem is that it would be hard to find a conventional test that would handle this three-way comparison in an efficient manner. Certainly it would be impossible to find a test that does not require formulae and tables that only a talented professional statistician could manage satisfactorily, and even they are not likely to fully understand those formulaic procedures.

23.4 Technical note

Some of the tests introduced in this chapter are similar to standard non-parametric rank and sign tests. They differ less in the structure of the test statistic than in the way in which significance is assessed (the comparison is to multiple simulations of a model based on the benchmark hypothesis, rather than to critical values calculated analytically).

24 The Statistics of Hypothesis-Testing With Measured Data

Chapter 21 and Chapter 19 discussed testing a hypothesis with data that either arrive in dichotomized (yes-no) form, or come as data in situations where it is convenient to dichotomize. We next consider hypothesis testing using measured data. Conventional statistical practice employs such devices as the "t-test" and "analysis of variance." In contrast to those complex devices, the resampling method does not differ greatly from what has been discussed in previous chapters.

24.0.1 Example: The Pig Rations Still Once Again, Using Measured Data

Testing for the difference between means of two equal-sized samples of measured-data observations

Let us now treat the pig-food problem without converting the quantitative data into qualitative data, because a conversion always loses information.

The term "lose information" can be understood intuitively. Consider two sets of three sacks of corn. Set A includes sacks containing, respectively, one pound, two pounds, and three pounds. Set B includes sacks of one pound, two pounds, and a hundred pounds. If we *rank* the sacks by weight, the two sets can no longer be distinguished. The one-pound and two-pound sacks have ranks one and two in both cases, and their relative places in their sets are the same. But if we know *not only* that the one-pound sack is the smallest of its set and the three-pound or hundred-pound sack is the largest, but *also* that the largest sack is three pounds (or a hundred pounds), we have more information about a set than if we only know the ranks of its sacks.

Rank data are also known as "ordinal" data, whereas data measured in (say) pounds are known as "cardinal" data. Even though converting from cardinal (measured) to ordinal (ranked) data loses information, the conversion may increase convenience, and may therefore be worth doing in some cases.

Table 24.1 has the measured data for pig rations A and B.

Table 24.1: Measured data for pig rations A and B

ration	weight_gain
A	31
A	34
A	29
A	26
A	32
A	35
A	38
A	34
A	31
A	29
A	32
A	31
B	26
B	24
B	28
B	29
B	30
B	29
B	31
B	29
B	32
B	26
B	28
B	32

We begin a measured-data procedure by noting that if the two pig foods are the same, then each of the observed weight gains came from the *same benchmark universe*. This is the basic tactic in our statistical strategy. That is, if the two foods came from the same universe, *our best guess about the composition of that universe is that it includes weight gains just like the twenty-four we have observed*, and in the same proportions, because that is all the information that we have about the universe; this is the bootstrap method. Since ours is (by definition) a sample from an infinite (or at least, a very large) universe of possible weight gains, we assume that there are *many* weight gains in the universe just like the ones we have observed, in the same proportion as we have observed them. For example, we assume that 2/24 of the universe is composed of 34-pound weight gains, as seen in Figure 24.1:

We recognize, of course, that weight gains other than the exact ones we observed certainly would occur in repeated experiments. And if we thought it reasonable to do so, we could assume that the "distribution" of the weight gains would follow a regular "smooth" shape

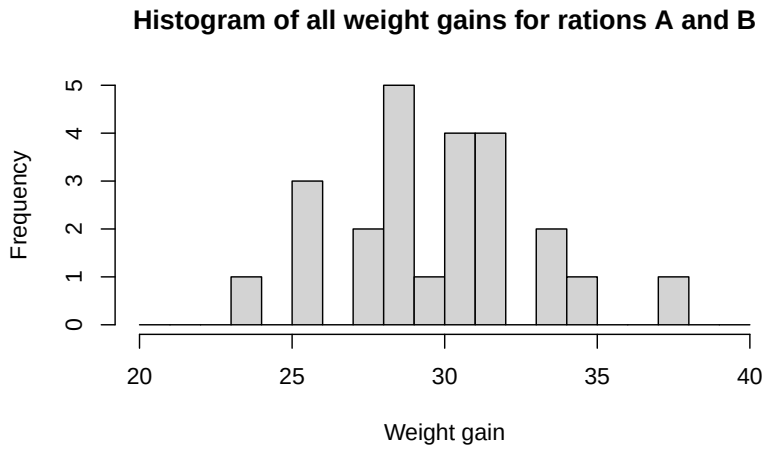


Figure 24.1: Histogram of weight gain

such as Figure 24.2. But deciding just how to draw Figure 24.2 from the data in Figure 24.1 requires that we make arbitrary assumptions about unknown conditions. And if we were to draw Figure 24.2 in a form that would be sufficiently regular for conventional mathematical analysis, we might have to make some *very strong* assumptions going far beyond the observed data.

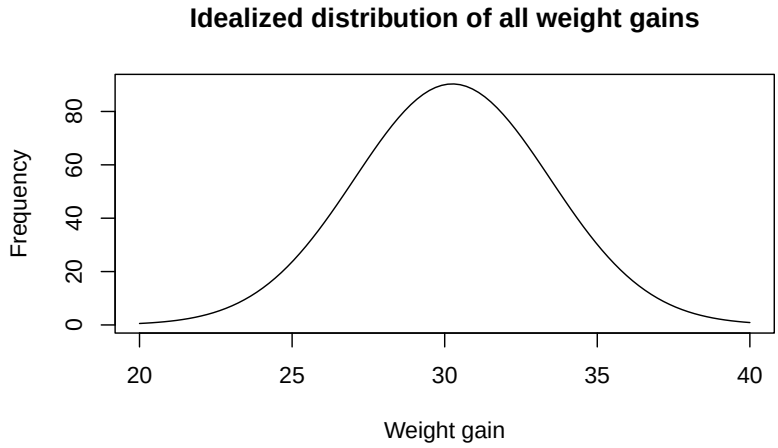


Figure 24.2: Idealized distribution for weight gain

Drawing a smooth curve such as Figure 24.2 from the raw data in Figure 24.1 might be satisfactory — if done with wisdom and good judgment. But there is no necessity to draw such a smooth curve, in this case or in most cases. We can proceed by assuming simply that the

benchmark universe — the universe to which we shall compare our samples, conventionally called the "null" or "hypothetical" universe — is composed only of elements similar to the observations we have in hand. We thereby lose no efficiency and avoid making unsound assumptions.

To carry out our procedure in practice: 1) Write down each of the twenty-four weight gains on a blank index card. We then have one card each for 31, 34, 29, 26, and so on. 2) Shuffle the twenty-four cards thoroughly, and pick one card. 3) Record the weight gain, and replace the card. (Recall that we are treating the weight gains as if they come from an infinite universe — that is, as if the probability of selecting any amount is the same no matter which others are selected randomly. Another way to say this is to state that each selection is independent of each other selection. If we did not replace the card before selecting the next weight gain, the selections would no longer be independent. See Chapter 11 for further discussion of this issue.) 4) Repeat this process until you have made two sets of 12 observations. 5) Call the first hand "food A" and the second hand "food B." Determine the average weight gain for the two hands, and record it as in Table 24.2. Repeat this procedure many times.

In operational steps:

- **Step 1.** Write down each observed weight gain on a card, e.g. 31, 34, 29 …
- **Step 2.** Shuffle and deal a card.
- **Step 3.** Record the weight and replace the card.
- **Step 4.** Repeat steps 2 and 3 eleven more times; call this group A.
- **Step 5.** Repeat steps 2-3 another twelve times; call this group B.
- **Step 6.** Calculate the mean weight gain of each group.
- **Step 7.** Subtract the mean of group A from the mean of group B and record. If larger (more positive) than 3.16 (the difference between the observed means) or more negative than -3.16, record "more." Otherwise record "less."
- **Step 8.** Repeat this procedure perhaps fifty times, and calculate the proportion "more." This estimates the probability sought.

In none of the first ten simulated trials did the difference in the means of the random hands exceed the observed difference (3.16 pounds, in the top line in the table) between foods A and B. (The difference between group *totals* tells the same story and is faster, requiring no division calculations.)

In the old days before a computer was easily available, I (JLS) would quit making trials at such a point, confident that a difference in means as great as observed is not likely to happen by chance. (Using the convenient "multiplication rule" described in Chapter 9, we can estimate the probability of such an occurrence happening by chance in 10 successive trials as $\frac{1}{2} * \frac{1}{2} * \frac{1}{2} ... = \frac{1}{2}^{10} = 1/1024 \approx .001 = .1$ percent, a small chance indeed.) Nevertheless, let us press on to do 50 trials.

Table 24.2: Results of fifty random samples for measured pig rations

Trial #	Mean of first 12 observations (first hand)	Mean of second 12 observations (second hand)	Difference	Greater or less than observed difference?
Observed	382 / 12=31.83			
1	368 / 12=30.67	357 / 12=29.75	.87	Less
2	364 / 12=30.33	361 / 12=30.08	.25	Less
3	352 / 12=29.33	373 / 12=31.08	(1.75)	Less
4	378 / 12=31.50	347 / 12=28.92	2.58	Less
5	365 / 12=30.42	360 / 12=30.00	.42	Less
6	352 / 12=29.33	373 / 12=31.08	(1.75)	Less
7	355 / 12=29.58	370 / 12=30.83	(1.25)	Less
8	366 / 12=30.50	359 / 12=29.92	.58	Less
9	360 / 12=30.00	365 / 12=30.42	(.42)	Less
10	355 / 12=29.58	370 / 12=30.83	(1.25)	Less
11	359 / 12=29.92	366 / 12=30.50	(.58)	Less
12	369 / 12=30.75	356 / 12=29.67	1.08	”
13	360 / 12=30.00	365 / 12=30.42	(.42)	Less
14	377 / 12=31.42	348 / 12=29.00	2.42	Less
15	365 / 12=30.42	360 / 12=30.00	.42	Less
16	364 / 12=30.33	361 / 12=30.08	.25	Less
17	363 / 12=30.25	362 / 12=30.17	.08	Less

Trial #	Mean of first 12 observations (first hand)	Mean of second 12 observations (second hand)	Difference	Greater or less than observed difference?
18	365 / 12=30.42	360 / 12=30.00	.42	Less
19	369 / 12=30.75	356 / 12=29.67	1.08	Less
20	369 / 12=30.75	356 / 12=29.67	1.08	Less
21	369 / 12=30.75	356 / 12=29.67	1.08	Less
22	364 / 12=30.33	361 / 12=30.08	.25	Less
23	363 / 12=30.25	362 / 12=30.17	.08	Less
24	363 / 12=30.25	362 / 12=30.17	.08	Less
25	364 / 12=30.33	361 / 12=30.08	.25	Less
26	359 / 12=29.92	366 / 12=30.50	(.58)	Less
27	362 / 12=30.17	363 / 12=30.25	(.08)	Less
28	362 / 12=30.17	363 / 12=30.25	(.08)	Less
29	373 / 12=31.08	352 / 12=29.33	1.75	Less
30	367 / 12=30.58	358 / 12=29.83	.75	Less
31	376 / 12=31.33	349 / 12=29.08	2.25	Less
32	365 / 12=30.42	360 / 12=30.00	.42	Less
33	357 / 12=29.75	368 / 12=30.67	(1.42)	Less
34	349 / 12=29.08	376 / 12=31.33	2.25	Less
35	356 / 12=29.67	396 / 12=30.75	(1.08)	Less
36	359 / 12=29.92	366 / 12=30.50	(.58)	Less

Trial #	Mean of first 12 observations (first hand)	Mean of second 12 observations (second hand)	Difference	Greater or less than observed difference?
37	372 / 12=31.00	353 / 12=29.42	1.58	Less
38	368 / 12=30.67	357 / 12=29.75	.92	Less
39	344 / 12=28.67	382 / 12=31.81	(3.16)	Equal
40	365 / 12=30.42	360 / 12=30.00	.42	Less
41	375 / 12=31.25	350 / 12=29.17	2.08	Less
42	353 / 12=29.42	372 / 12=31.00	(1.58)	Less
43	357 / 12=29.75	368 / 12=30.67	(.92)	Less
44	363 / 12=30.25	362 / 12=30.17	.08	Less
45	353 / 12=29.42	372 / 12=31.00	(1.58)	Less
46	354 / 12=29.50	371 / 12=30.92	(1.42)	Less
47	353 / 12=29.42	372 / 12=31.00	(1.58)	Less
48	366 / 12=30.50	350 / 12=29.92	.58	Less
49	364 / 12=30.53	361 / 12=30.08	.25	Less
50	370 / 12=30.83	355 / 12=29.58	1.25	Less

Table 24.2 shows fifty trials of which only one (the thirty-ninth) is as "far out" as the observed samples. These data give us an estimate of the probability that, if the two foods come from the same universe, a difference this great or greater would occur just by chance. (Compare this 2 percent estimate with the probability of roughly 1 percent estimated with the conventional t test — a "significance level" of 1 percent.) On the average, the test described in this section yields a significance level as high as such mathematical-probability tests as the t test — that is, it is just as efficient — though the tests described in the examples of Section 21.2.8 and Section 23.3.1 are likely to be less efficient because they convert measured data to ranked or

classified data.[1]

It is *not* appropriate to say that these data give us an estimate of the probability that the foods "do not come" from the same universe. This is because we can *never* state a probability that a sample came from a given universe unless the alternatives are fully specified in advance.[2]

This example also illustrates how the *dispersion within samples* affects the difficulty of finding out whether the samples differ from each other. For example, the average weight gain for food A was 32 pounds, versus 29 pounds for food B. If *all* the food A-fed pigs had gained weight within a range of say 29.9 and 30.1 pounds, and if all the food B-fed pigs had gained weight within a range of 28.9 and 29.1 pounds — that is, if the highest weight gain in food B had been lower than the lowest weight gain in food A — then there would be no question that food A is better, and even fewer observations would have made this statistically conclusive. Variation (dispersion) is thus of great importance in statistics and in the social sciences. The larger the dispersion among the observations *within* the samples, the larger the *sample size* necessary to make a conclusive comparison between two groups or reliable estimates of summarization statistics. (The dispersion might be measured by the mean absolute deviation (the average absolute difference between the mean and the individual observations, treating both plus and minus differences as positive), the variance (the average *squared* difference between the mean and the observations), the standard deviation (the square root of the variance), the range (the difference between the smallest and largest observations), or some other device.)

If you are performing your tests by hand rather than using a computer (a good exercise even nowadays when computers are so accessible), you might prefer to work with the median instead of the mean, because the median requires less computation. (The median also has the advantage of being less influenced by a single far-out observation that might be quite atypical; all measures have their special advantages and disadvantages.) Simply compare the difference in medians of the twelve-pig resamples to the difference in medians of the actual samples, just as was done with the means. The only operational difference is to substitute the word "median" for the word "mean" in the steps listed above. You may need a somewhat larger number of trials when working with medians, however, for they tend to be less precise than means.

The Python notebook compares the difference in the sums of the weight gains for the actual pigs against the difference resulting from two randomly-chosen groups of pigs, using the same

[1] Technical Note: The test described in this section is non-parametric and therefore makes no assumptions about the shapes of the distributions, which is good because we would be on soft ground if we assumed normality in the pig-food case, given the sample sizes. This test does not, however, throw away information as do the rank and median tests illustrated earlier. And indeed, this test proves to be more powerful than the other non-parametric tests. After developing this test, I discovered that its general logic follows the tradition of the "randomization" tests, based on an idea by R.A. Fisher (1935; 1960, chap. III, section 21) and worked out for the two-sample cases by E.J.G. Pitman (1937). But the only earlier mentions of sampling from the universe of possibilities are in M. Dwass (1957) and J.H. Chung and D. Fraser (1958). I am grateful to J. Pratt for bringing the latter literature to my attention.

[2] This short comment is the tip of the iceberg of an argument that has been going on for 200 years among statisticians. It needs much more discussion to be understandable or persuasive.

numerical weight gains of individual pigs as were obtained in the actual experiment. If the differences in average weight gains of the randomly ordered groups are rarely as large as the difference in weight gains from the actual sets of pigs fed food A-alpha and food B-beta, then we can conclude that the foods do make a difference in pigs' weight gains.

Note first that pigs in group A gained a total of 382 pounds while group B gained a total of 344 pounds — 38 fewer. To minimize computations, we will deal with totals like these, not averages.

First we construct vectors A and B of the weight gains of the pigs fed with the two foods. Then we combine the two vectors into one long vector and select two groups of 12 randomly and with replacement (the two `rnd.choice` commands). Notice we sample *with replacement*. This is the *bootstrap* procedure*, where we simulate new samples by resampling with replacement from the original sample. We `sum` the weight gains for the two resamples, and calculate the difference. We keep track of those differences by storing them in the `results` array, graph them on a histogram, and see how many times resample A exceeded resample B by at least 38 pounds, or vice versa (we are testing whether the two are different, not whether food A produces larger weight gains).

Notebook with data file

As you saw in Note 36, the following notebook reads a data file, so the download link points to a `.zip` file containing the notebook and the data file.

Note 59: Notebook: Pig rations via bootstrap

- Download zip with notebook + data file
- Interact

First we need to get the measured data from the data file using the Pandas library:

```
# Load the Numpy library for arrays.
import numpy as np
# Load the Pandas library for loading and selecting data.
import pandas as pd
```

We load the file containing the data:

```
# Read the data file containing pig ration data.
rations_df = pd.read_csv('data/pig_rations.csv')
# Show the first 5 rows.
rations_df.head()
```

```
  ration  weight_gain
0      A           31
1      A           34
2      A           29
3      A           26
4      A           32
```

Let us first select the rows containing data for ration B (we will get the rows for ration A afterwards):

```
# Select ration B rows.
ration_b_df = rations_df[rations_df['ration'] == 'B']
#  Show the first five rows.
ration_b_df.head()
```

```
   ration  weight_gain
12      B           26
13      B           24
14      B           28
15      B           29
16      B           30
```

Finally for ration B, convert the weights to an array for use in the simulation.

```
b_weights = np.array(ration_b_df['weight_gain'])
# Show the result.
b_weights
```

```
array([26, 24, 28, 29, 30, 29, 31, 29, 32, 26, 28, 32])
```

Select ration A rows, and get the weights as an array:

```
ration_a_df = rations_df[rations_df['ration'] == 'A']
a_weights = np.array(ration_a_df['weight_gain'])
# Show the result.
a_weights
```

```
array([31, 34, 29, 26, 32, 35, 38, 34, 31, 29, 32, 31])
```

We will use the `a_weights` and `b_weights` arrays for our simulation. We are going to shuffle these weights, so we first *concatenate* the two arrays (see Section 12.15.1) so we can shuffle them:

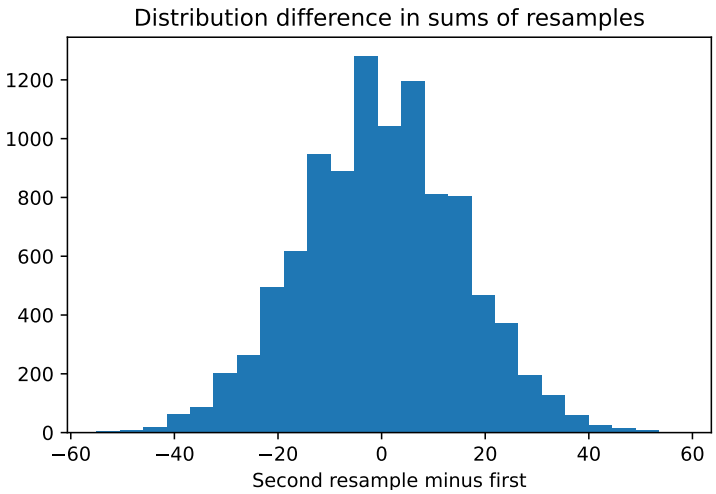
```
both = np.concatenate([a_weights, b_weights])
both
```

```
array([31, 34, 29, 26, 32, 35, 38, 34, 31, 29, 32, 31, 26, 24, 28, 29, 30,
       29, 31, 29, 32, 26, 28, 32])
```

Now do the simulation:

```
import matplotlib.pyplot as plt

# set up the random number generator
rnd = np.random.default_rng()

# Set the number of trials
n_trials = 10_000

# An empty array to store the trial results.
results = np.zeros(n_trials)

# Do 10,000 experiments.
for i in range(n_trials):
    # Take a "resample" of 12 with replacement from both and put it in fake_a
    fake_a = rnd.choice(both, size=12)
    # Likewise to make fake_b
    fake_b = rnd.choice(both, size=12)
    # Sum the first "resample."
    fake_a_sum = np.sum(fake_a)
    # Sum the second "resample."
    fake_b_sum = np.sum(fake_b)
    # Calculate the difference between the two resamples.
    fake_diff = fake_a_sum - fake_b_sum
    # Keep track of each trial result.
    results[i] = fake_diff
    # End one experiment, go back and repeat until all trials are complete,
    # then proceed.
# Produce a histogram of trial results.
plt.hist(results, bins=25)
```

```
plt.xlabel('Second resample minus first')
plt.title('Distribution difference in sums of resamples')
```



From this histogram we see that a very small proportion of the trials produced a difference between groups as large as that observed (or larger). Python will calculate this for us with the following code:

```
# Determine how many of the trials produced a difference between resamples.
count_more = np.sum(results >= 38)
# Likewise for a difference of -38.
count_less = np.sum(results <= -38)
# Add the two together.
k = count_more + count_less
# Divide by number of trials to convert to proportion.
kk = k / n_trials
# Print the result.
print('Proportion of trials with either >=38 or <=-38:', kk)
```

```
Proportion of trials with either >=38 or <=-38: 0.0148
```

ℹ End of notebook: Pig rations via bootstrap

`measured_rations` starts at Note 59.

24.0.2 Example: Is There a Difference in Liquor Prices Between State-Run and Privately-Run Systems?

This is an example of **testing for differences between means of unequal-sized samples of measured data**.

In the 1960s I (JLS) studied the price of liquor in the sixteen "monopoly" states (where the state government owns the retail liquor stores) compared to the twenty-six states in which retail liquor stores are privately owned. (Some states were omitted for technical reasons. And it is interesting to note that the situation and the price pattern has changed radically since then.) These data were introduced in the context of a problem in probability in Section 12.15.

Table 24.3 is the same as the matching table in Section 12.15. They show the representative 1961 prices of a fifth of Seagram 7 Crown whiskey in the two sets of states:[3]

Table 24.3: Whiskey prices by state category

Private	Government
4.82	4.65
5.29	4.55
4.89	4.11
4.95	4.15
4.55	4.2
4.9	4.55
5.25	3.8
5.3	4.0
4.29	4.19
4.85	4.75
4.54	4.74
4.75	4.5
4.85	4.1
4.85	4.0
4.5	5.05
4.75	4.2
4.79	
4.85	
4.79	
4.95	
4.95	
4.75	
5.2	

[3]The data are from *The Liquor Handbook* (1962, p. 68). Eight states are omitted for various reasons. For more information, see Simon and Simon (1996).

	Private	Government
	5.1	
	4.8	
	4.29	
Count	26	16
Mean	4.84	4.35

The *economic* question that underlay the investigation — having both theoretical and policy ramifications — is as follows: Does state ownership affect prices? The *empirical* question is whether the prices in the two sets of states were systematically different. In *statistical* terms, we wish to test the hypothesis that there was a difference between the groups of states related to their mode of liquor distribution, or whether the observed $.49 differential in means might well have occurred by happenstance. In other words, we want to know whether the two subgroups of states differed systematically in their liquor prices, or whether the observed pattern could well have been produced by chance variability.

The first step is to examine the two sets of data graphically to see whether there was such a clear-cut difference between them — of the order of Snow's data on cholera, or the Japanese Navy data on beri-beri — that no test was necessary. The separate displays, and then the two combined together, are shown in Figure 24.3; the answer is not clear-cut and hence a formal test is necessary.

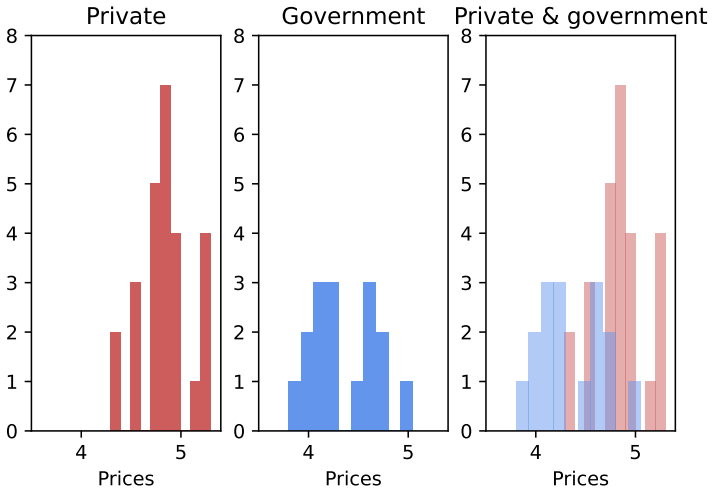


Figure 24.3: Whiskey prices by state category

At first I used a resampling permutation test as follows: Assuming that the entire universe of possible prices consists of the set of events that were observed, because that is all the

information available about the universe, I wrote each of the forty-two observed state prices on a separate card. The shuffled deck simulated a situation in which each state has an equal chance for each price.

On the "null hypothesis" that the two groups' prices do *not* reflect different price-setting mechanisms, but rather differ only by chance, I then examined how often that simulated universe stochastically produces groups with results as different as observed in 1961. I repeatedly dealt groups of 16 and 26 cards, without replacing the cards, to simulate hypothetical monopoly-state and private-state samples, each time calculating the difference in mean prices.

The probability that the benchmark null-hypothesis universe would produce a difference between groups as large or larger than observed in 1961 is estimated by how frequently the mean of the group of randomly-chosen sixteen prices from the simulated state-ownership universe is less than (or equal to) the mean of the actual sixteen state-ownership prices. If the simulated difference between the randomly-chosen groups was frequently equal to or greater than observed in 1961, one would not conclude that the observed difference was due to the type of retailing system because it could well have been due to chance variation.

Here is that procedure as a Python notebook. Compare to the very similar bootstrap procedure in Section 12.15.

> **Note 60: Notebook: Permutation test of public and private liquor prices**
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np

rnd = np.random.default_rng()

# Import the plotting library
import matplotlib.pyplot as plt
```

```
# Load the data from a data file.
prices_df = pd.read_csv('data/liquor_prices.csv')
# Show this first five rows.
prices_df.head()
```

```
  state_type  price
0    private   4.82
1    private   5.29
2    private   4.89
3    private   4.95
4    private   4.55
```

Take all prices from the loaded data file, and convert into an arrays for each category.

```
# Rows for private prices.
priv_df = prices_df[prices_df['state_type'] == 'private']
# Convert corresponding prices to array.
priv = np.array(priv_df['price'])
# Show the result
priv
```

```
array([4.82, 5.29, 4.89, 4.95, 4.55, 4.9 , 5.25, 5.3 , 4.29, 4.85, 4.54,
       4.75, 4.85, 4.85, 4.5 , 4.75, 4.79, 4.85, 4.79, 4.95, 4.95, 4.75,
       5.2 , 5.1 , 4.8 , 4.29])
```

```
# Rows for government prices.
govt_df = prices_df[prices_df['state_type'] == 'government']
# Convert corresponding prices to array.
govt = np.array(govt_df['price'])
# Show the result
govt
```

```
array([4.65, 4.55, 4.11, 4.15, 4.2 , 4.55, 3.8 , 4.  , 4.19, 4.75, 4.74,
       4.5 , 4.1 , 4.  , 5.05, 4.2 ])
```

Calculate actual difference:

```
actual_diff = np.mean(priv) - np.mean(govt)
actual_diff
```

```
np.float64(0.49221153846153953)
```

Concatenate the private and government values into one array:

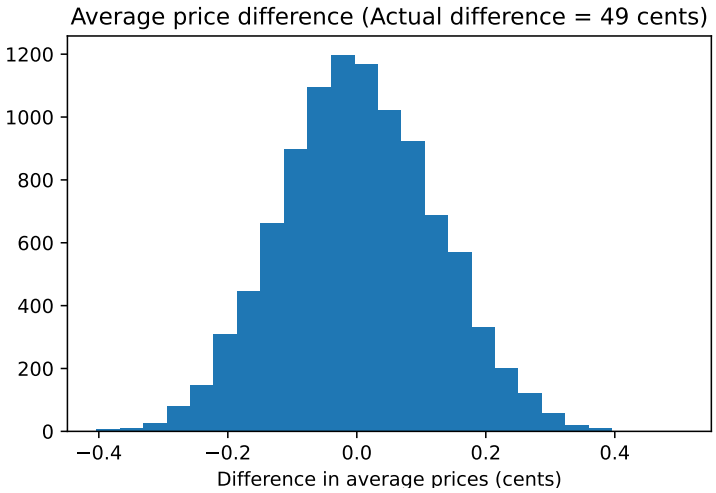
```
# Join the two arrays of data into one array.
both = np.concatenate([priv, govt])
both
```

```
array([4.82, 5.29, 4.89, 4.95, 4.55, 4.9 , 5.25, 5.3 , 4.29, 4.85, 4.54,
       4.75, 4.85, 4.85, 4.5 , 4.75, 4.79, 4.85, 4.79, 4.95, 4.95, 4.75,
       5.2 , 5.1 , 4.8 , 4.29, 4.65, 4.55, 4.11, 4.15, 4.2 , 4.55, 3.8 ,
       4.  , 4.19, 4.75, 4.74, 4.5 , 4.1 , 4.  , 5.05, 4.2 ])
```

Do simulation:

```
n_trials = 10_000

# Fake differences for each trial.
results = np.zeros(n_trials)

# Repeat 10000 simulation trials
for i in range(n_trials):

    # Shuffle 42 values to a random order.
    shuffled = rnd.permuted(both)

    # Take first 26 shuffled values as fake private group
    fake_priv = shuffled[:26]

    # Remaining values (from position 26 to end, 16 values)
    # form the fake government group.
    fake_govt = shuffled[26:]

    # Find the mean of the "private" group.
    p = np.mean(fake_priv)

    # Mean of the "govt." group
    g = np.mean(fake_govt)

    # Difference in the means
    diff = p - g

    # Keep score of the trials
    results[i] = diff

# Graph of simulation results to compare with the observed result.
plt.hist(results, bins=25)
plt.xlabel('Difference in average prices (cents)')
plt.title('Average price difference (Actual difference = '
f'{actual_diff * 100:.0f} cents)');

# Number of trials where fake difference >= actual.
k = np.sum(results >= actual_diff)
kk = k / n_trials

print('Proportion fake differences <= actual_difference:', kk)
```

```
Proportion fake differences <= actual_difference: 0.0001
```



> End of notebook: Permutation test of public and private liquor prices
>
> `liquor_permutation` starts at Note 60.

The result — about zero percent of the simulations generated a value as large or larger than the actual difference — implies that there is a very small probability that two groups with mean prices as different as were observed would happen by chance if drawn from the universe of 42 observed prices. So we "reject the null hypothesis" and instead find persuasive the proposition that the type of liquor distribution system influences the prices that consumers pay.[4]

The logical framework of this resampling version of the permutation test differs greatly from the formulaic version, which relies on some sophisticated mathematics. The formula-method alternative would be the Student's t-test, in which the user simply plugs into an unintuitive formula and reads the result from a table.

A Python program to handle the liquor problem with an infinite-universe bootstrap distribution simply substitutes the random sampling command `rnd.choice` for the commands to do shuffling and splitting above. The results of the bootstrap test are indistinguishable from those in the permutation version of the program above — see: Section 12.15.

[4] Various tests indicate that the difference between the groups of states was highly significant. See Simon and Simon (1996).

Still another difficult question is whether *any* hypothesis test is appropriate, because the states were not randomly selected for inclusion in one group or another, and the results could be caused by factors other than the liquor system; this applies to both the above methods. The states constitute the entire universe in which we are interested, rather than being a sample taken from some larger universe as with a biological experiment or a small survey sample. But this objection pertains to a conventional test as well as to resampling methods. And a similar question arises throughout medical and social science — to the two water suppliers between which John Snow detected vast differences in cholera rates, to rates of lung cancer in human smokers, to analyses of changes in speeding laws, and so on.

The appropriate question is not whether the units were assigned randomly, however, but whether there is strong reason to believe that the results are not meaningful because they are the result of a *particular* "hidden" variable.

These debates about fundamentals illustrate the unsettled state of statistical thinking about basic issues. Other disciplines also have their controversies about fundamentals. But in statistics these issues arise as early as the introductory course, because all but the most contrived problems are shot through with these questions. Instructors and researchers usually gloss over these matters, as Gigerenzer et al., show (*The Empire of Chance*). Again, because with resampling one does not become immersed in the difficult mathematical techniques that underlie conventional methods, one is quicker to see these difficult questions, which apply equally to conventional methods and resampling.

24.0.3 Example: Is there a difference between treatments to prevent low birthweights?

Next we consider the use of resampling with *measured* data to test the hypothesis that drug A prevents low birthweights (Rosner 1995, 291). The data for the treatment and control groups are shown in Table 24.4.

Table 24.4: Birthweights in clinical trial for drug to prevent low birthweights

Treatment	Birthweight
Drug A	6.9
Drug A	7.6
Drug A	7.3
Drug A	7.6
Drug A	6.8
Drug A	7.2
Drug A	8.0
Drug A	5.5
Drug A	5.8
Drug A	7.3

Treatment	Birthweight
Drug A	8.2
Drug A	6.9
Drug A	6.8
Drug A	5.7
Drug A	8.6
Control	6.4
Control	6.7
Control	5.4
Control	8.2
Control	5.3
Control	6.6
Control	5.8
Control	5.7
Control	6.2
Control	7.1
Control	7.0
Control	6.9
Control	5.6
Control	4.2
Control	6.8

The treatment group averaged .82 pounds more than the control group (Table 24.5):

Table 24.5: Mean birthweight per group

	Mean birthweight
Control	6.26
Drug A	7.08

Here is a resampling approach to the problem:

1. If the drug has no effect, our best guess about the "universe" of birthweights is that it is composed of (say) a million each of the observed weights, all lumped together. In other words, in the absence of any other information or compelling theory, we assume that the combination of our samples is our best estimate of the universe. Hence let us write each of the birthweights on a card, and put them into a hat. Drawing them one by one and then replacing them is the operational equivalent of a very large (but equal) number of each birthweight.
2. Repeatedly draw two samples of 15 birthweights each, and check how frequently the observed difference is as large as, or larger than, the actual difference of .82 pounds.

This is the so-called bootstrap test of the null-hypothesis.

> **Note 61: Notebook: Bootstrap test of birthweight difference**
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Get treatment and control values from data file.
birth_df = pd.read_csv('data/birthweights.csv')
# Birthweidhts for Drug A participants.
treat_df = birth_df[birth_df['Treatment'] == 'Drug A']
# Birthweigts as array.
treat = np.array(treat_df['Birthweight'])
# Control birthweights.
control_df = birth_df[birth_df['Treatment'] == 'Control']
# Birthweigts as array.
control = np.array(control_df['Birthweight'])

# Actual difference.
actual_diff = np.mean(treat) - np.mean(control)
# Show the actual difference.
actual_diff
```

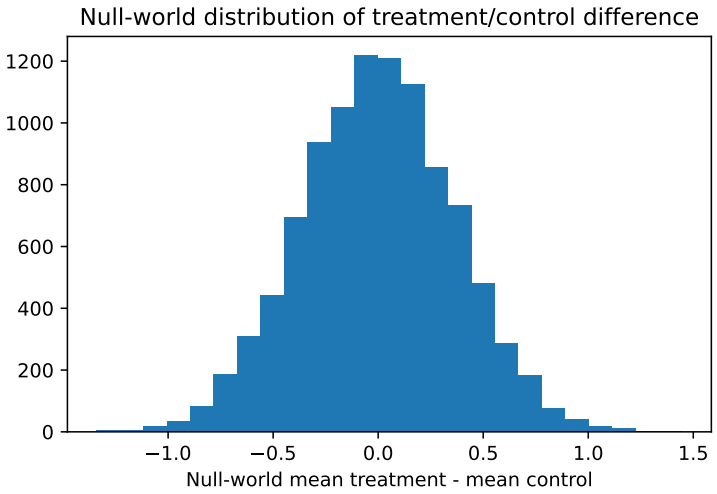
```
np.float64(0.8199999999999994)
```

Proceed with the simulation:

```
# Concatenate treatment and control arrays.
both = np.concatenate([treat, control])

# Number of trials.
n_trials = 10_000

# Make array to store results for each trial.
results = np.zeros(n_trials)
```

```
# Do 10000 simulations
for i in range(n_trials):
    # Take a resample of 15 from all birth weights.
    fake_treat = rnd.choice(both, size=15)
    # Take a second, similar resample.
    fake_control = rnd.choice(both, size=15)
    # Find the means of the two resamples.
    mt = np.mean(fake_treat)
    mc = np.mean(fake_control)
    # Find the difference between the means of the two resamples.
    diff = mt - mc
    # Keep score of the result.
    results[i] = diff
    # End the simulation experiment, go back and repeat
# Produce a histogram of the resample differences
plt.hist(results, bins=25)
plt.title('Null-world distribution of treatment/control difference')
plt.xlabel('Null-world mean treatment - mean control')
# How often did resample differences exceed the observed difference of
# .82?
k = sum(results >= actual_diff)
kk = k / n_trials
print('Proportion null-world differences >= actual difference:', kk)
```

```
Proportion null-world differences >= actual difference: 0.0114
```



Proceed with the simulation:

> ℹ End of notebook: Bootstrap test of birthweight difference
>
> `birthweight_bootstap` starts at Note 61.

Result: only about one percent of the pairs of resamples produced means that differed by as much as .82. We can conclude that the observed difference is unlikely to have occurred by chance.

24.0.4 Example: Bootstrap Sampling with Replacement

Efron and Tibshirani (1993, 11) present this as their basic problem illustrating the bootstrap method: Seven mice were given a new medical treatment intended to improve their survival rates after surgery, and nine mice were not treated. The numbers of days the treated mice survived were 94, 38, 23, 197, 99, 16 and 14, whereas the numbers of days the untreated mice (the control group) survived were 52, 10, 40, 104, 51, 27, 146, 30, and 46. The question we ask is: Did the treatment prolong survival, or might chance variation be responsible for the observed difference in mean survival times?

We start by supposing the treatment did NOT prolong survival and that chance variation in the mice was responsible for the observed difference.

If that is so, then we consider that the two groups came from the same universe. Now we'd like to know how likely it is that two groups drawn from this common universe would differ as much as the two observed groups differ.

If we had unlimited time and money, we would seek additional samples in the same way that we obtained these. Lacking time and money, we create a hypothetical universe that embodies everything we know about such a common universe. We imagine replicating each sample element millions of times to create an almost infinite universe that looks just like our samples. Then we can take resamples from this hypothetical universe and see how they behave.

Even on a computer, creating such a large universe is tedious so we use a shortcut. We replace each element after we pick it for a resample. That way, our hypothetical (bootstrap) universe is effectively infinite.

The following procedure will serve:

- **Step 1.** Calculate the difference between the means of the two observed samples – it's 30.63 days in favor of the treated mice.
- **Step 2.** Consider the two samples combined (16 observations) as the relevant universe to resample from.
- **Step 3**. Draw 7 hypothetical observations with replacement and designate them "Treatment"; draw 9 hypothetical observations with replacement and designate them "Control."

- **Step 4.** Compute and record the difference between the means of the two samples. ** Step 5.** Repeat steps 2 and 3 perhaps 10000 times. ** Step 6.** Determine how often the resampled difference exceeds the observed difference of 30.63.

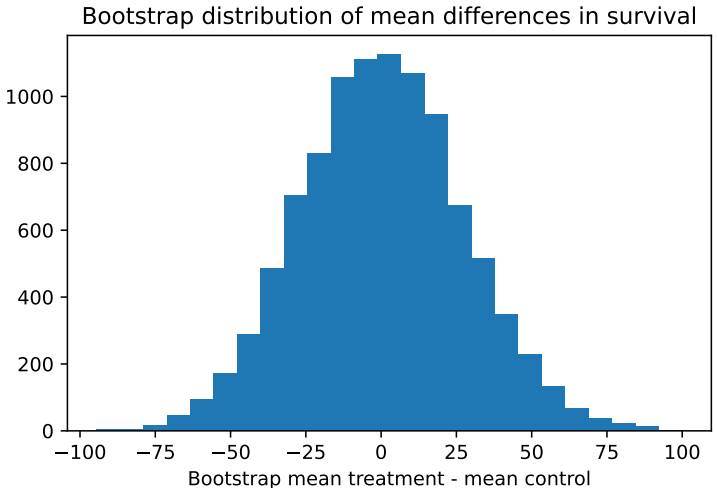
The following notebook follows the above procedure:

> Note 62: Notebook: A classic bootstrap example
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

# Set up the random number generator.
rnd = np.random.default_rng()

# Treatment group.
treat = np.array([94, 38, 23, 197, 99, 16, 141])
# control group
control = np.array([52, 10, 40, 104, 51, 27, 146, 30, 46])

# Observed difference in real world.
actual_diff = np.mean(treat) - np.mean(control)

# Set the number of trials.
n_trials = 10_000

# An empty array to store the trials.
results = np.zeros(n_trials)

# U is our universe (Step 2 above)
u = np.concatenate([treat, control])

# step 5 above.
for i in range(n_trials):
    # Step 3 above.
    fake_treat = rnd.choice(u, size=7)
    # Step 3
    fake_control = rnd.choice(u, size=9)
    # Step 4
    mt = np.mean(fake_treat)
    # Step 4
```

```
    mc = np.mean(fake_control)
    # Step 4
    diff = mt - mc
    # Step 4
    results[i] = diff

# Step 6
plt.hist(results, bins=25)
plt.title('Bootstrap distribution of mean differences in survival')
plt.xlabel('Bootstrap mean treatment - mean control')

# Step 6
k = np.sum(results >= actual_diff)
kk = k / n_trials
print('Proportion bootstrap mean difference >= actual difference:', kk)
```

```
Proportion bootstrap mean difference >= actual difference: 0.1327
```



End of notebook: A classic bootstrap example

`mouse_bootstrap` starts at Note 62.

Interpretation: 10000 simulated resamples (of sizes 7 and 9) from a combined universe produced a difference as big as 30.63 about 13.3 percent of the time. We cannot rule out the possibility that chance might be responsible for the observed advantage of the treatment group.

24.0.5 Example: Permutation Sampling Without Replacement

This section discusses at some length the question of when sampling with replacement (the bootstrap), and sampling without replacement (permutation or "exact" test) are the appropriate resampling methods. The case at hand seems like a clearcut case where the bootstrap is appropriate. (Note that in this case we draw both samples from a combined universe consisting of all observations, whether we do so with or without replacement.) Nevertheless, let us see how the technique would differ if one were to consider that the permutation test is appropriate. The algorithm would then be as follows (with the steps that are the same as above labeled "a" and those that are different labeled "b"):

- **Step 1a.** Calculate the difference between the means of the two observed samples – it's 30.63 days in favor of the treated mice.
- **Step 2a.** Consider the two samples combined (16 observations) as the relevant universe to resample from.
- **Step 3b.** Draw 7 hypothetical observations without replacement and designate them "Treatment"; draw the remaining 9 hypothetical observations and designate them "Control." (We can do this by shuffling our universe and splitting it into groups by taking the first 7 and the last 9.)
- **Step 4a.** Compute and record the difference between the means of the two samples.
- **Step 5a.** Repeat steps 2 and 3 perhaps 10,000 times
- **Step 6a.** Determine how often the resampled difference exceeds the observed difference of 30.63.

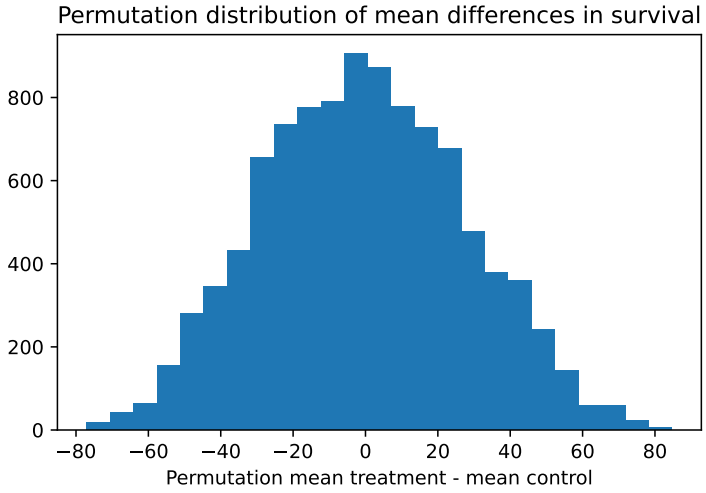
Here is the Python notebook:

> Note 63: Notebook: Permutation test for mouse survival data
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

# Set up the random number generator.
rnd = np.random.default_rng()

# Treatment group.
treat = np.array([94, 38, 23, 197, 99, 16, 141])
# control group
control = np.array([52, 10, 40, 104, 51, 27, 146, 30, 46])

# Observed difference in real world.
```

```
actual_diff = np.mean(treat) - np.mean(control)

# Set the number of trials.
n_trials = 10_000

# An empty array to store the trials.
results = np.zeros(n_trials)

# U is our universe (Step 2 above)
u = np.concatenate([treat, control])

# step 5 above.
for i in range(n_trials):
    # Step 3b above.
    shuffled = rnd.permuted(u)
    # Step 3b - take the first 7 values.
    fake_treat = shuffled[:7]
    # Step 3b - take the remaining values.
    fake_control = shuffled[7:]
    # Step 4
    mt = np.mean(fake_treat)
    # Step 4
    mc = np.mean(fake_control)
    # Step 4
    diff = mt - mc
    # Step 4
    results[i] = diff

# Step 6
plt.hist(results, bins=25)
plt.title('Permutation distribution of mean differences in survival')
plt.xlabel('Permutation mean treatment - mean control')

# Step 6
k = np.sum(results >= actual_diff)
kk = k / n_trials
print('Proportion permutation mean difference >= actual difference:', kk)
```

```
Proportion permutation mean difference >= actual difference: 0.1425
```



End of notebook: Permutation test for mouse survival data

`mouse_permutation` starts at Note 63.

Interpretation: 10000 simulated resamples (of sizes 7 and 9) from a combined universe produced a difference as big as 30.63 about 14.2 percent of the time. We therefore should not rule out the possibility that chance might be responsible for the observed advantage of the treatment group.

24.1 Differences among four means

24.1.1 Example: differences among four pig rations

Test for Differences Among Means of More Than Two Samples of Measured Data.

In the examples of Section 21.2.1 and Section 21.2.2 we investigated whether or not the results shown by a *single* sample are sufficiently different from a null (benchmark) hypothesis so that the sample is unlikely to have come from the null-hypothesis benchmark universe. In the examples of Section 21.2.5, Section 23.3.1, and Section 24.0.1 we then investigated whether or not the results shown by *two* samples suggest that both had come from the *same* universe, a universe that was assumed to be the composite of the two samples. Now, as in Section 23.2.1, we investigate whether or not *several* samples come from the same universe, except that now we work with measured data rather than with counted data.

If one experiments with each of 100 different pig foods on twelve pigs, some of the foods will show much better results than will others *just by chance*, just as one family in sixteen is likely to have the very "high" number of 4 daughters in its first four children. Therefore, it is wrong reasoning to try out the 100 pig foods, select the food that shows the best results, and then compare it statistically with the average (sum) of all the other foods (or worse, with the poorest food). With such a procedure and enough samples, you will surely find one (or more) that seems very atypical statistically. A bridge hand with 12 or 13 spades seems very atypical, too, but if you deal enough bridge hands you will sooner or later get one with 12 or 13 spades — as a purely chance phenomenon, dealt randomly from a standard deck. Therefore we need a test that prevents our falling into such traps. Such a test usually operates by taking into account the differences among *all* the foods that were tried.

The method of Section 24.0.1 can be extended to handle this problem. Assume that *four* foods were each tested on twelve pigs. The weight gains in pounds for the pigs fed on foods A and B were as before. Table 24.6 has the weight gains for foods C and D. (Compare these with the weight gains for foods A and B in Table 24.1).

Table 24.6: Measured data for pig rations C and D

	ration	weight_gain
25	C	30
26	C	30
27	C	32
28	C	31
29	C	29
30	C	27
31	C	25
32	C	30
33	C	31
34	C	32
35	C	34
36	C	33
37	D	32
38	D	25
39	D	31
40	D	26
41	D	32
42	D	27
43	D	28
44	D	29
45	D	29
46	D	28
47	D	23

	ration	weight_gain
48	D	25

Now construct a benchmark universe of *forty-eight* index cards, one for each weight gain. Then deal out sets of four hands randomly. More specifically:

- **Step 1.** Constitute a universe of the forty-eight observed weight gains in the four samples, writing the weight gains on cards.
- **Step 2.** Draw four groups of twelve weight gains, with replacement, since we are drawing from a hypothesized infinite universe in which consecutive draws are independent. Determine whether the difference between the lowest and highest group means is as large or larger than the observed difference. If so write "yes," otherwise "no."
- **Step 3.** Repeat step 2 fifty times.
- **Step 4.** Count the trials in which the differences between the simulated groups with the highest and lowest means are as large or larger than the differences between the means of the highest and lowest observed samples. The proportion of such trials to the total number of trials is the probability that all four samples would differ as much as do the observed samples if they (in technical terms) come from the same universe.

The problem, as handled by the steps given above, is quite similar to the way we handled the example in Section 23.2.2, except that the data are measured (in pounds of weight gain) rather than simply counted (the number of rehabilitations).

Instead of working through a program for the procedure outlined above, let us consider a different approach to the problem — computing the difference between each *pair* of foods, six differences in all, converting all minus (-) signs to (+) differences (taking the *absolute* value). Then we can *total* the six absolute differences, and compare the total with the sum of the six absolute differences in the *observed* sample. The proportion of the resampling trials in which the observed sample sum is equaled or exceeded by the sum of the differences in the trials is the probability that the observed samples would differ as much as they do if they come from the same universe.[5]

One naturally wonders whether this latter test statistic is better than the range, as discussed above. It would seem obvious that using the information contained in all four samples should increase the precision of the estimate. And indeed it is so, as you can confirm for yourself by comparing the results of the two approaches. But in the long run, the estimate provided by

[5]Technical Note: Computing the sum of squared differences renders this test superficially more similar to the analysis of variance but will not alter the results. This test has not been discussed in the statistical literature, to my knowledge, except perhaps for a faint suggestion at the end of Chung and Fraser (1958). This and the two-sample test can easily be performed with canned computer programs as well as Python. In addition to their advantages of nonparametricity, they are equally efficient and vastly easier to teach and to understand than the t-test and the analysis of variance. Therefore, we believe that these tests should be "treatments of choice," as the doctors say.

the two approaches would be much the same. That is, there is no reason to think that one or another of the estimates is *biased*. However, successive samples from the population would steady down faster to the true value using the four-group-based estimate than they would using the range. That is, the four-group-based estimate would require a smaller sample of pigs.

Is there reason to prefer one or the other approach from the point of view of some decision that might be made? One might think that the range procedure throws light on which *one* of the foods is best in a way that the four-group-based approach does not. But this is not correct. Both approaches answer this question, and *only* this question: Are the results from the four foods likely to have resulted from the same "universe" of weight gains or not? If one wants to know whether the *best* food is similar to, say, all the other three, the appropriate approach would be a *two-sample* approach similar to various two-sample examples discussed earlier. (It would be still *another* question to ask whether the best food is different from the worst. One would then use a procedure different from either of those discussed above.)

If the foods cost the same, one would not need even a two-sample analysis to decide which food to feed. Feed the one whose results are best in the experiment, without bothering to ask whether it is "really" the best; you can't go wrong as long as it doesn't cost more to use it. (One could inquire about the probability that the food yielding the best results in the experiment would attain those results by chance even if it was worse than the others by some stipulated amount, but pursuing that line of thought may be left to the student as an exercise.)

In the notebook below, we want a measure of how the groups differ. The obvious first step is to add up the total weight gains for each group (Table 24.7):

Table 24.7: Weight gain sum per group

	Sum of weight gains
A	382
B	344
C	364
D	335

The next step is to calculate the differences between all the possible combinations of groups.

```
differences = np.array([382 - 344, 382 - 364, 382 - 335,
                        344 - 364, 344 - 335, 364 - 335])
differences
```

```
array([ 38,  18,  47, -20,   9,  29])
```

24.2 Using Squared Differences

Here we face a choice. We could work with the absolute differences — that is, the results of the subtractions — treating each result as a positive number even if it is negative. We have seen this approach before. Therefore let us now take the opportunity of showing another approach. Instead of working with the absolute differences, we *square* each difference, and then sum the squares.

```
squared_differences = differences ** 2
squared_differences
```

```
array([1444,  324, 2209,  400,   81,  841])
```

```
sum_sq_d = np.sum(squared_differences)
sum_sq_d
```

```
np.int64(5299)
```

Squaring arrays squares *each element* of the array (see Section 16.7.1):

An advantage of working with the squares is that they are positive — a negative number squared is positive — which is convenient. Additionally, conventional statistics works mainly with squared quantities, and therefore it is worth getting familiar with that point of view. The squared differences in this case add up to 5299.

Using Python, we shuffle all the weight gains together, select four random groups, and determine whether the sum of squared differences in the resample equal or exceed 5299. If they do so with regularity, then we conclude that the observed differences could easily have occurred by chance.

With the `np.concatenate` func, we string the four arrays into a single array called `all_weights`. After shuffling (using `rnd.permuted` the 48-pig weight-gain array `all_weights` into `shuffled`, we split `shuffled` into four randomized samples. And we compute the squared differences between the pairs of groups and `sum` the squared differences just as we did above for the observed groups.

Last, we examine how often the simulated-trials data produce differences among the groups as large as (or larger than) the actually observed data — 5299.

> **Note 64: Notebook: Sum of squared differences for pig rations**
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set up the random number generator.
rnd = np.random.default_rng()

# Load data file.y
rations_df = pd.read_csv('data/pig_rations.csv')
# Show the first five rows.
rations_df.head()
```

```
  ration  weight_gain
0      A           31
1      A           34
2      A           29
3      A           26
4      A           32
```

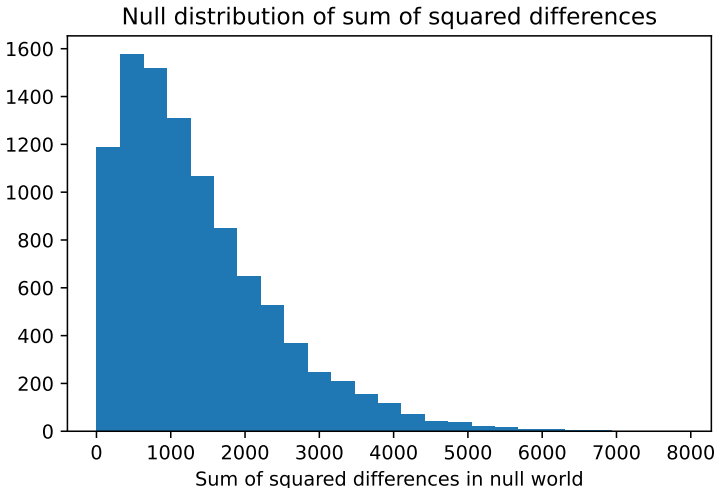
```
# Get arrays for each ration.
# A
a_rows = rations_df[rations_df['ration'] == 'A']
a_weights = np.array(a_rows['weight_gain'])
# B
b_rows = rations_df[rations_df['ration'] == 'B']
b_weights = np.array(b_rows['weight_gain'])
# C
c_rows = rations_df[rations_df['ration'] == 'C']
c_weights = np.array(c_rows['weight_gain'])
# D
d_rows = rations_df[rations_df['ration'] == 'D']
d_weights = np.array(d_rows['weight_gain'])
# Concatenate into one long array.
all_weights = np.concatenate([a_weights, b_weights, c_weights, d_weights])
# Show the concatenated array.
all_weights
```

```
array([31, 34, 29, 26, 32, 35, 38, 34, 31, 29, 32, 31, 26, 24, 28, 29, 30,
       29, 31, 29, 32, 26, 28, 32, 30, 30, 32, 31, 29, 27, 25, 30, 31, 32,
       34, 33, 32, 25, 31, 26, 32, 27, 28, 29, 29, 28, 23, 25])
```

```
n_trials = 10_000

# An array to store the result of each trial.
results = np.zeros(n_trials)

# Do 10000 trials
for i in range(n_trials):
    # Shuffle all the weight gains.
    shuffled = rnd.permuted(all_weights)
    # Split into 4 now random samples.
    fake_a = shuffled[:12]
    fake_b = shuffled[12:24]
    fake_c = shuffled[24:36]
    fake_d = shuffled[36:]
    # Sum the weight gains for the 4 resamples.
    sum_a = np.sum(fake_a)
    sum_b = np.sum(fake_b)
    sum_c = np.sum(fake_c)
    sum_d = np.sum(fake_d)
    # Find the differences between all the possible pairs of resamples.
    a_b = sum_a - sum_b
    a_c = sum_a - sum_c
    a_d = sum_a - sum_d
    b_c = sum_b - sum_c
    b_d = sum_b - sum_d
    c_d = sum_c - sum_d
    # Put the differences into an array.
    fake_diffs = np.array([a_b, a_c, a_d, b_c, b_d, c_d])
    # Square them to give six squared differences.
    sq_fake_diffs = fake_diffs ** 2
    # Sum the squares.
    sum_sq_fake_diffs = np.sum(sq_fake_diffs)
    # Keep track of the total for each trial.
    results[i] = sum_sq_fake_diffs
    # End one trial, go back and repeat until 10000 trials are complete.

# Produce a histogram of the trial results.
plt.hist(results, bins=25)
plt.title('Null distribution of sum of squared differences')
plt.xlabel('Sum of squared differences in null world')

# Find out how many trials produced differences among groups as great as
```

```
# or greater than those observed.
k = np.sum(results >= 5299)
# Convert to a proportion.
kk = k / n_trials
# Print the result.
print('Number of sum of squared differences >= 5299:', kk)
```

```
Number of sum of squared differences >= 5299: 0.0052
```



> End of notebook: Sum of squared differences for pig rations
>
> `squared_rations` starts at Note 64.

We find that our observed sum of squares — 5299 — was equaled or exceeded by randomly-drawn sums of squares in only 0.5 percent of our trials. We conclude that the four treatments are likely not all similar.

24.3 Exercises

Solutions for problems may be found in Appendix A.

24.3.1 Exercise: paired differences

The data shown in Table 24.8 might be data for the outcomes of two different mechanics, showing the length of time until the next overhaul is needed for nine pairs of similar vehicles. Or they could be two readings made by different instruments on the same sample of rock. In fact, they represent data for two successive tests for depression on a version of the Hamilton Depression Scale before and after drug therapy.[6]

Table 24.8: Hamilton Depression Scale Factor IV Values

Patient #	Score before	Score after
1	1.83	0.878
2	0.50	0.647
3	1.62	0.598
4	2.48	2.050
5	1.68	1.060
6	1.88	1.290
7	1.55	1.060
8	3.06	3.140
9	1.30	1.290

The task is to perform a test that will help decide whether there is a difference in the depression scores at the two visits (or the performances of the two mechanics). Perform both a bootstrap test and a permutation test, and give some reason for preferring one to the other in principle. How much do they differ in practice? (For this exercise, assume that we can treat the before and after scores as two different and independent samples and ignore the pairing between the before and after scores).

Here's a notebook to get you started:

[6] The data are the contents of Hollander, Wolfe, and Chicken (2013), Table 3.1. Quoting from the description there:

> The data in Table 3.1 are a portion of the data obtained by Salsburg (1970). These data, based on nine patients who received tranquilizer, were taken from a double-blind clinical trial involving two tranquilizers. The measure used was the Hamilton (1960) depression scale factor IV (the "suicidal" factor). The X (pre) was obtained at the first patient visit after initiating therapy, whereas the (post) value was obtained at the second visit after initiation of therapy.

The bibliography entry for "Salsburg (1970)" is "Personal communication (with the cooperation of Pfizer and co, Groton, Conn.)".

We could not help but notice that Hamilton (1960) did not call his fourth factor "suicidal", or anything else. He writes: "It is difficult to attach any label to the third and fourth factors, as they do not bring any clinical pattern to mind."

Note 65: Notebook: Paired differences exercise

- Download zip with notebook + data file
- Interact

```
import numpy as np
import pandas as pd

rnd = np.random.default_rng()

df = pd.read_csv('data/hamilton.csv')
before = np.array(df['score_before'])
after = np.array(df['score_after'])

# Your code here.
```

End of notebook: Paired differences exercise

`paired_differences` starts at Note 65.

See Section A.1 for a solution.

24.3.2 Exercise: seatbelt proportions

Thirty-six of 72 (.5) taxis surveyed in Pittsburgh had visible seatbelts. Seventy-seven of 129 taxis in Chicago (.597) had visible seatbelts. Calculate a confidence interval for the difference in proportions, estimated at -.097. (Source: Peskun, Peter H., "A New Confidence Interval Method Based on the Normal Approximation for the Difference of Two Binomial Probabilities," *Journal of the American Statistical Association* , 6/93 p. 656).

For solution, see Section A.2}

25 General Procedures for Testing Hypotheses

25.1 Introduction

The previous chapters have presented procedures for making statistical inferences that apply to both testing hypotheses and constructing confidence intervals: This chapter focuses on specific procedures for testing hypotheses.

'The general idea in testing hypotheses is to ask: Is there some *other* universe which might well have produced the observed sample? So we consider alternative hypotheses. This is a straightforward exercise in probability, asking about behavior of one or more universes. The choice of another universe(s) to examine depends upon purposes and other considerations.

25.2 Canonical question-and-answer procedure for testing hypotheses

25.3 Skeleton procedure for testing hypotheses

Akin to skeleton procedure for questions in probability and confidence intervals shown elsewhere

The following series of questions will be repeated below in the context of a specific inference.

What is the question? What is the purpose to be served by answering the question?

Is this a "probability" or a "statistics" question?

Assuming the Question is a Statistical Inference Question

What is the form of the statistics question?

Hypothesis test, or confidence interval, or other inference? One must first decide whether the conceptual-scientific question is of the form a) a test about the probability that some sample is likely to happen by chance rather than being very surprising (a test of a hypothesis), or b) a question about the accuracy of the estimate of a parameter of the population based upon sample evidence (a confidence interval):

Assuming the Question Concerns Testing Hypotheses

Will you state the costs and benefits of various outcomes, perhaps in the form of a "loss function"? If "yes," what are they?

How many samples of data have been observed?

One, two, more than two?

What is the description of the observed sample(s)?

Raw data?

Which characteristic(s) (parameters) of the population are of interest to you?

What are the statistics of the sample(s) that refer to this (these) characteristics(s) in which you are interested?

What comparison(s) to make?

Samples to each other?

Sample to particular universe(s)? If so, which?

What is the benchmark (null) universe?

This may include presenting the raw data and/or such summary statistics as the computed mean, median, standard deviation, range, interquartile range, other:

If there is to be a Neyman-Pearson-type alternative universe, what is it? (In most cases the answer to this technical question is "no.")

Which symbols for the observed entities?

Discrete or continuous?

What values or ranges of values?

Which sample(s) do you wish to compare to which, or to the null universe (and perhaps to the alternative universe)? (Answer: samples the same size as has been observed)

[Here one may continue with the conventional method, using perhaps a t or f or chi-square test or whatever: Everything up to now is the same whether continuing with resampling or with standard parametric test.]

What procedure will be used to produce the resampled entities?

Randomly drawn?

Simple (single step) or complex (multiple "if" drawings)?

What procedure to produce resample?

Which universe will you draw them from? With or without replacement?

What size resamples? Number of resample trials?

What to record as outcome of each resample trial?

Mean, median, or whatever of resample?

Classifying the outcomes

What is the criterion of significance to be used in evaluating the results of the test?

Stating the distribution of results

Graph of each statistic recorded — occurrences for each value.

Count the outcomes that exceed criterion and divide by number of trials.

25.4 An example: can the bio-engineer increase the female calf rate?

The question. (from (Hodges Jr and Lehmann 1970, 310): Female calves are more valuable than male calves. A bio-engineer claims to have a method that can produce more females. He tests the procedure on ten of your pregnant cows, and the result is nine females. Should you believe that his method has some effect? That is, what is the probability of a result this surprising occurring by chance?

The purpose: Female calves are more valuable than male.

Inference? Yes.

Test of hypothesis? Yes.

Will you state the costs and benefits of various outcomes (or a loss function)? We need only say that the benefits of a method that works are very large, and if the results are promising, it is worth gathering more data to confirm results.

How many samples of data are part of the significance test? One

What is the size of the first sample about which you wish to make significance statements? Ten.

What comparison(s) to make? Compare sample to benchmark universe.

What is the benchmark universe that embodies the null hypothesis? 50-50 female, or 100/206 female.

If there is to be a Neyman-Pearson alternative universe, what is it? None.

Which symbols for the observed entities? Balls in bucket, or numbers.

What values or ranges of values? 0-1, (1-100), or 101-206.

Finite or infinite? Infinite.

Which sample(s) do you wish to compare to which, or to the null universe (and perhaps to the alternative universe)? Ten calves compared to universe.

What procedure to produce entities? Sampling with replacement,

Simple (single step) or complex (multiple "if" drawings)? One can think of it either way.

What to record as outcome of each resample trial? The proportion (or number) of females.

What is the criterion to be used in the test? The probability that in a sample of ten calves, nine (or more) females would be drawn by chance from the benchmark universe of half females. (Or frame in terms of a significance level.)

"One-tail" or "two-tail" test? One tail, because the farmer is only interested in females: Finding a large proportion of males would not be of interest, and would not cause one to reject the null hypothesis.

Computation of the probability sought. The actual computation of probability may be done with several formulaic or sample-space methods, and with several resampling methods: I will first show a resampling method and then several conventional methods. The following material, which allows one to compare resampling and conventional methods, is more germane to the earlier explication of resampling taken altogether in earlier chapters than it is to the theory of hypothesis tests discussed in this chapter, but it is more expedient to present it here.

25.5 Computation of Probabilities with Resampling

We can do the problem by hand as follows:

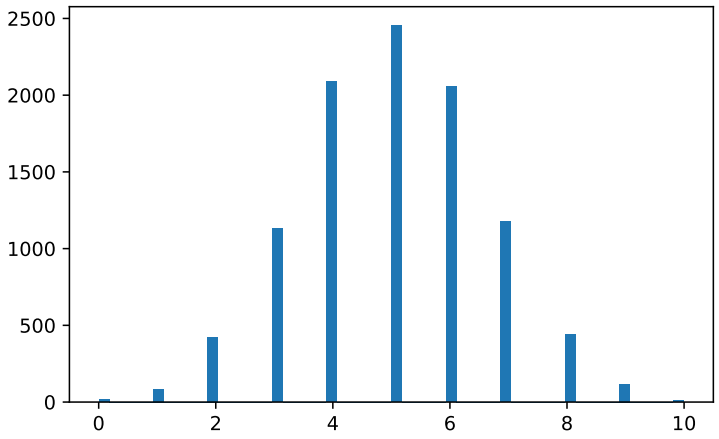
1. Constitute a bucket with either one blue and one pink ball, or 106 blue and 100 pink balls.
2. Draw ten balls with replacement, count pinks, and record.
3. Repeat step (2) say 400 times.
4. Calculate proportion of results with 9 or 10 pinks.

Or, we can take advantage of the speed and efficiency of the computer as follows:

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

n = 10000

females = np.zeros(n)
```

```
for i in range(n):
    samp = rnd.choice(['female', 'male'], size=10, replace=True)
    females[i] = np.sum(samp == 'female')

plt.hist(females, bins='auto')

k = np.sum(females >= 9)
kk = k / n
print('Proportion with >= 9 females:', kk)
```

```
Proportion with >= 9 females: 0.0127
```



This outcome implies that there is roughly a one percent chance that one would observe 9 or 10 female births in a single sample of 10 calves if the probability of a female on each birth is .5. This outcome should help the decision-maker decide about the plausibility of the bio-engineer's claim to be able to increase the probability of female calves being born.

25.6 Conventional methods

25.6.1 The Sample Space and First Principles

Assume for a moment that our problem is a smaller one and therefore much easier — the probability of getting two females in two calves if the probability of a female is .5. One could then map out what mathematicians call the "sample space," a technique that (in its simplest

form) assigns to each outcome a single point, and find the proportion of points that correspond to a "success." We list all four possible combinations — FF, FM, MF, MM. Now we look at the ratio of the number of combinations that have 2 females to the total, which is 1/4. We may then interpret this probability.

We might also use this method for (say) *five* female calves *in a row*. We can make a list of possibilities such as FFFFF, MFFFF, MMFFF, MMMFFF … MFMFM … MMMMM. There will be 2*2*2*2*2 = 32 possibilities, and 64 and 128 possibilities for six and seven calves respectively. But when we get as high as ten calves, this method would become very troublesome.

25.6.2 Sample Space Calculations

For two females in a row, we could use the well known, and very simple, multiplication rule; we could do so even for ten females in a row. But calculating the probability of nine females in ten is a bit more complex.

25.6.3 Pascal's Triangle

One can use Pascal's Triangle to obtain binomial coefficients for p = .5 and a sample size of 10, focusing on those for 9 or 10 successes. Then calculate the *proportion of the total cases* with 9 or 10 "successes" in one direction, to find the proportion of cases that pass beyond the criterion of 9 females. The method of Pascal's Triangle requires more complete understanding of the probabilistic system than does the resampling simulation described above because Pascal's Triangle requires that one understand the entire structure; simulation requires only that you follow the rules of the model.

25.6.4 The Quincunx

The quincunx — a device that filters tiny balls through a set of bumper points not unlike a pinball machine, mentioned here simply for completeness — is more a simulation method than theoretical, but it may be considered "conventional." Hence, it is included here.

25.6.5 Table of Binomial Coefficients

Pascal's Triangle becomes cumbersome or impractical with large numbers — say, 17 females of 20 births — or with probabilities other than .5. One might produce the binomial coefficients by algebraic multiplication, but that, too, becomes tedious even with small sample sizes. One can also use the pre-computed table of binomial coefficients found in any standard text. But the probabilities for n = 10 and 9 or 10 females are too small to be shown.

25.6.6 Binomial Formula

For larger sample sizes, one can use the binomial formula. The binomial formula gives no deeper understanding of the statistical structure than does the Triangle (but it does yield a deeper understanding of the pure mathematics). With very large numbers, even the binomial formula is cumbersome.

25.6.7 The Normal Approximation

When the sample size becomes too large for any of the above methods, one can then use the Normal approximation, which yields results close to the binomial (as seen very nicely in the output of the quincunx). But use of the Normal distribution requires an estimate of the standard deviation, which can be derived either by formula or by resampling. (See a more extended parallel discussion in Chapter 27 on confidence intervals for the Bush-Dukakis comparison.)

The desired probability can be obtained from the Z formula and a standard table of the Normal distribution found in every elementary text.

The Z table can be made less mysterious if we generate it with simulation, or with graph paper or Archimedes' method, using as raw material (say) five "continuous" (that is, non-binomial) distributions, many of which are skewed: 1) Draw samples of (say) 50 or 100. 2) Plot the means to see that the Normal shape is the outcome. Then 3) standardize with the standard deviation by marking the standard deviations onto the histograms.

The aim of the above exercise and the heart of the conventional parametric method is to compare the sample result — the mean — to a standardized plot of the means of samples drawn from the universe of interest to see how likely it is that that universe produces means deviating as much from the universe mean as does our observed sample mean. The steps are:

1. Establish the Normal shape — from the exercise above, or from the quincunx or Pascal's Triangle or the binomial formula or the formula for the Normal approximation or some other device.
2. Standardize that shape in standard deviations.
3. Compute the Z score for the sample mean — that is, its deviation from the universe mean in standard deviations.
4. Examine the Normal (or really, tables computed from graph paper, etc.) to find the probability of a mean deviating that far by chance.

This is the canon of the procedure for most parametric work in statistics. (For some small samples, accuracy is improved with an adjustment.)

25.7 Choice of the benchmark universe[1]

In the example of the ten calves, the choice of a benchmark universe — a universe that (on average) produces equal proportions of males and females — seems rather straightforward and even automatic, requiring no difficult judgments. But in other cases the process requires more judgments.

Let's consider another case where the choice of a benchmark universe requires no difficult judgments. Assume the U.S. Department of Labor's Bureau of Labor Statistics (BLS) takes a very large sample — say, 20,000 persons — and finds a 10 percent unemployment rate. At some later time another but smaller sample is drawn — 2,000 persons — showing an 11 percent unemployment rate. Should BLS conclude that unemployment has risen, or is there a large chance that the difference between 10 percent and 11 percent is due to sample variability? In this case, it makes rather obvious sense to ask how often a sample of 2,000 drawn from a universe of 10 percent unemployment (ignoring the variability in the larger sample) will be as different as 11 percent due solely to sample variability? This problem differs from that of the calves only in the proportions and the sizes of the samples.

Let's change the facts and assume that a very large sample had not been drawn and only a sample of 2,000 had been taken, indicating 11 percent unemployment. A policy-maker asks the probability that unemployment is above ten percent. It would still seem rather straightforward to ask how often a universe of 10 percent unemployment would produce a sample of 2000 with a proportion of 11 percent unemployed.

Still another problem where the choice of benchmark hypothesis is relatively straightforward: Say that BLS takes two samples of 2000 persons a month apart, and asks whether there is a difference in the results. Pooling the two samples and examining how often two samples drawn from the pooled universe would be as different as observed seems obvious.

One of the reasons that the above cases — especially the two-sample case — seem so clear-cut is that the variance of the benchmark hypothesis is not an issue, being implied by the fact that the samples deal with proportions. If the data were continuous, however, this issue would quickly arise. Consider, for example, that the BLS might take the same sorts of samples and ask unemployed persons the *lengths of time* they had been unemployed. Comparing a small sample to a very large one would be easy to decide about. And even comparing two small samples might be straightforward — simply pooling them as is.

But what about if you have a sample of 2,000 with data on lengths of unemployment spells with a mean of 30 days, and you are asked the probability that it comes from a universe with a mean of 25 days? Now there arises the question about the amount of variability to assume for that benchmark universe. Should it be the variability observed in the sample? That is probably an overestimate, because a universe with a smaller mean would probably have a

[1]This is one of many issues that Peter Bruce first raised, and whose treatment here reflects back-and-forth discussion between us.

smaller variance, too. So some judgment is required; there cannot be an automatic "objective" process here, whether one proceeds with the conventional or the resampling method.

The example of the comparison of liquor retailing systems in Section 24.0.2 provides more material on this subject.

25.8 Why is statistics — and hypothesis testing — so difficult?

Why is statistics such a difficult subject? The aforegoing procedural outline provides a window to the explanation. Hypothesis testing — as is also true of the construction of confidence intervals (but unlike simple probability problems) — involves a very long chain of reasoning, perhaps longer than in any other realm of systematic thinking. Furthermore, many decisions in the process require judgment that goes beyond technical analysis. All this emerges as one proceeds through the skeleton procedure above with any specific example.

(Bayes' rule also is very difficult intuitively, but that probably is a result of the twists and turns required in all complex problems in conditional probability. Decision-tree analysis is counter-intuitive, too, probably because it starts at the end instead of the beginning of the story, as we are usually accustomed to doing.)

26 Confidence Intervals, Part 1: Assessing the Accuracy of Samples

26.1 Introduction

This chapter discusses how to assess the accuracy of a point estimate of the mean, median, or other statistic of a sample. We want to know: How close is our estimate of (say) the sample mean likely to be to the population mean? The chapter begins with an intuitive discussion of the relationship between a) a statistic derived from sample data, and b) a parameter of a universe from which the sample is drawn. Then we discuss the actual construction of confidence intervals using two different approaches which produce the same numbers though they have different logic. The following chapter shows illustrations of these procedures.

The accuracy of an estimate is a hard intellectual nut to crack, so hard that for hundreds of years statisticians and scientists wrestled with the problem with little success; it was not until the last century or two that much progress was made. The kernel of the problem is learning the extent of the variation in the population. But whereas the sample mean can be used straightforwardly to estimate the population mean, the extent of variation in the sample does not directly estimate the extent of the variation in the population, because the variation differs at different places in the distribution, and there is no reason to expect it to be symmetrical around the estimate or the mean.

The intellectual difficulty of confidence intervals is one reason why they are less prominent in statistics literature and practice than are tests of hypotheses (though statisticians often favor confidence intervals). Another reason is that tests of hypotheses are more fundamental for pure science because they address the question that is at the heart of all knowledge-getting: "Should these groups be considered *different or the same* ?" The statistical inference represented by confidence limits addresses what seems to be a secondary question in most sciences (though not in astronomy or perhaps physics): "How reliable is the estimate?" Still, confidence intervals are very important in some applied sciences such as geology — estimating the variation in grades of ores, for example — and in some parts of business and industry.

Confidence intervals and hypothesis tests are not disjoint ideas. Indeed, hypothesis testing of a single sample against a benchmark value is (in all schools of thought, I believe) operationally identical with the most common way (Approach 1 below) of constructing a confidence interval and checking whether it includes that benchmark value. But the underlying *reasoning* is different for confidence limits and hypothesis tests.

The logic of confidence intervals is on shakier ground, in my judgment, than that of hypothesis testing, though there are many thoughtful and respected statisticians who argue that the logic of confidence intervals is better grounded and leads less often to error.

Confidence intervals are considered by many to be part of the same topic as *estimation*, being an estimation of accuracy, in their view. And confidence intervals and hypothesis testing are seen as sub-cases of each other by some people. Whatever the importance of these distinctions among these intellectual tasks in other contexts, they need not concern us here.

26.2 Estimating the accuracy of a sample mean

If one draws a sample that is very, very large — large enough so that one need not worry about sample size and dispersion in the case at hand — from a universe whose characteristics one *knows*, one then can *deduce* the probability that the sample mean will fall within a given distance of the population mean. Intuitively, it *seems* as if one should also be able to reverse the process — to infer something about the location of the population mean *from the sample mean*. But this inverse inference turns out to be a slippery business indeed.

Let's put it differently: It is all very well to say — as one logically may — that *on average* the sample mean (or other point estimator) equals a population parameter in most situations.

But what about the result of any *particular* sample? How accurate or inaccurate an estimate of the population mean is the sample likely to produce?

Because the logic of confidence intervals is subtle, most statistics texts skim right past the conceptual difficulties, and go directly to computation. Indeed, the topic of confidence intervals has been so controversial that some eminent statisticians refuse to discuss it at all. And when the concept is combined with the conventional algebraic treatment, the composite is truly baffling; the formal mathematics makes impossible any intuitive understanding. For students, "pluginski" is the only viable option for passing exams.

With the resampling method, however, the estimation of confidence intervals is easy. The topic then is manageable though subtle and challenging — sometimes pleasurably so. Even beginning undergraduates can enjoy the subtlety and find that it feels good to stretch the brain and get down to fundamentals.

One thing is clear: Despite the subtlety of the topic, the accuracy of estimates must be dealt with, one way or another.

I hope the discussion below resolves much of the confusion of the topic.

26.3 The logic of confidence intervals

To preview the treatment of confidence intervals presented below: We do not learn about the reliability of sample estimates of the mean (and other parameters) by logical inference from any one particular sample to any one particular universe, because this cannot be done *in principle*. Instead, we investigate the behavior of various universes in the neighborhood of the sample, universes whose characteristics are chosen on the basis of their similarity to the sample. In this way the estimation of confidence intervals is like all other statistical inference: One investigates the probabilistic behavior of one or more hypothesized universes that are implicitly suggested by the sample evidence but are not logically implied by that evidence.

The examples worked in the following chapter help explain why statistics is a difficult subject. The procedure required to transit successfully from the original question to a statistical probability, and then through a sensible interpretation of the probability, involves a great many choices about the appropriate model based on analysis of the problem at hand; a wrong choice at any point dooms the procedure. The actual computation of the probability — whether done with formulaic probability theory or with resampling simulation — is only a very small part of the procedure, and it is the least difficult part if one proceeds with resampling. The difficulties in the statistical process are not mathematical but rather stem from the hard clear thinking needed to understand the nature of the situation and to ascertain the appropriate way to model it.

Again, the purpose of a confidence interval is to help us assess the reliability of a statistic of the sample — for example, its mean or median — as an estimator of the parameter of the universe. The line of thought runs as follows: It is possible to map the distribution of the means (or other such parameter) of samples of any given size (the size of interest in any investigation usually being the size of the observed sample) and of any given pattern of dispersion (which we will assume for now can be estimated from the sample) that a universe in the neighborhood of the sample will produce. For example, we can compute how large an interval to the right and left of a postulated universe's mean is required to include 45 percent of the samples on either side of the mean.

What *cannot be done* is to draw conclusions from sample evidence about the nature of the universe from which it was drawn, in the absence of *some information* about the set of universes from which it *might* have been drawn. That is, one can investigate the behavior of one or more specified universes, and discover the absolute and relative probabilities that the given *specified* universe(s) *might produce* such a sample. But the universe(s) to be so investigated must be specified in advance (which is consistent with the Bayesian view of statistics). To put it differently, we can employ probability theory to learn the pattern(s) of results produced by samples drawn from a particular specified universe, and then compare that pattern to the observed sample. But we cannot infer the probability that that sample was drawn from any given universe in the absence of knowledge of the other possible sources of the sample. That is a subtle difference, I know, but I hope that the following discussion makes it understandable.

26.4 Computing confidence intervals

In the first part of the discussion we shall leave aside the issue of estimating the extent of the dispersion — a troublesome matter, but one which seldom will result in unsound conclusions even if handled crudely. To start from scratch again: The first — and seemingly straightforward — step is to estimate the mean of the population based on the sample data. The next and more complex step is to ask about the range of values (and their probabilities) that the estimate of the mean might take — that is, the construction of confidence intervals. It seems natural to assume that if our best guess about the population mean is the value of the sample mean, our best guesses about the various values that the population mean might take if unbiased sampling error causes discrepancies between population parameters and sample statistics, should be values clustering around the sample mean in a symmetrical fashion (assuming that asymmetry is not forced by the distribution — as for example, the binomial is close to symmetric near its middle values). But *how far away* from the sample mean might the population mean be?

Let's walk slowly through the logic, going back to basics to enhance intuition. Let's start with the familiar saying, "The apple doesn't fall far from the tree." Imagine that you are in a very hypothetical place where an apple tree is above you, and you are not allowed to look up at the tree, whose trunk has an infinitely thin diameter. You see an apple on the ground. You must now guess where the trunk (center) of the tree is. The obvious guess for the location of the trunk is right above the apple. But the trunk is not likely to be *exactly* above the apple because of the small probability of the trunk being at *any* particular location, due to sampling dispersion.

Though you find it easy to make a best guess about where the mean is (the true trunk), with the given information alone you have no way of making an estimate of the *probability* that the mean is one place or another, other than that the probability is the same that the tree is to the north or south, east or west, of you. You have no idea about *how* far the center of the tree is from you. You cannot even put a maximum on the distance it is from you, and without a maximum you could not even reasonably assume a rectangular distribution, or a Normal distribution, or any other.

Next you see two apples. What guesses do you make now? The midpoint between the two obviously is your best guess about the location of the center of the tree. But still there is no way to estimate the probability distribution of the location of the center of the tree.

Now assume you are given still another piece of information: The outermost spread of the tree's branches (the range) equals the distance between the two apples you see. With this information, you could immediately locate the *boundaries* of the location of the center of the tree. But this is only because the answer you sought was given to you in disguised form.

You could, however, come up with some statements of *relative* probabilities. In the absence of prior information on where the tree might be, you would offer higher odds that the center (the trunk) is in any unit of area close to the center of your two apples than in a unit of area far from the center. That is, if you are told that either one apple, or two apples, came from

one of two specified trees whose locations are given, with no reason to believe it is one tree or the other (later, we can put other prior probabilities on the two trees), and you are also told the dispersions, you now can put *relative* probabilities on *one tree or the other* being the source. (Note to the advanced student: This is like the Neyman-Pearson procedure, and it is easily reconciled with the Bayesian point of view to be explored later. One can also connect this concept of relative probability to the Fisherian concept of maximum likelihood — which is a probability relative to all others). And you could list from high to low the probabilities for each unit of area in the neighborhood of your apple sample. But this procedure is quite different from making any single absolute numerical probability estimate of the location of the mean.

Now let's say you see 10 apples on the ground. Of course your best estimate is that the trunk of the tree is at their arithmetic center. But *how close* to the actual tree trunk (the population mean) is your estimate likely to be? This is the question involved in confidence intervals. We want to estimate a *range* (around the center, which we estimate with the center mean of the sample, we said) within which we are pretty sure that the trunk lies.

To simplify, we consider variation along only one dimension — that is, on (say) a north-south line rather than on two dimensions (the entire surface).

We first note that you have no reason to estimate the trunk's location to be outside the sample pattern, or at its edge, though it could be so in principle.

If the pattern of the 10 apples is tight, you imagine the pattern of the likely locations of the population mean to be tight; if not, not. That is, *it is intuitively clear that there is some connection between how spread out are the sample observations and your confidence about the location of the population mean.* For example, consider two patterns of a thousand apples, one with twice the spread of another, where we measure spread by (say) the diameter of the circle that holds the inner half of the apples for each tree, or by the standard deviation. It makes sense that if the two patterns have the same center point (mean), you would put higher odds on the tree with the smaller spread being within some given distance — say, a foot — of the estimated mean. But what odds would you give on that bet?

26.5 Procedure for estimating confidence intervals

Here is a canonical list of questions that help organize one's thinking when constructing confidence intervals. The list is comparable to the lists for questions in probability and for hypothesis testing provided in earlier chapters. This set of questions will be applied operationally in Chapter 27.

What Is The Question?

What is the purpose to be served by answering the question? Is this a "probability" or a "statistics" question?

If the Question Is a Statistical Inference Question:

What is the form of the statistics question?

Hypothesis test or confidence limits or other inference?

Assuming Question Is About Confidence Limits:

What is the description of the sample that has been observed?

Raw data?

Statistics of the sample?

Which universe? Assuming that the observed sample is representative of the universe from which it is drawn, what is your best guess of the properties of the universe whose parameter you wish to make statements about? Finite or infinite? Bayesian possibilities?

Which parameter do you wish to make statements about?

Mean, median, standard deviation, range, interquartile range, other?

Which symbols for the observed entities?

Discrete or continuous?

What values or ranges of values?

If the universe is as guessed at, for which samples do you wish to estimate the variation? (Answer: samples the same size as has been observed)

Here one may continue with the conventional method, using perhaps a t or F or chi-square test or whatever. Everything up to now is the same whether continuing with resampling or with standard parametric test.

What procedure to produce the original entities in the sample?

What universe will you draw them from? Random selection?

What size resample?

Simple (single step) or complex (multiple "if" drawings)?

What procedure to produce resamples?

With or without replacement? Number of drawings?

What to record as result of resample drawing?

Mean, median, or whatever of resample

Stating the Distribution of Results

Histogram, frequency distribution, other?

Choice Of Confidence Bounds

One or two-tailed?

90%, 95%, etc.?

Computation of Probabilities Within Chosen Bounds

26.6 Summary

This chapter discussed the theoretical basis for assessing the accuracy of population averages from sample data. The following chapter shows two very different approaches to confidence intervals, and provides examples of the computations.

27 Confidence Intervals, Part 2: The Two Approaches to Estimating Confidence Intervals

There are two broad conceptual approaches to the question at hand: 1) Study the probability of various distances between the sample mean and the *likeliest* population mean; and 2) study the behavior of particular *border* universes. Computationally, both approaches often yield the same result, but their interpretations differ. Approach 1 follows the conventional logic although carrying out the calculations with resampling simulation.

27.1 Approach 1: The distance between sample and population mean

If the study of probability can tell us the probability that a given population will produce a sample with a mean at a given distance x from the population mean, and if a sample is an unbiased estimator of the population, then it seems natural to turn the matter around and interpret the same sort of data as telling us the probability that the estimate of the population mean is that far from the "actual" population mean. A fly in the ointment is our lack of knowledge of the dispersion, but we can safely put that aside for now. (See below, however.)

This first approach begins by assuming that the universe that actually produced the sample has the same amount of dispersion (but not necessarily the same mean) that one would estimate from the sample. One then produces (either with resampling or with Normal distribution theory) the distribution of sample means that would occur with repeated sampling from that designated universe with samples the size of the observed sample. One can then compute the distance between the (assumed) population mean and (say) the inner 45 percent of sample means on each side of the actually observed sample mean.

The crucial step is to shift vantage points. We look from the sample to the universe, instead of *from a hypothesized universe to simulated samples* (as we have done so far). This same interval as computed above must be the relevant distance as when one looks from the sample to the universe. Putting this algebraically, we can state (on the basis of either simulation or formal calculation) that for any given population S, and for any given distance d from its

mean μ, that $P((\mu - \bar{x}) < d) = \alpha$, where $\bar{x}$ is a randomly generated sample mean and α is the probability resulting from the simulation or calculation.

The above equation focuses on the deviation of various sample means ($\bar{x}$) from a stated population mean (μ). But we are logically entitled to read the algebra in another fashion, focusing on the deviation of μ from a randomly generated sample mean. This implies that for any given randomly generated sample mean we observe, the same probability (α) describes the probability that μ will be at a distance d or less from the observed $\bar{x}$. (I believe that this is the logic underlying the conventional view of confidence intervals, but I have yet to find a clear-cut statement of it; in any case, it appears to be logically correct.)

To repeat this difficult idea in slightly different words: If one draws a sample (large enough to not worry about sample size and dispersion), one can say in advance that there is a probability p that the sample mean ($\bar{x}$) will fall within z standard deviations of the population mean (μ). One estimates the population dispersion from the sample. If there is a probability p that $\bar{x}$ is within z standard deviations of μ, then with probability p, μ must be within that same z standard deviations of $\bar{x}$. To repeat, this is, I believe, the heart of the standard concept of the confidence interval, to the extent that there is thought through consensus on the matter.

So we can state for such populations the probability that the distance between the population and sample means will be d or less. Or with respect to a given distance, we can say that the probability that the population and sample means will be that close together is p.

That is, we start by focusing on how much the sample mean diverges from the known population mean. But then — and to repeat once more this key conceptual step — we refocus our attention to *begin with the sample mean* and then discuss the probability that the population mean will be within a given distance. The resulting distance is what we call the "confidence interval."

Please notice that the distribution (universe) assumed at the beginning of this approach did not include the assumption that the distribution is centered on the sample mean or anywhere else. It is true that the sample mean is used *for purposes of reporting the location of the estimated universe mean.* But despite how the subject is treated in the conventional approach, the estimated population mean is not part of the work of constructing confidence intervals. Rather, the calculations apply in the same way to *all universes in the neighborhood of the sample* (which are assumed, for the purpose of the work, to have the same dispersion). And indeed, it must be so, because the probability that the universe from which the sample was drawn is centered exactly at the sample mean is very small.

This independence of the confidence-intervals construction from the mean of the sample (and the mean of the estimated universe) is surprising at first, but after a bit of thought it makes sense.

In this first approach, as noted more generally above, we do *not* make estimates of the confidence intervals on the basis of any logical inference from any one particular sample to any one particular universe, because *this cannot be done in principle* ; it is the futile search for

this connection that for decades roiled the brains of so many statisticians and now continues to trouble the minds of so many students. Instead, we investigate the behavior of (in this first approach) the universe that has a higher probability of producing the observed sample than does any other universe (in the absence of any additional evidence to the contrary), and whose characteristics are chosen on the basis of its resemblance to the sample. In this way the estimation of confidence intervals is like all other statistical inference: One investigates the probabilistic behavior of one or more hypothesized universes, the universe(s) being implicitly suggested by the sample evidence but not logically implied by that evidence. And there are no grounds for dispute about exactly what is being done — only about how to interpret the results.

One difficulty with the above approach is that the estimate of the population *dispersion* does not rest on sound foundations; this matter will be discussed later, but it is not likely to lead to a seriously misleading conclusion.

A second difficulty with this approach is in interpreting the result. What is the justification for focusing our attention on a universe centered on the sample mean? While this particular universe may be more likely than any other, it undoubtedly has a low probability. And indeed, the statement of the confidence intervals refers to the probabilities that the sample has come from universes *other than* the universe centered at the sample mean, and quite a distance from it.

My answer to this question does not rest on a set of meaningful mathematical axioms, and I assert that a meaningful axiomatic answer is impossible in principle. Rather, I reason that we should consider the behavior of this universe because other universes near it will produce much the same results, differing only in dispersion from this one, and this difference is not likely to be crucial; this last assumption is all-important, of course. True, we do not know what the dispersion might be for the "true" universe. But elsewhere (Simon, forthcoming) I argue that the concept of the "true universe" is not helpful — or maybe even worse than nothing — and should be forsworn. And we can postulate a dispersion for any *other* universe we choose to investigate. That is, for this postulation we unabashedly bring in any other knowledge we may have. The defense for such an almost-arbitrary move would be that this is a second-order matter relative to the location of the estimated universe mean, and therefore it is not likely to lead to serious error. (This sort of approximative guessing sticks in the throats of many trained mathematicians, of course, who want to feel an unbroken logic leading backwards into the mists of axiom formation. But the axioms themselves inevitably are chosen arbitrarily just as there is arbitrariness in the practice at hand, though the choice process for axioms is less obvious and more hallowed by having been done by the masterminds of the past. (See Simon (1998), on the necessity for judgment.) The absence of a sequence of equations leading from some first principles to the procedure described in the paragraph above is evidence of what is felt to be missing by those who crave logical justification. The key equation in this approach is formally unassailable, but it seems to come from nowhere.)

In the examples in the following chapter may be found computations for two population distributions — one binomial and one quantitative — of the histograms of the sample means

produced with this procedure.

Operationally, we use the observed sample mean, together with an estimate of the dispersion from the sample, to estimate a mean and dispersion for the population. Then with reference to the sample mean we state a combination of a distance (on each side) and a probability pertaining to the population mean. The computational examples will illustrate this procedure.

Once we have obtained a numerical answer, we must decide how to interpret it. There is a natural and almost irresistible tendency to talk about the probability that the mean of the universe lies within the intervals, but this has proven confusing and controversial. Interpretation in terms of a repeated process is not very satisfying intuitively.[1]

In my view, it is not worth arguing about any "true" interpretation of these computations. One could sensibly interpret the computations in terms of the odds a decision maker, given

[1] An example of this sort of interpretation is as follows. Let us say the actual (unknown) population mean is some number we will call μ. We have a specific sample, and that sample has a mean we will call $\bar{x}$. The mean $\bar{x}$ that we happen to observe is almost certain to be a bit high or a bit low. Accordingly, if we want to be reasonably confident that our inference is correct, we cannot claim that μ is precisely equal to the observed $\bar{x}$. Instead, we must construct an interval estimate or confidence interval of the form: $\mu = \bar{x} \pm$ sampling error.

The crucial question is: How wide must this allowance for sampling error be? The answer, of course, will depend on how much $\bar{x}$ fluctuates…

> Constructing 95% confidence intervals is like pitching horseshoes. In each there is a fixed target, either the population μ or the stake. We are trying to bracket it with some chancy device, either the random interval or the horseshoe. …
>
> There are two important ways, however, that confidence intervals differ from pitching horseshoes. First, only one confidence interval is customarily constructed. Second, the target μ is not visible like a horseshoe stake. Thus, whereas the horseshoe player always knows the score (and specifically, whether or not the last toss bracketed the stake), the statistician does not. He continues to "throw in the dark," without knowing whether or not a specific interval estimate has bracketed μ. All he has to go on is the statistical theory that assures him that, in the long run, he will succeed 95% of the time. Wonnacott and Wonnacott (1990), (p. 258).

Savage refers to this type of interpretation as follows: (Savage 1972, 260–61):

> … whenever its advocates talk of making assertions that have high probability, whether in connection with testing or estimation, they do not actually make such assertions themselves, but endlessly pass the buck, saying in effect, "This assertion has arisen according to a system that will seldom lead you to make false assertions, if you adopt it. As for myself, I assert nothing but the properties of the system."

Lee writes at greater length (Lee 2012, preface to the first edition):

> … the statement that a 95% confidence interval for an unknown parameter ran from -2 to +2 sounded as if the parameter lay in that interval with 95% probability and yet I was warned that all I could say was that if I carried out similar procedures time after time then the unknown parameters would lie in the confidence intervals I constructed 95% of the time. …
>
> Subsequently, I discovered that the whole theory had been worked out in very considerable detail in such books as Lehmann (1959 and 1986). But attempts such as those that Lehmann describes to put everything on a firm foundation raised even more questions.

the evidence, would reasonably offer about the relative probabilities that the sample came from one of two specified universes (one of them probably being centered on the sample); this does provide some information on reliability, but this procedure departs from the concept of confidence intervals.

27.1.1 Example: Counted Data: The Accuracy of Political Polls

Consider the reliability of a randomly selected 1988 presidential election poll, showing 840 intended votes for Bush and 660 intended votes for Dukakis out of 1500 (Wonnacott and Wonnacott 1990, 5). Let us work through the logic of this example.

- **What is the question?** Stated technically, what are the 95% confidence limits for the proportion of Bush supporters in the population? (The proportion is the mean of a binomial population or sample, of course.) More broadly, within which bounds could one confidently believe that the population proportion was likely to lie? At this stage of the work, we must already have translated the conceptual question (in this case, a decision-making question from the point of view of the candidates) into a statistical question. (See Chapter 20 on translating questions into statistical form.)
- **What is the purpose** to be served by answering this question? There is no sharp and clear answer in this case. The goal could be to satisfy public curiosity, or strategy planning for a candidate (though a national proportion is not as helpful for planning strategy as state data would be). A secondary goal might be to help guide decisions about the sample size of subsequent polls.
- **Is this a "probability" or a "probability-statistics" question?** The latter; we wish to infer from sample to population rather than the converse.
- **Given that this is a statistics question: What is the form of the statistics question — confidence limits or hypothesis testing?** Confidence limits.
- **Given that the question is about confidence limits: What is the description of the sample that has been observed?** a) The raw sample data — the observed numbers of interviewees are 840 for Bush and 660 for Dukakis — constitutes the best description of the universe. The *statistics of the sample* are the given proportions — 56 percent for Bush, 44 percent for Dukakis.
- **Which universe?** (Assuming that the observed sample is representative of the universe from which it is drawn, what is your *best guess about the properties* of the universe about whose parameter you wish to make statements? The best guess is that the population proportion is the sample proportion — that is, the population contains 56 percent Bush votes, 44 percent Dukakis votes.
- **Possibilities for Bayesian analysis?** Not in this case, unless you believe that the sample was biased somehow.
- **Which parameter(s) do you wish to make statements about?** Mean, median, standard deviation, range, interquartile range, other? We wish to estimate the proportion in favor of Bush (or Dukakis).

- **Which symbols for the observed entities?** Perhaps 56 green and 44 yellow balls, if a bucket is used, or "0" and "1" if the computer is used.
- **Discrete or continuous distribution?** In principle, discrete. (*All* distributions must be discrete *in practice*.)
- **What values or ranges of values?*** "0" or "1."
- **Finite or infinite?** Infinite — the sample is small relative to the population.
- **If the universe is what you guess it to be, for which samples do you wish to estimate the variation?** A sample the same size as the observed poll.

Here one may continue either with resampling or with the conventional method. Everything done up to now would be the same whether continuing with resampling or with a standard parametric test.

27.2 Conventional Calculational Methods

Estimating the Distribution of Differences Between Sample and Population Means With the Normal Distribution.

In the conventional approach, one could in principle work from first principles with lists and sample space, but that would surely be too cumbersome. One could work with binomial proportions, but this problem has too large a sample for tree-drawing and quincunx techniques; even the ordinary textbook table of binomial coefficients is too small for this job. Calculating binomial coefficients also is a big job. So instead one would use the Normal approximation to the binomial formula.

(Note to the beginner: The distribution of means that we manipulate has the Normal shape because of the operation of the Law of Large Numbers (The Central Limit theorem). Sums and averages, when the sample is reasonably large, take on this shape even if the underlying distribution is not Normal. This is a truly astonishing property of randomly drawn samples — the distribution of their means quickly comes to resemble a "Normal" distribution, no matter the shape of the underlying distribution. We then standardize it with the standard deviation or other devices so that we can state the probability distribution of the sampling error of the mean for any sample of reasonable size.)

The exercise of creating the Normal shape *empirically* is simply a generalization of particular cases such as we will later create here for the poll by resampling simulation. One can also go one step further and use the formula of de Moivre-Laplace-Gauss to describe the empirical distributions, and to serve instead of the empirical distributions. Looking ahead now, the difference between resampling and the conventional approach can be said to be that in the conventional approach we simply plot the Gaussian distribution very carefully, and use a formula instead of the empirical histograms, afterwards putting the results in a standardized table so that we can read them quickly without having to recreate the curve each time we use it. More about the nature of the Normal distribution may be found in Simon (forthcoming).

All the work done above uses the information specified previously — the sample size of 1500, the drawing with replacement, the observed proportion as the criterion.

27.3 Confidence Intervals Empirically — With Resampling

Estimating the Distribution of Differences Between Sample and Population Means By Resampling

- **What procedure to produce entities?**: Random selection from bucket or computer.
- **Simple (single step) or complex (multiple "if" drawings)?**: Simple.
- **What procedure to produce resamples?** That is, with or without replacement? With replacement.
- **Number of drawings observations in actual sample, and hence, number of drawings in resamples?** 1500.
- **What to record as result of each resample drawing?** Mean, median, or whatever of resample? The proportion is what we seek.
- **Stating the distribution of results :** The distribution of proportions for the trial samples.
- **Choice of confidence bounds? :** 95%, two tails (choice made by the textbook that posed the problem).
- **Computation of probabilities within chosen bounds :** Read the probabilistic result from the histogram of results.
- **Computation of upper and lower confidence bounds:** Locate the values corresponding to the 2.5^{th} and 97.5^{th} percentile of the resampled proportions.

Because the theory of confidence intervals is so abstract (even with the resampling method of computation), let us now walk through this resampling demonstration slowly, using the conventional Approach 1 described previously. We first produce a sample, and then see how the process works in reverse to estimate the reliability of the sample, using the Bush-Dukakis poll as an example. The computer program follows below.

- **Step 1:** Draw a sample of 1500 voters from a universe that, based on the observed sample, is 56 percent for Bush, 44 percent for Dukakis. The first such sample produced by the computer happens to be 53 percent for Bush; it might have been 58 percent, or 55 percent, or very rarely, 49 percent for Bush.
- **Step 2:** Repeat step 1 perhaps 400 or 1000 times.
- **Step 3:** Estimate the distribution of means (proportions) of samples of size 1500 drawn from this 56-44 percent Bush- Dukakis universe; the resampling result is shown below.
- **Step 4:** In a fashion similar to what was done in steps 13, now compute the 95 percent confidence intervals for some *other* postulated universe mean — say 53% for Bush, 47% for Dukakis. This step produces a confidence interval that is not centered on the sample mean and the estimated universe mean, and hence it shows the independence of the

procedure from that magnitude. And we now compare the breadth of the estimated confidence interval generated with the 53-47 percent universe against the confidence interval derived from the corresponding distribution of sample means generated by the "true" Bush-Dukakis population of 56 percent — 44 percent. If the procedure works well, the results of the two procedures should be similar.

Now we interpret the results using this first approach. The histogram shows the probability that the difference between the sample mean and the population mean — the error in the sample result — will be about 2.5 percentage points too low. It follows that about 47.5 percent (half of 95 percent) of the time, a sample like this one will be between the population mean and 2.5 percent too low. We do not know the actual population mean. But for any observed sample like this one, we can say that there is a 47.5 percent chance that the distance between it and the mean of the population that generated it is minus 2.5 percent or less.

Now a crucial step: We turn around the statement just above, and say that there is an 47.5 percent chance that the population mean is less than three percentage points higher than the mean of a sample drawn like this one, but at or above the sample mean. (And we do the same for the other side of the sample mean.) So to recapitulate: We observe a sample and its mean. We estimate the error by experimenting with one or more universes in that neighborhood, and we then give the probability that the population mean is within that margin of error from the sample mean.

27.3.1 Example: Measured Data Example — the Bootstrap

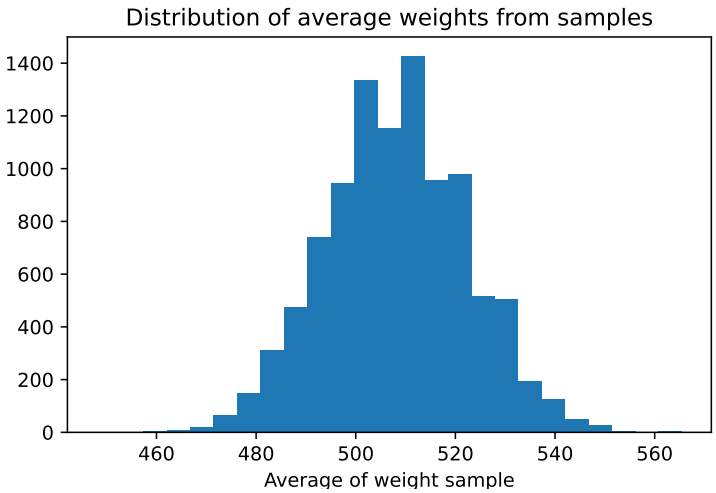
A feed merchant decides to experiment with a new pig ration — ration A — on twelve pigs. To obtain a random sample, he provides twelve customers (selected at random) with sufficient food for one pig. After 4 weeks, the 12 pigs experience an average gain of 508 ounces. The weight gain of the individual pigs are as follows: 496, 544, 464, 416, 512, 560, 608, 544, 480, 466, 512, 496.

The merchant sees that the ration produces results that are quite variable (from a low of 466 ounces to a high of 560 ounces) and is therefore reluctant to advertise an average weight gain of 508 ounces. He speculates that a different sample of pigs might well produce a different average weight gain.

Unfortunately, it is impractical to sample additional pigs to gain additional information about the universe of weight gains. The merchant must rely on the data already gathered. How can these data be used to tell us more about the sampling variability of the average weight gain?

Recalling that all we know about the universe of weight gains is the sample we have observed, we can replicate that sample millions of times, creating a "pseudo-universe" that embodies all our knowledge about the real universe. We can then draw additional samples from this pseudo-universe and see how they behave.

More specifically, we replicate each observed weight gain millions of times — we can imagine writing each result that many times on separate pieces of paper — then shuffle those weight gains and pick out a sample of 12. Average the weight gain for that sample, and record the result. Take repeated samples, and record the result for each. We can then make a histogram of the results; it might look something like this:



Though we do not know the true average weight gain, we can use this histogram to estimate the bounds within which it falls. The merchant can consider various weight gains for advertising purposes, and estimate the probability that the true weight gain falls below the value. For example, he might wish to advertise a weight gain of 500 ounces. Examining the histogram, we see that about 36% of our samples yielded weight gains less than 500 ounces. The merchant might wish to choose a lower weight gain to advertise, to reduce the risk of overstating the effectiveness of the ration.

This illustrates the "bootstrap" method. By re-using our original sample many times (and using nothing else), we are able to make inferences about the population from which the sample came. This problem would conventionally be addressed with the "t-test."

27.3.2 Example: Measured Data Example: Estimating Tree Diameters

- **What is the question?** A horticulturist is experimenting with a new type of tree. She plants 20 of them on a plot of land, and measures their trunk diameter after two years. She wants to establish a 90% confidence interval for the population average trunk diameter. For the data given below, calculate the mean of the sample and calculate (or describe a simulation procedure for calculating) a 90% confidence interval around the mean. Here are the 20 diameters, in centimeters and in no particular order (Table 27.1):

Table 27.1: Tree Diameters, in Centimeters

8.5	7.6	9.3	5.5	11.4	6.9	6.5	12.9	8.7	4.8
4.2	8.1	6.5	5.8	6.7	2.4	11.1	7.1	8.8	7.2

- **What is the purpose to be served by answering the question?** Either research & development, or pure science.
- **Is this a "probability" or a "statistics" question?** Statistics.
- **What is the form of the statistics question?** Confidence limits.
- **What is the description of the sample that has been observed?** The raw data as shown above.
- **Statistics of the sample ?** Mean of the tree data.
- **Which universe?** Assuming that the observed sample is representative of the universe from which it is drawn, what is your best guess about the properties of the universe whose parameter you wish to make statements about? Answer: The universe is like the sample above but much, much bigger. That is, in the absence of other information, we imagine this "bootstrap" universe as a collection of (say) one million trees of 8.5 centimeters width, one million of 7.2 centimeters, and so on. We'll see in a moment that the device of sampling with replacement makes it unnecessary for us to work with such a large universe; by replacing each element after we draw it in a resample, we achieve the same effect as creating an almost-infinite universe from which to draw the resamples. (Are there possibilities for Bayesian analysis?) No Bayesian prior information will be included.
- **Which parameter do you wish to make statements about?** The mean.
- **Which symbols for the observed entities?** Cards or computer entries with numbers 8.5…7.2, sample of an infinite size.
- **If the universe is as guessed at, for which samples do you wish to estimate the variation?** Samples of size 20.

Here one may continue with the conventional method. Everything up to now is the same whether continuing with resampling or with a standard parametric test. The information listed above is the basis for a conventional test.

Continuing with resampling:

- **What procedure will be used to produce the trial entities?** Random selection: simple (single step), not complex (multiple "if") sample drawings).

- **What procedure to produce resamples?** With replacement. As noted above, sampling with replacement allows us to forego creating a very large bootstrap universe; replacing the elements after we draw them achieves the same effect as would an infinite universe.
- **Number of drawings?** 20 trees
- **What to record as result of resample drawing?** The mean.
- **How to state the distribution of results?** See histogram.
- **Choice of confidence bounds?** 90%, two-tailed.
- **Computation of values of the resample statistic corresponding to chosen confidence bounds?** Read from histogram.

As has been discussed in Chapter 19, it often is more appropriate to work with the median than with the mean. One reason is that the median is not so sensitive to the extreme observations as is the mean. Another reason is that one need not assume a Normal distribution for the universe under study: this consideration affects conventional statistics but usually does not affect resampling, but it is worth keeping mind when a statistician is making a choice between a parametric (that is, Normal-based) and a non-parametric procedure.

27.3.3 Example: Determining a Confidence Interval for the Median Aluminum Content in Theban Jars

Data for the percentages of aluminum content in a sample of 18 ancient Theban jars (Catling and Jones 1977) are as follows, arranged in ascending order: 11.4, 13.4, 13.5, 13.8, 13.9, 14.4, 14.5, 15.0, 15.1, 15.8, 16.0, 16.3, 16.5, 16.9, 17.0, 17.2, 17.5, 19.0. Consider now putting a confidence interval around the median of 15.45 (halfway between the middle observations 15.1 and 15.8).

One may simply estimate a confidence interval around the median with a bootstrap procedure by substituting the median for the mean in the usual bootstrap procedure for estimating a confidence limit around the mean, as follows:

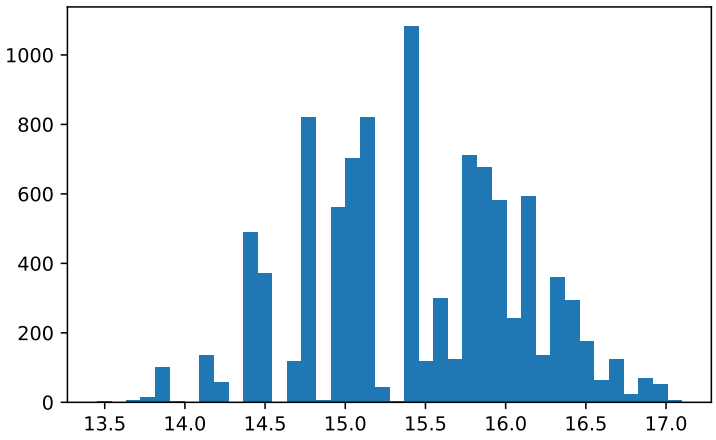
```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

data = np.array(
    [11.4, 13.4, 13.5, 13.8, 13.9, 14.4, 14.5, 15.0, 15.1, 15.8, 16.0, 16.3,
     16.5, 16.9, 17.0, 17.2, 17.5, 19.0]
)
observed_median = np.median(data)

n = 10000
```

```
medians = np.zeros(n)

for i in range(n):
    sample = rnd.choice(data, size=18, replace=True)
    # In the line above, replace=True is the default, so we could leave it out to
    # get the same result.  We added it just to emphasize that bootstrap samples
    # are samples _with_ replacement.
    medians[i] = np.median(sample)

plt.hist(medians, bins='auto')

print('Observed median aluminum content:', observed_median)
```

```
Observed median aluminum content: 15.45
```

```
pp = np.percentile(medians, (2.5, 97.5))
print('Estimate of 95 percent confidence interval:', pp)
```

```
Estimate of 95 percent confidence interval: [14.15 16.7 ]
```



(This problem would be approached conventionally with a binomial procedure leading to quite wide confidence intervals (Deshpande, Gore, and Shanubhogue 1995, 32)).

27.3.4 Example: Confidence Interval for the Median Price Elasticity of Demand for Cigarettes

The data for a measure of responsiveness of demand to a price change (the "elasticity" — percent change in demand divided by percent change in price) are shown for cigarette price changes as follows (Table 27.2). I (JLS) computed the data from cigarette sales data preceding and following a tax change in a state (Lyon and Simon 1968).

Table 27.2: Price elasticity of demand in various states at various dates

1.725	1.139	.957	.863	.802	.517	.407	.304
.204	.125	.122	.106	.031	-.032	-.1	-.142
-.174	-.234	-.240	-.251	-.277	-.301	-.302	-.302
-.307	-.328	-.329	-.346	-.357	-.376	-.377	-.383
-.385	-.393	-.444	-.482	-.511	-.538	-.541	-.549
-.554	-.600	-.613	-.644	-.692	-.713	-.724	-.734
-.749	-.752	-.753	-.766	-.805	-.866	-.926	-.971
-.972	-.975	-1.018	-1.024	-1.066	-1.118	-1.145	-1.146
-1.157	-1.282	-1.339	-1.420	-1.443	-1.478	-2.041	-2.092
-7.100							

The positive observations (implying an increase in demand when the price rises) run against all theory, but can be considered to be the result simply of measurement errors, and treated as they stand. Aside from this minor complication, the reader may work this example similarly to the case of the Theban jars. Consider this program:

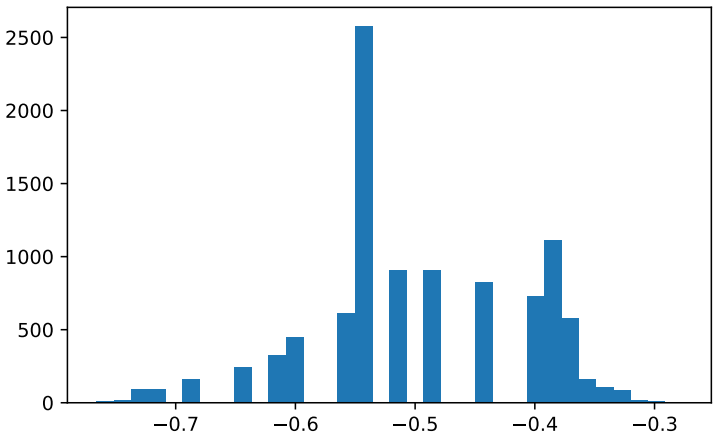
```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

data = np.array([
    1.725, 1.139, 0.957, 0.863, 0.802, 0.517, 0.407, 0.304,
    0.204, 0.125, 0.122, 0.106, 0.031, -0.032, -0.1,  -0.142,
    -0.174, -0.234, -0.240, -0.251, -0.277, -0.301, -0.302, -0.302,
    -0.307, -0.328, -0.329, -0.346, -0.357, -0.376, -0.377, -0.383,
    -0.385, -0.393, -0.444, -0.482, -0.511, -0.538, -0.541, -0.549,
    -0.554, -0.600, -0.613, -0.644, -0.692, -0.713, -0.724, -0.734,
    -0.749, -0.752, -0.753, -0.766, -0.805, -0.866, -0.926, -0.971,
    -0.972, -0.975, -1.018, -1.024, -1.066, -1.118, -1.145, -1.146,
    -1.157, -1.282, -1.339, -1.420, -1.443, -1.478, -2.041, -2.092,
    -7.100
```

```
])
data_median = np.median(data)

n = 10000

medians = np.zeros(n)

for i in range(n):
    sample = np.random.choice(data, size=73, replace=True)
    medians[i] = np.median(sample)

plt.hist(medians, bins='auto')

print('Observed median elasticity', data_median)
```

```
Observed median elasticity -0.511
```

```
pp = np.percentile(medians, (2.5, 97.5))
print('Estimate of 95 percent confidence interval', pp)
```

```
Estimate of 95 percent confidence interval [-0.692 -0.357]
```



27.4 Measured Data Example: Confidence Intervals For a Difference Between Two Means

This is another example from the mice data.

Returning to the data on the survival times of the two groups of mice in Section 24.0.4. It is the view of this book that confidence intervals should be calculated for a difference between two groups only if one is reasonably satisfied that the difference is not due to chance. Some statisticians might choose to compute a confidence interval in this case nevertheless, some because they believe that the confidence-interval machinery is more appropriate to deciding whether the difference is the likely outcome of chance than is the machinery of a hypothesis test in which you are concerned with the behavior of a benchmark or null universe. So let us calculate a confidence interval for these data, which will in any case demonstrate the technique for determining a confidence interval for a difference between two samples.

Our starting point is our estimate for the difference in mean survival times between the two samples — 30.63 days. We ask "How much might this estimate be in error? If we drew additional samples from the control universe and additional samples from the treatment universe, how much might they differ from this result?"

We do not have the ability to go back to these universes and draw more samples, but from the samples themselves we can create hypothetical universes that embody all that we know about the treatment and control universes. We imagine replicating each element in each sample millions of times to create a hypothetical control universe and (separately) a hypothetical treatment universe. Then we can draw samples (separately) from these hypothetical universes to see how reliable is our original estimate of the difference in means (30.63 days).

Actually, we use a shortcut — instead of copying each sample element a million times, we simply replace it after drawing it for our resample, thus creating a universe that is effectively infinite.

Here are the steps:

- **Step 1:** Consider the two samples separately as the relevant universes.
- **Step 2:** Draw a sample of 7 with replacement from the treatment group and calculate the mean.
- **Step 3:** Draw a sample of 9 with replacement from the control group and calculate the mean.
- **Step 4:** Calculate the difference in means (treatment minus control) & record.
- **Step 5:** Repeat steps 2-4 many times.
- **Step 6:** Review the distribution of resample means; the 5^{th} and 95^{th} percentiles are estimates of the endpoints of a 90 percent confidence interval.

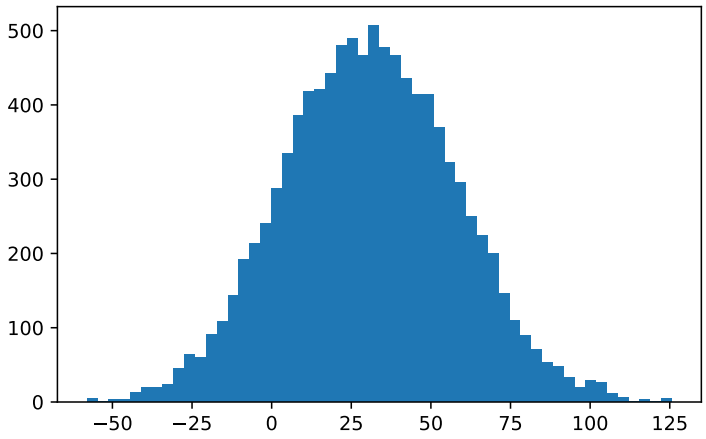
Here is a Python example:

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

treatment = np.array([94, 38, 23, 197, 99, 16, 141])
control = np.array([52, 10, 40, 104, 51, 27, 146, 30, 46])

observed_diff = np.mean(treatment) - np.mean(control)

n = 10000
mean_delta = np.zeros(n)

for i in range(n):
    treatment_sample = rnd.choice(treatment, size=7, replace=True)
    control_sample = rnd.choice(control, size=9, replace=True)
    mean_delta[i] = np.mean(treatment_sample) - np.mean(control_sample)

plt.hist(mean_delta, bins='auto')

print('Observed difference in means:', observed_diff)
```

```
Observed difference in means: 30.63492063492064
```

```
pp = np.percentile(mean_delta, (5, 95))
print('Estimate of 90 percent confidence interval:', pp)
```

```
Estimate of 90 percent confidence interval: [-12.6515873  74.7484127]
```



Interpretation: This means that one can be 90 percent confident that the mean of the difference (which is estimated to be 30.635) falls between -12.652) and 74.748). So the reliability of the estimate of the mean is very small.

27.5 Count Data Example: Confidence Limit on a Proportion, Framingham Cholesterol Data

The Framingham cholesterol data were used in Section 21.2.7 to illustrate the first classic question in statistical inference — interpretation of sample data for testing hypotheses. Now we use the same data for the other main theme in statistical inference — the estimation of confidence intervals. Indeed, the bootstrap method discussed above was originally devised for estimation of confidence intervals. The bootstrap method may also be used to calculate the appropriate sample size for experiments and surveys, another important topic in statistics.

Consider for now just the data for the sub-group of 135 high-cholesterol men in Table 21.4. Our second classic statistical question is as follows: How much confidence should we have that if we were to take a much larger sample than was actually obtained, the sample mean (that is, the proportion 10/135 = .07) would be in some close vicinity of the observed sample mean? Let us first carry out a resampling procedure to answer the questions, waiting until afterwards to discuss the logic of the inference.

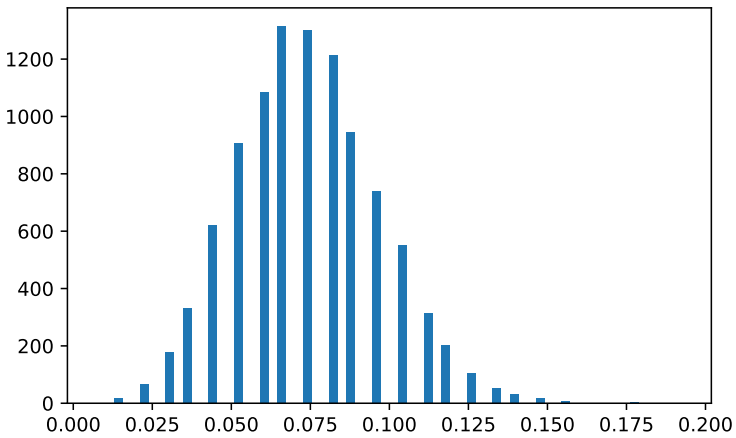
1. Construct a bucket containing 135 balls — 10 red (infarction) and 125 green (no infarction) to simulate the universe as we guess it to be.
2. Mix, choose a ball, record its color, replace it, and repeat 135 times (to simulate a sample of 135 men).
3. Record the number of red balls among the 135 balls drawn.

4. Repeat steps 2-3 perhaps 10000 times, and observe how much the total number of reds varies from sample to sample. We arbitrarily denote the boundary lines that include 47.5 percent of the hypothetical samples on each side of the sample mean as the 95 percent "confidence limits" around the mean of the actual population.

Here is a Python program:

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

men = np.repeat([1, 0], repeats=[10, 125])

n = 10000
z = np.zeros(n)

for i in range(n):
    sample = rnd.choice(men, size=135, replace=True)
    infarctions = np.sum(sample == 1)
    z[i] = infarctions / 135

plt.hist(z, bins='auto')

pp = np.percentile(z, (2.5, 97.5))
print('Estimate of 95 percent confidence interval', pp)
```

```
Estimate of 95 percent confidence interval [0.02962963 0.11851852]
```



(The result is the 95 percent confidence interval, enclosing 95 percent of the resample results)

The variation in the histogram above highlights the fact that a sample containing only 10 cases of infarction is very small, and the number of observed cases — or the proportion of cases — necessarily varies greatly from sample to sample. Perhaps the most important implication of this statistical analysis, then, is that we badly need to collect additional data.

Again, this is a classic problem in confidence intervals, found in all subject fields. The language used in the cholesterol-infarction example is exactly the same as the language used for the Bush-Dukakis poll above except for labels and numbers.

As noted above, the philosophic logic of confidence intervals is quite deep and controversial, less obvious than for the hypothesis test. The key idea is that we can estimate for any given universe the probability P that a sample's mean will fall within any given distance D of the universe's mean; we then turn this around and assume that if we know the sample mean, the probability is P that the universe mean is within distance D of it. This inversion is more slippery than it may seem. But the logic is exactly the same for the formulaic method and for resampling. The only difference is how one estimates the probabilities — either with a numerical resampling simulation (as here), or with a formula or other deductive mathematical device (such as counting and partitioning all the possibilities, as Galileo did when he answered a gambler's question about three dice). And when one uses the resampling method, the probabilistic calculations are the least demanding part of the work. One then has mental capacity available to focus on the crucial part of the job — framing the original question soundly, choosing a model for the facts so as to properly resemble the actual situation, and drawing appropriate inferences from the simulation.

27.6 Approach 2: Probability of various universes producing this sample

A second approach to the general question of estimate accuracy is to analyze the behavior of a variety of universes centered at *other* points on the line, rather than the universe centered on the sample mean. One can ask the probability that a distribution centered away from the sample mean, with a given dispersion, would produce (say) a 10-apple scatter having a mean as far away from the given point as the observed sample mean. If we assume the situation to be symmetric, we can find a point at which we can say that a distribution centered there would have only a (say) 5 percent chance of producing the observed sample. And we can also say that a distribution *even further away from the sample mean* would have an even lower probability of producing the given sample. But we cannot turn the matter around and say that there is any particular chance that the distribution that *actually produced* the observed sample is between that point and the center of the sample.

Imagine a situation where you are standing on one side of a canyon, and you are hit by a baseball, the only ball in the vicinity that day. Based on experiments, you can estimate that

a baseball thrower who you see standing on the other side of the canyon has only a 5 percent chance of hitting you with a single throw. But this does not imply that the source of the ball that hit you was someone else standing in the middle of the canyon, because that is patently impossible. That is, your knowledge about the behavior of the "boundary" universe does not logically imply anything about the existence and behavior of any other universes. But just as in the discussion of testing hypotheses, if you know that one possibility is unlikely, it is reasonable that as a result you will draw conclusions about other possibilities in the context of your general knowledge and judgment.

We can find the "boundary" distribution(s) we seek if we a) specify a measure of dispersion, and b) try every point along the line leading away from the sample mean, until we find that distribution that produces samples such as that observed with a (say) 5 percent probability or less.

To estimate the dispersion, in many cases we can safely use an estimate based on the sample dispersion, using either resampling or Normal distribution theory. The hardest cases for resampling are a) a very small sample of data, and b) a proportion near 0 or near 1.0 (because the presence or absence in the sample of a small number of observations can change the estimate radically, and therefore a large sample is needed for reliability). In such situations one should use additional outside information, or Normal distribution theory, or both.

We can also create a confidence interval in the following fashion: We can first estimate the dispersion for a universe in the general neighborhood of the sample mean, using various devices to be "conservative," if we like.[2] Given the estimated dispersion, we then estimate the probability distribution of various amounts of error between observed sample means and the population mean. We can do this with resampling simulation as follows: a) Create other universes at various distances from the sample mean, but with other characteristics similar to the universe that we postulate for the immediate neighborhood of the sample, and b) experiment with those universes. One can also apply the same logic with a more conventional parametric approach, using general knowledge of the sampling distribution of the mean, based on Normal distribution theory or previous experience with resampling. We shall not discuss the latter method here.

As with approach 1, we do not make any probability statements about where the population mean may be found. Rather, we discuss only what various hypothetical universes *might produce*, and make inferences about the "actual" population's characteristics by comparison with those hypothesized universes.

If we are interested in (say) a 95 percent confidence interval, we want to find the distribution on each side of the sample mean that would produce a sample with a mean that far away only 2.5 percent of the time (2 * .025 = 1-.95). A shortcut to find these "border distributions" is to plot the sampling distribution of the mean at the *center* of the sample, as in Approach 1. Then

[2]More about this later; it is, as I said earlier, not of primary importance in estimating the accuracy of the confidence intervals; note, please, that as we talk about the accuracy of statements about accuracy, we are moving down the ladder of sizes of causes of error.

find the (say) 2.5 percent cutoffs at each end of that distribution. On the assumption of equal dispersion at the two points along the line, we now reproduce the previously-plotted distribution with its centroid (mean) at those 2.5 percent points on the line. The new distributions will have 2.5 percent of their areas on the other side of the mean of the sample.

27.6.1 Example: Approach 2 for Counted Data: the Bush-Dukakis Poll

Let's implement Approach 2 for counted data, using for comparison the Bush-Dukakis poll data discussed earlier in the context of Approach 1.

We seek to state, for universes that we select on the basis that their results will interest us, the probability that they (or it, for a particular universe) would produce a sample as far or farther away from the mean of the universe in question as the mean of the observed sample — 56 percent for Bush. The most interesting universe is that which produces such a sample only about 5 percent of the time, simply because of the correspondence of this value to a conventional breakpoint in statistical inference. So we could experiment with various universes by trial and error to find this universe.

We can learn from our previous simulations of the Bush — Dukakis poll in Approach 1 that about 95 percent of the samples fall within .025 on either side of the sample mean (which we had been implicitly assuming is the location of the population mean). If we assume (and there seems no reason not to) that the dispersions of the universes we experiment with are the same, we will find (by symmetry) that the universe we seek is centered on those points .025 away from .56, or .535 and .585.

From the standpoint of Approach 2, then, the conventional sample formula that is centered at the mean can be considered a shortcut to estimating the boundary distributions. We say that the boundary is at the point that centers a distribution which has only a (say) 2.5 percent chance of producing the observed sample; it is that distribution which is the subject of the discussion, and not the distribution which is centered at $\mu = \bar{x}$. Results of these simulations are shown in Figure 27.1.

About these distributions centered at .535 and .585 — or more importantly for understanding an election situation, the universe centered at .535 — one can say: Even if the "true" value is as low as 53.5 percent for Bush, there is only a 2 ½ percent chance that a sample as high as 56 percent pro-Bush would be observed. (The values of a 2 ½ percent probability and a 2 ½ percent difference between 56 percent and 53.5 percent coincide only by chance in this case.) It would be even more revealing in an election situation to make a similar statement about the universe located at 50-50, but this would bring us almost entirely within the intellectual ambit of hypothesis testing.

To restate, then: Moving progressively farther away from the sample mean, we can eventually find a universe that has only some (any) specified small probability of producing a sample like

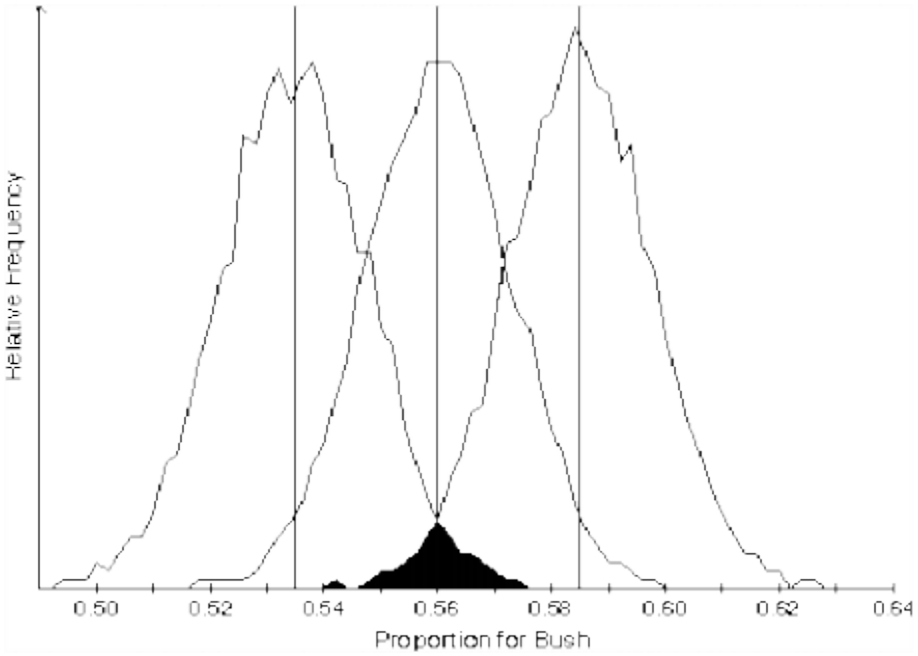


Figure 27.1: Approach 2 for Bush-Dukakis problem

the one observed. One can then say that this point represents a "limit" or "boundary" so that the interval between it and the sample mean may be called a confidence interval.

27.6.2 Example: Approach 2 for Measured Data: The Diameters of Trees

To implement Approach 2 for measured data, one may proceed exactly as with Approach 1 above except that the output of the simulation with the sample mean as midpoint will be used for guidance about where to locate trial universes for Approach 2. The results for the tree diameter data (Table 27.1) are shown in Figure 27.2.

27.7 Interpretation of Approach 2

Now to interpret the results of the second approach: Assume that the sample is not drawn in a biased fashion (such as the wind blowing all the apples in the same direction), and that the population has the same dispersion as the sample. We can then say that *distributions centered at the two endpoints of the 95 percent confidence interval (each of them including a tail in the direction of the observed sample mean with 2.5 percent of the area), or even further away from the sample mean, will produce the observed sample only 5 percent of the time or less.* The result of the second approach is more in the spirit of a hypothesis test than of

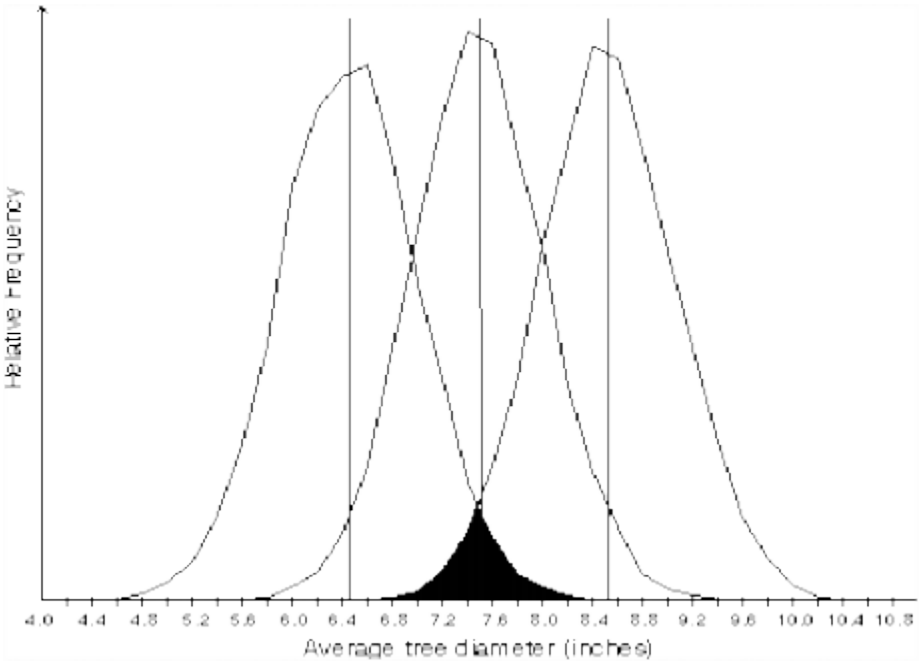


Figure 27.2: Approach 2 for tree diameters

the usual interpretation of confidence intervals. Another statement of the result of the second approach is: We postulate a given universe — say, a universe at (say) the two-tailed 95 percent boundary line. We then say: The probability that the observed sample would be produced by a universe with a mean as far (or further) from the observed sample's mean as the universe under investigation is only 2.5 percent. This is similar to the probability value interpretation of a hypothesis-test framework. It is not a direct statement about the location of the mean of the universe from which the sample has been drawn. But it is certainly reasonable to derive a betting-odds interpretation of the statement just above, to wit: The chances are 2½ in 100 (or, the odds are 2½ to 97½) that a population located here would generate a sample with a mean as far away as the observed sample. And it would seem legitimate to proceed to the further betting-odds statement that (assuming we have no additional information) the odds are 97 ½ to 2 ½ that the mean of the universe that generated this sample is no farther away from the sample mean than the mean of the boundary universe under discussion. About this statement there is nothing slippery, and its meaning should not be controversial.

Here again the tactic for interpreting the statistical procedure is to restate the facts of the behavior of the universe that we are manipulating and examining at that moment. We use a heuristic device to find a particular distribution — the one that is at (say) the 97 ½ –2 ½ percent boundary — and simply state explicitly what the distribution tells us implicitly: The probability of this distribution generating the observed sample (or a sample even further removed) is 2 ½ percent. We could go on to say (if it were of interest to us at the moment) that because the probability of this universe generating the observed sample is as low as it is,

we "reject" the "hypothesis" that the sample came from a universe this far away or further. Or in other words, we could say that because we would be very surprised if the sample were to have come from this universe, we instead believe that another hypothesis is true. The "other" hypothesis often is that the universe that generated the sample has a mean located at the sample mean or closer to it than the boundary universe.

The behavior of the universe at the 97 ½ –2 ½ percent boundary line can also be interpreted in terms of our "confidence" about the location of the mean of the universe that generated the observed sample. We can say: At this boundary point lies the end of the region within which we would bet 97 ½ to 2 ½ that the mean of the universe that generated this sample lies to the (say) right of it.

As noted in the preview to this chapter, we do not learn about the reliability of sample estimates of the population mean (and other parameters) by logical inference from any one particular sample to any one particular universe, because *in principle this cannot be done*. Instead, in this second approach we investigate the behavior of various universes at the borderline of the neighborhood of the sample, those universes being chosen on the basis of their resemblances to the sample. We seek, for example, to find the universes that would produce samples with the mean of the observed sample less than (say) 5 percent of the time. In this way the estimation of confidence intervals is like all other statistical inference: One investigates the probabilistic behavior of hypothesized universes, the hypotheses being implicitly suggested by the sample evidence but not logically implied by that evidence.

Approaches 1 and 2 may (if one chooses) be seen as identical conceptually as well as (in many cases) computationally (except for the asymmetric distributions mentioned earlier). But as I see it, the interpretation of them is rather different, and distinguishing them helps one's intuitive understanding.

27.8 Exercises

You will find solutions for problems in Appendix A.

27.8.1 Exercise: unemployment percentage

> **Note 66: Notebook: Unemployment percent exercise**
>
> - Download notebook
> - Interact

In a sample of 200 people, 7 percent are found to be unemployed. Determine a 95 percent confidence interval for the true population proportion.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Your code here.
```

End of notebook: Unemployment percent exercise

`unemployment_percent_exercise` starts at Note 66.

See {Section A.3}.

27.8.2 Exercise: battery lifetime

Note 67: Notebook: Battery lifetime exercise

- Download notebook
- Interact

A sample of 20 batteries is tested, and the average lifetime is 28.85 months. Establish a 95 percent confidence interval for the true average value. The sample values (lifetimes in months) are listed below.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

lifetimes = np.array([30, 32, 31, 28, 31, 29, 29, 24, 30, 31,
                      28, 28, 32, 31, 24, 23, 31, 27, 27, 31])

print('Mean is:', np.mean(lifetimes))
```

```
Mean is: 28.85
```

End of notebook: Battery lifetime exercise

`battery_lifetime_exercise` starts at Note 67.

See Section A.4.

27.8.3 Exercise: optical density

Note 68: Notebook: Optical density exercise

- Download notebook
- Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()
```

Suppose we have 10 measurements of Optical Density on a batch of HIV negative control samples:

```
density = np.array(
    [.02, .026, .023, .017, .022, .019, .018, .018, .017, .022])
```

Derive a 95 percent confidence interval for the sample mean. Are there enough measurements to produce a satisfactory answer?

End of notebook: Optical density exercise

`optical_density_exercise` starts at Note 68.

See: Section A.5.

28 Some Last Words About the Reliability of Sample Averages

28.1 The problem of uncertainty about the dispersion

The inescapable difficulty of estimating the amount of dispersion in the population has greatly exercised statisticians over the years. Hence I must try to clarify the matter. Yet in practice this issue turns out not to be the likely source of much error even if one is somewhat wrong about the extent of dispersion, and therefore we should not let it be a stumbling block in the way of our producing estimates of the accuracy of samples in estimating population parameters.

Student's t test was designed to get around the problem of the lack of knowledge of the population dispersion. But Wallis and Roberts wrote about the t test: "[F]ar-reaching as have been the consequences of the t distribution for technical statistics, in elementary applications it does not differ enough from the normal distribution…to justify giving beginners this added complexity." [wallis1956statistics], p. x) "Although Student's t and the F ratio are explained…the student…is advised not ordinarily to use them himself but to use the shortcut methods… These, being non-parametric and involving simpler computations, are more nearly foolproof in the hands of the beginner — and, ordinarily, only a little less powerful." (p. xi)[1]

If we knew the *population* parameter — the proportion, in the case we will discuss — we could easily determine how inaccurate the *sample* proportion is likely to be. If, for example, we wanted to know about the likely inaccuracy of the proportion of a sample of 100 voters drawn from a population of a million that is 60% Democratic, we could simply simulate drawing (say) 200 samples of 100 voters from such a universe, and examine the average inaccuracy of the 200 sample proportions.

But in fact we do *not* know the characteristics of the actual universe. Rather, the nature of the actual universe is what we seek to learn about. Of course, if the amount of variation among samples were *the same no matter what* the Republican-Democrat proportions in the universe, the issue would still be simple, because we could then estimate the average inaccuracy of the

[1] They go on to say, "Techniques and details, beyond a comparatively small range of fairly basic methods, are likely to do more harm than good in the hands of beginners…The great ideas…are lost… nonparametric [methods] … involving simpler computations are more nearly foolproof in the hands of the beginner …" (Wallis and Roberts 1956, viii, xi) Their stance is very much in contrast to that of Fisher, who wrote somewhere about the t test as a "revolution."

sample proportion for any universe and then assume that it would hold for our universe. But it is reasonable to suppose that the amount of variation among samples will be different for different Democrat-Republican proportions in the universe.

Let us first see why the amount of variation among samples drawn from a given universe is different with different relative proportions of the events in the universe. Consider a universe of 999,999 Democrats and one Republican. Most samples of 100 taken from this universe will contain 100 Democrats. A few (and only a very, very few) samples will contain 99 Democrats and one Republican. So the *biggest* possible difference between the sample proportion and the population proportion (99.9999%) is *less than one percent* (for the very few samples of 99% Democrats). And most of the time the difference will only be the tiny difference between a sample of 100 Democrats (sample proportion = 100%), and the population proportion of 99.9999%.

Compare the above to the possible difference between a sample of 100 from a universe of half a million Republicans and half a million Democrats. At worst a sample *could* be off by as much as 50% (if it got zero Republicans or zero Democrats), and at best it is unlikely to get exactly 50 of each. So it will almost always be off by 1% or more.

It seems, therefore, intuitively reasonable (and in fact it is true) that the likely difference between a sample proportion and the population proportion is greatest with a 50%-50% universe, least with a 0%-100% universe, and somewhere in between for probabilities, in the fashion of Figure 28.1.

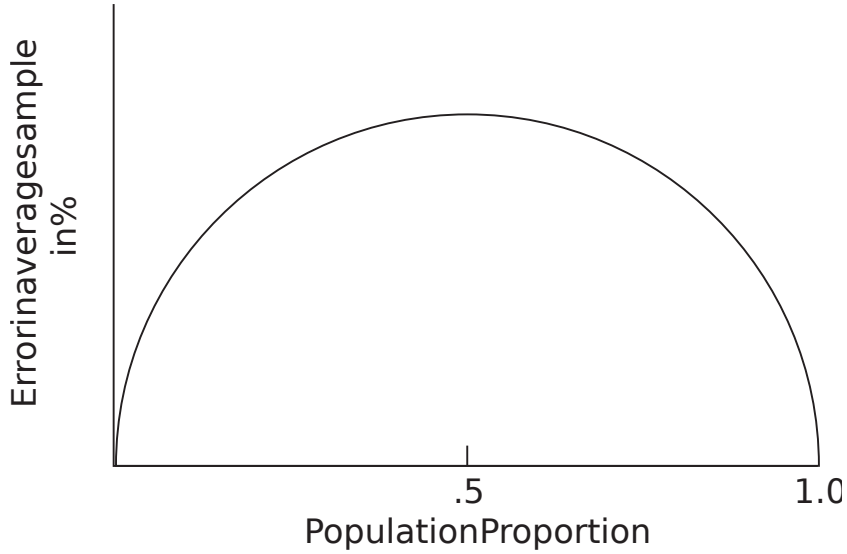


Figure 28.1: Relationship Between the Population Proportion and the Likely Error In a Sample

Perhaps it will help to clarify the issue of estimating dispersion if we consider this: If we compare estimates for a second sample based on a) the *population*, versus b) the *first sample*, the former will be more accurate than the latter, because of the sampling variation in the

first sample that affects the latter estimate. But we cannot estimate that sampling variation without knowing more about the population.

28.2 Notes on the use of confidence intervals

1. Confidence intervals are used more frequently in the physical sciences — indeed, the concept was developed for use in astronomy — than in bio-statistics and in the social sciences; in these latter fields, measurement is less often the main problem and the distinction between hypotheses often is difficult.
2. Some statisticians suggest that one can do hypothesis tests with the confidence-interval concept. But that seems to me equivalent to suggesting that one can get from New York to Chicago by flying first to Los Angeles. Additionally, the logic of hypothesis tests is much clearer than the logic of confidence intervals, and it corresponds to our intuitions so much more easily.
3. Discussions of confidence intervals sometimes assert that one cannot make a probability statement about where the population mean may be, yet can make statements about the probability that a particular set of samples may bound that mean.

If we agree that our interest is upcoming events and probably decision-making, then we obviously are interested in putting betting odds on the location of the population mean (and subsequent samples). And a statement about process will not help us with that, but only a probability statement.

Moving progressively farther away from the sample mean, we can find a universe that has only some (any) specified small probability of producing a sample like the one observed. One can say that this point represents a "limit" or "boundary" between which and the sample mean may be called a confidence interval, I suppose.

This issue is discussed in more detail in Simon (1998, published online).

28.3 Overall summary and conclusions about confidence intervals

The first task in statistics is to measure how much — to make a quantitative estimate of the universe from which a given sample has been drawn, including especially the average and the dispersion; the theory of point estimation is discussed in Chapter 19.

The next task is to make inferences about the meaning of the estimates. A hypothesis test helps us decide whether two or more universes are the same or different from each other. In contrast, the confidence interval concept helps us decide on the reliability of an estimate.

Confidence intervals and hypothesis tests are not entirely disjoint. In fact, hypothesis testing of a single sample against a benchmark value is, under all interpretations, I think, operationally

identical with constructing a confidence interval and checking whether it includes that benchmark value. But the underlying reasoning is different because the questions which they are designed to answer are different.

Having now worked through the entire procedure of producing a confidence interval, it should be glaringly obvious why statistics is such a difficult subject. The procedure is very long, and involves a very large number of logical steps. Such a long logical train is very hard to control intellectually, and very hard to follow with one's intuition. The actual computation of the probabilities is the very least of it, almost a trivial exercise.

29 Correlation and Causation

29.1 Preview

The correlation (speaking in a loose way for now) between two variables measures the strength of the relationship between them. A positive "linear" correlation between two variables x and y implies that high values of x are associated with high values of y, and that low values of x are associated with low values of y. A negative correlation implies the opposite; high values of x are associated with *low* values of y. By definition a "correlation coefficient" (Section 29.5) close to zero indicates little or no linear relationship between two variables; correlation coefficients close to 1 and -1 denote a strong positive or negative relationship. We will start by using a simpler measure of correlation than the correlation coefficient, however.

One way to measure correlation with the resampling method is to rank both variables from highest to lowest, and investigate how often in randomly-generated samples the rankings of the two variables are as close to each other as the rankings in the observed variables. A better approach, because it uses more of the quantitative information contained in the data — though it requires more computation — is to multiply the values for the corresponding pairs of values for the two variables, and compare the sum of the resulting products to the analogous sum for randomly-generated pairs of the observed variable values (Section 29.4). The last section of the chapter (Section 29.6) shows how the strength of a relationship can be determined when the data are counted, rather than measured. First comes some discussion of the philosophical issues involved in correlation and causation.

29.2 Introduction to correlation and causation

The questions in examples Section 12.1 to Section 13.3.3 have been stated in the following form: Does the independent variable (say, irradiation; or type of pig ration) have an effect upon the dependent variable (say, sex of fruit flies; or weight gain of pigs)? This is another way to state the following question: Is there a *causal relationship* between the independent variable(s) and the dependent variable? ("Independent" or "control" is the name we give to the variable(s) the researcher believes is (are) responsible for changes in the other variable, which we call the "dependent" or "response" variable.)

A causal relationship cannot be defined perfectly neatly. Even an experiment does not determine perfectly whether a relationship deserves to be called "causal" because, among other

reasons, the independent variable may not be clear-cut. For example, even if cigarette smoking experimentally produces cancer in rats, it might be the paper and not the tobacco that causes the cancer. Or consider the fabled gentlemen who got experimentally drunk on bourbon and soda on Monday night, scotch and soda on Tuesday night, and brandy and soda on Wednesday night — and stayed sober on Thursday night by drinking nothing. With a vast inductive leap of scientific imagination, they treated their experience as an empirical demonstration that soda, the common element each evening, was the cause of the inebriated state they had experienced. Notice that their deduction was perfectly sound, given only the recent evidence they had. Other knowledge of the world is necessary to set them straight. That is, even in a controlled experiment there is often no way except subject-matter knowledge to avoid erroneous conclusions about causality. Nothing except substantive knowledge or scientific intuition would have led them to the recognition that it is the alcohol rather than the soda that made them drunk, *as long as they always took soda with their drinks.* And no statistical procedure can suggest to them that they ought to experiment with the presence and absence of soda. If this is true for an experiment, it must also be true for an uncontrolled study.

Here are some tests that a relationship usually must pass to be called causal. That is, a working definition of a particular causal relationship is expressed in a statement that has these important characteristics:

1. It is an association that is strong enough so that the observer believes it to have a predictive (explanatory) power great enough to be scientifically useful or interesting. For example, he is not likely to say that wearing glasses causes (or is a cause of) auto accidents if the observed *correlation coefficient* is .07[1], even if the sample is large enough to make the correlation statistically convincing. In other words, unimportant relationships are not likely to be labeled causal.

   Various observers may well differ in judging whether or not an association is strong enough to be important and therefore "causal." And the particular field in which the observer works may affect this judgment. This is an indication that whether or not a relationship is dubbed "causal" involves a good deal of human judgment and is subject to dispute.

2. The "side conditions" must be sufficiently *few* and sufficiently observable so that the relationship will apply under a wide enough range of conditions to be considered useful or interesting. In other words, *the relationship must not require too many "if"s, "and"s, and "but"s in order to hold.* For example, one might say that an increase in income caused an increase in the birth rate if this relationship were observed everywhere. But, if the relationship were found to hold only in developed countries, among the educated classes, and among the higher-income groups, then it would be less likely to be called "causal"

[1] As we discussed in the introduction, correlation coefficient values can vary between 1 and -1, where values near 1 mean a very strong positive linear relationship (high values in one variable predict high values in the other), values near -1 mean a very strong negative linear relationship (high values in one predict low values in the other), and values near 0 mean very little linear relationship between the variables.

— even if the correlation were extremely high once the specified conditions had been met. A similar example can be made of the relationship between income and happiness.

3. For a relationship to be called "causal," there should be sound reason to believe that, even if the control variable were not the "real" cause (and it never is), other relevant "hidden" and "real" cause variables must also change *consistently* with changes in the control variables. That is, a variable being manipulated may reasonably be called "causal" if the real variable for which it is believed to be a proxy must always be tied intimately to it. (Between two variables, v and w, v may be said to be the "more real" cause and w a "spurious" cause, if v and w require the same side conditions, except that v does not require w as a side condition.) This third criterion (non-spuriousness) is of particular importance to policy makers. The difference between it and the previous criterion for side conditions is that a plenitude of very restrictive side conditions may take the relationship out of the class of causal relationships, *even though the effects of the side conditions are known*. This criterion of nonspuriousness concerns variables that are as yet *unknown and unevaluated* but that have a *possible* ability to *upset* the observed association.

   Examples of spurious relationships and hidden-third-factor causation are commonplace. For a single example, toy sales rise in December. There is no danger in saying that December causes an increase in toy sales, even though it is "really" Christmas that causes the increase, because Christmas and December practically always accompany each other.

   Belief that the relationship is not spurious is increased if *many* likely variables have been investigated and none removes the relationship. This is further demonstration that the test of whether or not an association should be called "causal" cannot be a logical one; there is no way that one can express in symbolic logic the fact that many other variables have been tried without changing the relationship in question.

4. The more tightly a relationship is bound into (that is, deduced from, compatible with, and logically connected to) a general framework of theory, the stronger is its claim to be called "causal." For an economics example, observed positive relationships between the interest rate and business investment and between profits and investment are more likely to be called "causal" than is the relationship between liquid assets and investment. This is so because the first two statements can be deduced from classical price theory, whereas the third statement cannot. Connection to a theoretical framework provides support for belief that the side conditions necessary for the statement to hold true are not restrictive and that the likelihood of spurious correlation is not great; because a statement is logically connected to the rest of the system, the statement tends to stand or fall as the rest of the system stands or falls. And, because the rest of the system of economic theory has, over a long period of time and in a wide variety of tests, been shown to have predictive power, a statement connected with it is cloaked in this mantle.

The social sciences other than economics do not have such well-developed bodies of deductive theory, and therefore this criterion of causality does not weigh as heavily in sociology, for instance, as in economics. Rather, the other social sciences seem to substitute a weaker and more

general criterion, that is, whether or not the statement of the relationship is accompanied by other statements that seem to "explain" the "mechanism" by which the relationship operates. Consider, for example, the relationship between the phases of the moon and the suicide rate. The reason that sociologists do not call it causal is that there are no auxiliary propositions that explain the relationship and describe an operative mechanism. On the other hand, the relationship between broken homes and youth crime is often referred to as "causal," presumably because a large body of psychological theory serves to explain why a child raised without one or the other parent, or in the presence of parental strife, should not adjust readily.

Furthermore, one can never decide with perfect certainty whether in any *given* situation one variable "causes" a particular change in another variable. At best, given your particular purposes in investigating a phenomena, you may be safe in judging that very likely there is causal influence.

In brief, it is correct to say (as it is so often said) that correlation does not prove causation — if we add the word "completely" to make it "correlation does not *completely* prove causation." On the other hand, causation can *never* be "proven" *completely* by correlation *or any other* tool or set of tools, including experimentation. The best we can do is make informed judgments about whether to call a relationship causal.

It is clear, however, that in any situation where we are interested in the possibility of causation, we must *at least* know whether there is a relationship (correlation) between the variables of interest; the existence of a relationship is necessary for a relationship to be judged causal even if it is not sufficient to receive the causal label. And in other situations where we are not even interested in causality, but rather simply want to predict events or understand the structure of a system, we may be interested in the existence of relationships quite apart from questions about causations. Therefore our next set of problems deals with the probability of there being a relationship between two measured variables, variables that can take on any values (say, the values on a test of athletic scores) rather than just two values (say, whether or not there has been irradiation.)[2]

Another way to think about such problems is to ask whether two variables are *independent* of each other — that is, whether you know anything about the value of one variable if you know the value of the other in a particular case — or whether they are not independent but rather are related.

29.3 A Note on Association Compared to Testing a Hypothesis

Problems in which we investigate a) whether there is an *association*, versus b) whether there is a *difference* between just two groups, often look very similar, especially when the data constitute a 2-by-2 table. There is this important difference between the two types of analysis, however: Questions about *association* refer to *variables* — say weight and age — and it never

[2]For a much fuller discussion on causality and causation, see Simon and Burstein (1985).

makes sense to ask whether there is a difference between variables (except when asking whether they measure the same quantity). Questions about *similarity or difference* refer to *groups of individuals*, and in such a situation it does make sense to ask whether or not two groups are observably different from each other.

29.3.1 Example: Is Athletic Ability Directly Related to Intelligence?

A more specific version of our question: **is there correlation between the two variables or are they independent?**

A scientist often wants to know whether or not two characteristics go together, that is, whether or not they are correlated (that is, related or associated). For example, do young adults with high athletic ability tend to also have high I.Q.s?

Hypothetical physical-education scores of a group of ten high-school boys are shown in Table 29.1, ordered from high to low, along with the I.Q. score for each boy. The ranks for each student's athletic and I.Q. scores are then shown in the third and fourth columns:

Table 29.1: Hypothetical athletic and I.Q. scores for high school boys

Athletic Score	I.Q. Score	Athletic Rank	I.Q.Rank
97	114	1	3
94	120	2	1
93	107	3	7
90	113	4	4
87	118	5	2
86	101	6	8
86	109	7	6
85	110	8	5
81	100	9	9
76	99	10	10

Figure 29.1 is a *scatterplot* with "Athletic Score" on the x-axis and "I.Q. Score" on the y-axis. Each point on the plot corresponds to one row of Table 29.1 (and therefore one boy); in particular each point is at the x, y coordinate given by the values in "Athletic Score" and "I.Q. Score". For example the point for the first boy is at position x=97, y=114.

We want to know whether a high score on athletic ability tends to be found along with a high I.Q. score more often than would be expected by chance. Therefore, our strategy is to see how often high scores on *both* variables are found by chance. We do this by disassociating the two variables and making two separate and independent universes, one composed of the athletic scores and another of the I.Q. scores. Then we draw pairs of observations from the two

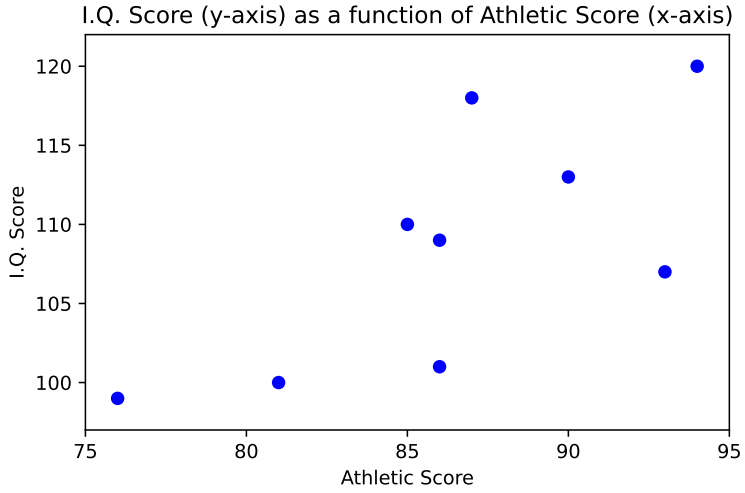


Figure 29.1: Scatter plot of I.Q. Score as a function of Athletic Score

universes at random, and compare the experimental patterns that occur by chance to what actually is observed to occur in the world.

The first testing scheme we shall use is similar to our first approach to the pig rations — splitting the results into just "highs" and "lows." We take ten cards, one of each denomination from "ace" to "10," shuffle, and deal five cards to correspond to the first five athletic ranks. The face values then correspond to the I.Q. ranks. Under the benchmark hypothesis the athletic ranks will not be associated with the I.Q. ranks. Add the face values in the first five cards in each trial; the first hand includes 2, 4, 5, 6, and 9, so the sum is 26. Record, shuffle, and repeat perhaps ten times. Then compare the random results to the sum of the observed ranks of the five top athletes, which equals 17.

The following steps describe a slightly different procedure than that just described, because this one may be easier to understand:

- **Step 1.** Convert the athletic and I.Q. scores to ranks. Then constitute a universe of spades, "ace" to "10," to correspond to the athletic ranks, and a universe of hearts, "ace" to "10," to correspond to the IQ ranks.
- **Step 2.** Deal out the well-shuffled cards into pairs, each pair with an athletic score and an I.Q. score.
- **Step 3.** Locate the cards with the top five athletic ranks, and add the I.Q. rank scores on their paired cards. Compare this sum to the observed sum of 17. If 17 or less, indicate "yes," otherwise "no." (Why do we use "17 or less" rather than "less than 17"? Because we are asking the probability of a score *this low or lower*.)
- **Step 4.** Repeat steps 2 and 3 ten times.
- **Step 5.** Calculate the proportion "yes." This estimates the probability sought.

In Table 29.2 we see that the observed sum (17) is lower than the sum of the top 5 ranks in all but one (shown in bold) of the ten random trials (trial 5), which suggests that there is a good chance (9 in 10) that the five best athletes will not have I.Q. scores that high by chance. But it might be well to deal some more to get a more reliable average. We add thirty hands, and thirty-nine of the total forty hands exceed the observed rank value, so the probability that the observed correlation of athletic and I.Q. scores would occur by chance is about .025. In other words, if there is no real association between the variables, the probability that the top 5 ranks would sum to a number this low or lower is only 1 in 40, and it therefore seems reasonable to believe that high athletic ability tends to accompany a high I.Q.

Table 29.2: 40 random trials of the athletic / IQ problem

Trial	Sum of IQ Ranks	<= observed (17)
1	26	No
2	23	No
3	22	No
4	37	No
5	**16**	**Yes**
6	22	No
7	22	No
8	28	No
9	38	No
10	22	No
11	35	No
12	36	No
13	31	No
14	29	No
15	32	No
16	25	No
17	25	No
18	29	No
19	25	No
20	22	No
21	30	No
22	31	No
23	35	No
24	25	No
25	33	No
26	30	No
27	24	No
28	29	No
29	30	No

Trial	Sum of IQ Ranks	<= observed (17)
30	31	No
31	30	No
32	21	No
33	25	No
34	19	No
35	29	No
36	23	No
37	23	No
38	34	No
39	23	No
40	26	No

In fact we can apply an even simpler procedure to get the same result, by reasoning about the individual trial.

One trial in our procedure is:

- **Step 2.** Deal out the well-shuffled cards into pairs, each pair with an athletic score and an I.Q. score.
- **Step 3.** Locate the cards with the top five athletic ranks, and add the I.Q. rank scores on their paired cards. Compare this sum to the observed sum of 17. If 17 or less, indicate "yes," otherwise "no." (Why do we use "17 or less" rather than "less than 17"? Because we are asking the probability of a score *this low or lower*.)

Now consider the 5 IQ rank cards. In the procedure above, we found these by *first* pairing the athletic ranks and the IQ ranks, *then* selecting the IQ ranks corresponding to the top 5 athletic ranks. A little thought may persuade you, that by doing this, we have have a *random selection of 5 IQ ranks*. We got that random selection by pairing, selecting on athletic rank — but the initial pairing and selection will do nothing other than giving us one particular set of randomly chosen 5 IQ rank cards. So we can simplify our procedure even further by missing out the pairing and selecting by rank steps; we can just shuffle the IQ rank cards and deal out 5 to be our randomly selected IQ ranks.

> **Note 71: Notebook: Athletic ability and IQ**
>
> - Download notebook
> - Interact

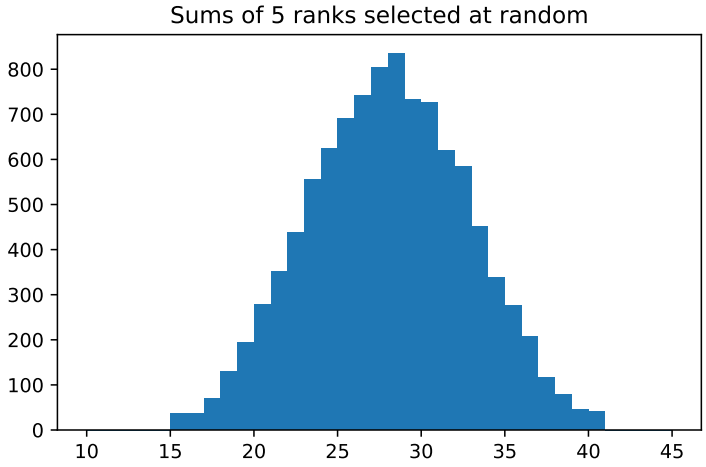
To simulate this problem in Python, we first create an array containing the I.Q. rankings of the top 5 students in athletics. The `sum` of these I.Q. rankings constitutes the observed result to be tested against randomly-drawn samples. We observe that the actual I.Q. rankings of

the top five athletes sums to 17. The more frequently that the sum of 5 randomly-generated rankings (out of 10) is as low as this observed number, the higher is the probability that there is no relationship between athletic performance and I.Q. based on these data.

First we record the 1 through 10 into array `iq_ranks`. Then we shuffle the numbers so the rankings are in a random order. Then select the first 5 of these numbers and put them in another array, `top_5`, and `sum` them, putting the result in `top_5_sum`. We repeat this procedure `N = 10000` times, recording each result in a scorekeeping vector: `z`. Graphing `z`, we get a histogram that shows us how often our randomly assigned sums are equal to or below 17.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()
```

```
# Number of repeats.
n_trials = 10000
# The IQ ranks, ready for shuffling.
iq_ranks = np.arange(1, 11)  # 1 through 10.
# Scorekeeping array.
results = np.zeros(n_trials)

# Repeat the experiment 10,000 times.
for i in range(n_trials):
    # Shuffle the ranks.
    shuffled = rnd.permuted(iq_ranks)
    # Take the first 5 ranks.
    top_5 = shuffled[:5]
    # Sum those ranks.
    top_5_sum = np.sum(top_5)
    # Keep track of the result of each trial.
    results[i] = top_5_sum
    # End the experiment, go back and repeat.

# Produce a histogram of trial results.
# Make the bins be the integers from 10 through 45.
plt.hist(results, bins=np.arange(10, 46))
plt.title('Sums of 5 ranks selected at random');
```



We see that in only about 1 % of the trials did random selection of ranks produce a total of 17 or lower. Python can calculate this for us directly:

```
# Determine how many trials produced sums of ranks <= 17 by chance.
k = np.sum(results <= 17)
# The proportion.
kk = k / n_trials
# Show the result.
kk
```

```
np.float64(0.0143)
```

> End of notebook: Athletic ability and IQ
>
> `athlete_iq` starts at Note 71.

Why do we sum the ranks of the first *five* athletes rather than taking the sum of the top three, say? Indeed, we could have looked at the top three, two, four, or even six or seven. The first reason for splitting the group in half is that an even split uses the available information more fully, and therefore we obtain greater efficiency. (I cannot prove this formally here, but perhaps it makes intuitive sense to you.) A second reason is that getting into the habit of always looking at an even split reduces the chances that you will pick and choose in such a manner as to fool yourself. For example, if the I.Q. ranks of the top five athletes were 3, 2, 1, 10, and 9, we would be deceiving ourselves if, after looking the data over, we drew the line between athletes 3 and 4. (More generally, choosing an appropriate measure before examining the data will help you avoid fooling yourself in such matters.)

A simpler but less efficient approach to this same problem is to classify the top-half athletes by whether or not they were also in the top half of the I.Q. scores. Of the first five athletes actually observed, *four* were in the top five I.Q. scores. We can then shuffle five black and five red cards and see how often four or more (that is, four or five) blacks come up with the first five cards. The proportion of times that four or more blacks occurs in the trial is the probability that an association as strong as that observed might occur by chance even if there is no association. Table 29.3 shows a proportion of five trials out of twenty.

Table 29.3: Results of 20 random trials of the top-5 rank counts

Trial	Score	Yes or No
1	4	Yes
2	2	No
3	2	No
4	2	No
5	3	No
6	2	No
7	4	Yes
8	3	No
9	3	No
10	4	Yes
11	3	No
12	1	No
13	3	No
14	3	No
15	4	Yes
16	3	No
17	2	No
18	2	No
19	2	No
20	4	Yes

Note 70: Notebook: Athletic ability and IQ using rank counts

- Download notebook
- Interact

In the Python code below, we first note that the top 5 athletes had 4 of the top 5 I.Q. scores. So we constitute the set of 10 IQ rankings array `iq_ranks`. We then shuffle `iq_ranks` and select the first 5 I.Q. rankings (out of 10). We count (`sum`) how many are in the top 5, and keep track of the result. After repeating 10,000 times using a `for` loop, we find out how often we select 4 of the top 5.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Number of repeats.
n_trials = 10000
# The IQ ranks, ready for shuffling.
iq_ranks = np.arange(1, 11)  # 1 through 10.
# Scorekeeping array.
results = np.zeros(n_trials)

for i in range(n_trials):
    # Shuffle the ranks.
    shuffled = rnd.permuted(iq_ranks)
    # Take the first 5 ranks.
    top_5 = shuffled[:5]
    # Everything up until this point is the same as the code above.
    # Here is the difference.
    # Check whether the selected IQ ranks are in the top 5.
    are_top = top_5 <= 5
    # Count how many were in the top 5
    n_are_top = np.sum(are_top)
    # Keep track of the result of each trial.
    results[i] = n_are_top
    # End the experiment, go back and repeat.

# Determine how many trials produced 4 or more top ranks by chance.
k = np.sum(results >= 4)
#  Convert to a proportion.
kk = k / n_trials
# Show the result.
kk
```

```
np.float64(0.0982)
```

> **i** End of notebook: Athletic ability and IQ using rank counts
>
> `athlete_iq_rank_count` starts at Note 70.

So far we have proceeded on the theory that if there is *any* relationship between athletics and I.Q., then the better athletes have higher rather than lower I.Q. scores. The justification for

this assumption is that past research suggests that it is probably true. But if we had *not* had the benefit of that past research, we would then have had to proceed somewhat differently; we would have had to consider the possibility that the top five athletes could have I.Q. scores either higher *or* lower than those of the other students. The results of the "two-tail" test would have yielded odds weaker than those we observed.

29.3.2 Example: Athletic ability and I.Q. — a third way

The example in Section 29.3.1 investigated the relationship between I.Q. and athletic score by ranking the two sets of scores. But ranking of scores loses some efficiency because it uses only an "ordinal" (rank-ordered) rather than a "cardinal" (measured) scale; the numerical shadings and relative relationships are lost when we convert to ranks. Therefore let us consider a test of correlation that uses the original cardinal numerical scores.

First a little background: Figure 29.2 and Figure 29.3 show two hypothetical cases of very high association among the I.Q. and athletic scores used in previous examples. Figure 29.2 indicates that the higher the I.Q. score, the higher the athletic score. With a boy's athletic score you can thus predict quite well his I.Q. score by means of a hand-drawn line — or vice versa. The same is true of Figure 29.3, but in the opposite direction. Notice that even though athletic score is on the x-axis (horizontal) and I.Q. score is on the y-axis (vertical), the athletic score does not *cause* the I.Q. score. (It is an unfortunate deficiency of such diagrams that *some* variable must arbitrarily be placed on the x-axis, whether you intend to suggest causation or not.)

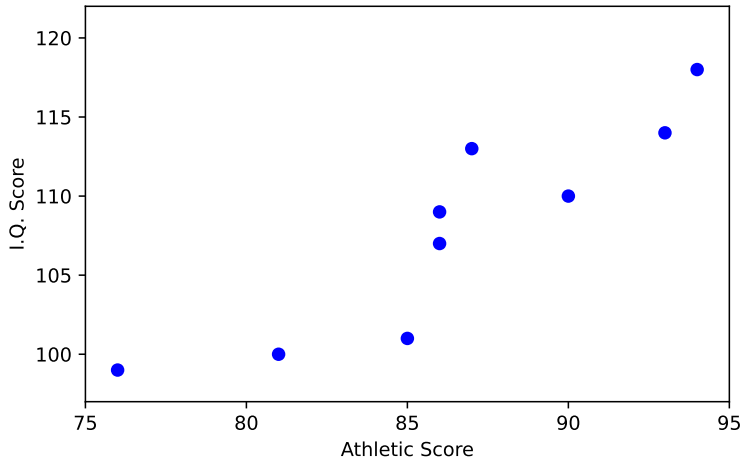


Figure 29.2: Hypothetical Scores for I.Q. and Athletic Ability — 1

In Figure 29.4, which plots the scores as given in Table 29.1, the prediction of athletic score given I.Q. score, or vice versa, is less clear-cut than in Figure 29.2. On the basis of Figure 29.4

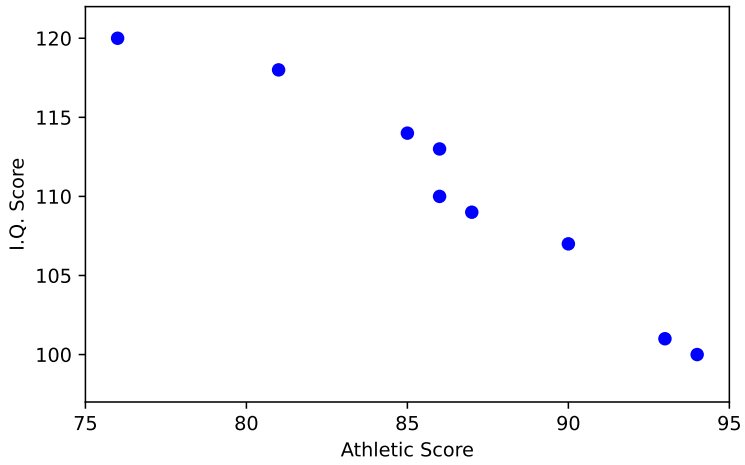


Figure 29.3: Hypothetical Scores for I.Q. and Athletic Ability — 2

alone, one can say only that there *might* be some association between the two variables.

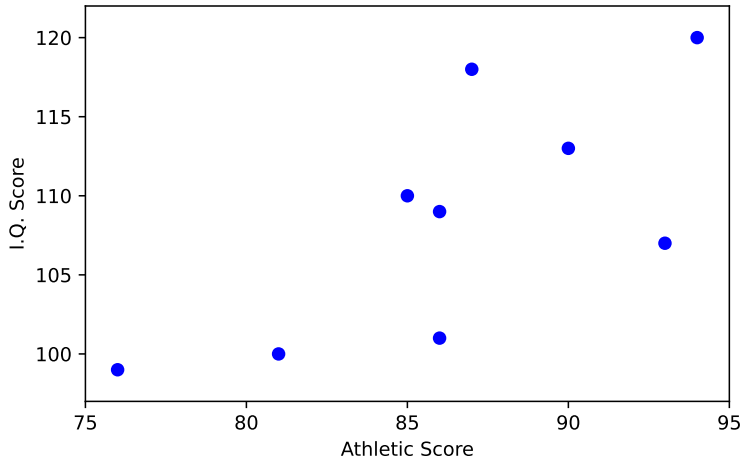


Figure 29.4: Given Scores for I.Q. and Athletic Ability

29.4 Correlation with sum of products

Now let us take advantage of a handy property of numbers. The more closely two sets of numbers match each other in order, the higher the sums of their products. Consider the following arrays of the numbers 1, 2, and 3:

Columns in matching order

$$1 * 1 = 1$$
$$2 * 2 = 4$$
$$3 * 3 = 9$$

$$\text{SUM} = 14$$

Columns not in matching order

$$1 * 2 = 2$$
$$2 * 3 = 6$$
$$3 * 1 = 3$$

$$\text{SUM} = 11$$

I will not attempt a mathematical proof, but the reader is encouraged to try additional combinations to be sure that the highest sum is obtained when the order of the two columns is the same. Likewise, the lowest sum is obtained when the two columns are in perfectly opposite order:

Columns in opposite order

$$1 * 3 = 3$$
$$2 * 3 = 4$$
$$3 * 1 = 3$$

$$\text{SUM} = 10$$

Consider the cases in Table 29.4 which are chosen to illustrate a perfect (linear) association between x (Column 1) and *y1* (Column 2), and also between x (Column 1) and *y2* (Column 4); the numbers shown in Columns 3 and 5 are those that would be consistent with perfect associations. Notice the sum of the multiples of the x and y values in the two cases. It is either higher (x * *y1*) or lower (x * *y2*) than for any other possible way of arranging the y's. Any other arrangement of the y's (*y3*, in Column 6, for example, chosen at random), when multiplied by the x's in Column 1, (x * *y3*), produces a sum that falls somewhere between

the sums of x * *y1* and x * *y2*, as is the case with any other set of *y3*'s which is not perfectly correlated with the x's.

Table 29.4: Comparison of Sums of Multiplications

Strong positive relationship			Strong negative relationship		Random Pairings	
X	Y1	X * Y1	Y2	X * Y2	Y3	X * Y3
2	2	4	10	20	4	8
4	4	8	8	32	8	32
6	6	36	6	36	6	36
8	8	64	4	32	2	16
10	10	100	2	20	10	100
Sums		220		140		192

29.4.1 Example: sum of products correlation of athletic and IQ scores

Table 29.5 shows that the sum of the products of the *observed* I.Q. scores multiplied by athletic scores (column 7) is between the sums that would occur if the I.Q. scores were ranked from best to worst (column 3) and worst to best (column 5). The extent of correlation (association) can thus be measured by whether the sum of the multiples of the observed x and y values is relatively much higher or much lower than are sums of randomly-chosen pairs of x and y.

Table 29.5: Sums of Products: IQ and Athletic Scores

1 Athletic score	2 Hypothetical IQ pos	3 Col 1 x Col 2	4 Hypothetical IQ neg	5 Col 1 x Col 4	6 Actual IQ	7 Col 1 x Col 6
97	120	11640	99	9603	114	11058
94	118	11092	100	9400	120	11280
93	114	10602	101	9393	107	9951
90	110	9900	107	9630	113	10170
87	113	9831	109	9483	118	10266
86	109	9374	113	9718	101	8686
86	107	9202	110	9460	109	9374
85	101	8585	114	9690	110	9350
81	100	8100	118	9558	100	8100
76	99	7524	120	9120	99	7524
SUMS		95850		95055		95759

Table 29.5 show three cases of products:

1. Perfect positive correlation (hypothetical); column 3
2. Perfect negative correlation (hypothetical); column 5
3. Observed; column 7

Now we attack the I.Q. and athletic-score problem using the property of numbers just discussed. First multiply the x and y values of the actual observations, and sum them; we find a total of 95,759 (Table 29.5). Then write the ten observed I.Q. scores on cards, and assign the cards in random order to the ten athletes, as shown in the column for trial 1 in Table 29.6.

Multiply the random IQ scores by their paired x's (Athletic scores), and sum to give the values at the bottom of Table 29.6. If the I.Q. scores and athletic scores are *positively associated*, that is, if high I.Q.s and high athletic scores go together, then the sum of the multiplications for the observed sample will be higher than for most of the random trials. (If high I.Q.s go with low athletic scores, the sum of the multiplications for the observed sample will be *lower* than most of the random trials.)

Table 29.6: Random Draws of IQ scores paired against athletic scores

Athletic score	1	2	3	4	5	6	7	8	9	10
97	100	110	101	113	107	109	109	114	100	118
94	120	109	99	101	100	113	107	110	118	107
93	114	113	113	120	99	120	100	113	99	100
90	101	120	114	99	110	107	101	107	107	120
87	118	101	110	107	118	100	120	99	114	114
86	110	114	118	114	101	101	110	109	101	110
86	107	100	120	109	114	114	113	120	120	101
85	99	99	107	110	120	110	118	100	109	113
81	113	118	100	118	109	99	99	118	113	109
76	109	107	109	100	113	118	114	101	110	99
Product sums	95452	95521	95389	95520	95259	95512	95352	95578	95331	95626

More specifically, by the steps:

- **Step 1.** Write the ten I.Q. scores on one set of cards, and the ten athletic scores on another set of cards.
- **Step 2.** Pair the I.Q. and athletic-score cards at random. Multiply the scores in each pair, and add the results of the ten multiplications.
- **Step 3.** Subtract the experimental sum in step 2 from the observed sum, 95,759.
- **Step 4.** Repeat steps 2 and 3 ten times.

- **Step 5.** Compute the proportion of trials where the difference is negative, which estimates the probability that an association as strong as the observed would occur by chance.

The sums of the multiplications for 10 trials are shown in Table 29.6. No random-trial sum was as high as the observed sum, which suggests that the probability of an association this strong happening by chance is so low as to approach zero. (An empirically-observed probability is never actually zero.)

This algorithm can be solved particularly easily with Python. The arrays `ath` and `iq` in the notebook below list the athletic scores and the I.Q. scores respectively of the 10 "actual" students. We multiply the corresponding elements of these arrays and proceed to compare the sum of these multiplications to the sums of experimental multiplications in which the elements of `iq` have been randomly permuted, to form random pairings.

Finally, we count (`sum`) the trials in which the sum of the products of the randomly-paired athletic and I.Q. scores equals or exceeds the sum of the products in the observed data.

> **Notebook with data file**
>
> As have seen already (see Note 36),the download link points to a `.zip` file containing the notebook and the data file the notebook will read.

> **Note 71: Notebook: Association of athletic and IQ scores**
>
> - Download zip with notebook + data file
> - Interact

```
# Load the Numpy library for arrays.
import numpy as np
# Load the Pandas library for loading and selecting data.
import pandas as pd
# Plotting library.
import matplotlib.pyplot as plt

# Set up the random number generator
rnd = np.random.default_rng()
```

We load the file containing the data:

```
# Read the data file containing athletic and IQ scores.
ath_iq_df = pd.read_csv('data/athletic_iq.csv')
```

```
# Show the data frame.
ath_iq_df
```

	athletic_score	iq_score	athletic_rank	iq_rank
0	97	114	1	3
1	94	120	2	1
2	93	107	3	7
3	90	113	4	4
4	87	118	5	2
5	86	101	6	8
6	86	109	7	6
7	85	110	8	5
8	81	100	9	9
9	76	99	10	10

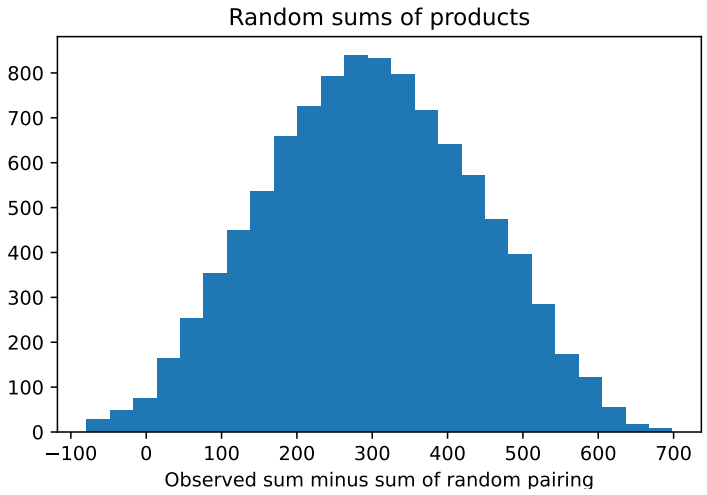
```
# Turn athletic and IQ scores into arrays.
ath = np.array(ath_iq_df['athletic_score'])
iq = np.array(ath_iq_df['iq_score'])
```

```
# Multiply the two sets of scores together.
actual_prod = ath * iq
# Sum the results - the "observed value."
actual_sum = np.sum(actual_prod)
actual_sum
```

```
np.int64(95759)
```

```
# Set the number of trials
n_trials = 10_000

# An empty array to store the trial results.
results = np.zeros(n_trials)

# Do 10,000 experiments.
for i in range(n_trials):
    # Shuffle the IQ scores so we can pair them against athletic scores.
    shuffled = rnd.permuted(iq)
    # Multiply the shuffled IQ scores by the athletic scores. (Note that we
    # could shuffle the athletic scores too but it would not achieve any
    # greater randomization, because shuffling one of the set of scores
```

```
    # already gives a random pairing between the sets of scores.
    fake_prod = ath * shuffled
    # Sum the randomized multiplications.
    fake_sum = np.sum(fake_prod)
    # Subtract the sum from the sum of the "observed" multiplication.
    diff = actual_sum - fake_sum
    # Keep track of the result in results array.
    results[i] = diff
    # End one trial, go back and repeat until 10000 trials are complete.
# Obtain a histogram of the trial results.
plt.hist(results, bins=25)
plt.title('Random sums of products')
plt.xlabel('Observed sum minus sum of random pairing')
```



We see that obtaining a chance trial result as great as that observed was rare. Python will calculate this proportion for us:

```
# Determine in how many trials the random sum of products was less than
# the observed sum of products.
k = np.sum(results <= 0)
# Convert to a proportion.
kk = k / n_trials
# Print the result.
print('Proportion of random pairings giving sum <= observed:', kk)
```

```
Proportion of random pairings giving sum <= observed: 0.0111
```

End of notebook: Association of athletic and IQ scores

`athlete_iq` starts at Note 71.

29.4.2 Example: Correlation Between Adherence to Medication Regime and Change in Cholesterol

Efron and Tibshirani (1993, 72) show data on the extents to which 164 men a) took the drug prescribed to them (cholestyramine), and b) showed a decrease in total plasma cholesterol. Table 29.7 shows the first 15 of the values (note that a positive value in the "Decrease in cholesterol" column denotes a decrease in cholesterol, while a negative value denotes an increase.)

Table 29.7: First 15 rows of compliance / blood cholesterol

% compliance	Decrease in cholesterol
0	-5.25
27	-1.50
71	59.50
95	32.50
0	-7.25
28	23.50
71	14.75
95	70.75
0	-6.25
29	33.00
72	63.00
95	18.25
0	11.50
31	4.25
72	0.00

The aim is to assess the effect of the compliance on the improvement. There are two related issues:

1. What form of regression should be fitted to these data, which we address later, and
2. Is there reason to believe that the relationship is meaningful? That is, we wish to ascertain if there is any meaningful correlation between the variables — because if there is no relationship between the variables, there is no basis for regressing one on the other. Sometimes people jump ahead in the latter question to first run the regression and

then ask whether the regression slope coefficient(s) is (are) different than zero, but this usually is not sound practice. The sensible way to proceed is first to graph the data to see whether there is visible indication of a relationship.

Efron and Tibshirani do this, and they find sufficient intuitive basis in the graph to continue the analysis (see Figure 29.5).

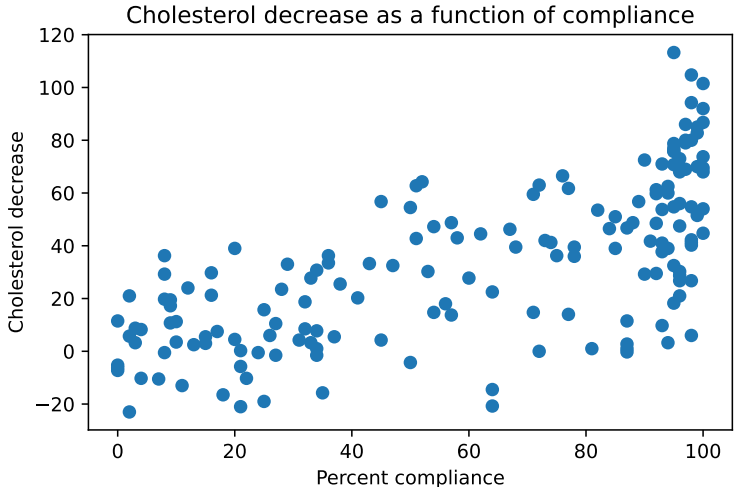


Figure 29.5: Cholesterol decrease as function of compliance

The next step is to investigate whether a measure of relationship is statistically convincing; this we do as follows:

1. Multiply the observed values for each of the 164 participants on the independent x variable (cholestyramine — percent of prescribed dosage actually taken) and the dependent y variable (cholesterol), and sum the results — it's 439,141.
2. Randomly shuffle the dependent variable y values among the participants. The sampling is being done without replacement, though an equally good argument could be made for sampling with replacement; the results do not differ meaningfully, however, because the sample size is so large.
3. Then multiply these x and y hypothetical values for each of the 164 participants, sum the results and record.
4. Repeat steps 2 and 3 perhaps 10,000 times.
5. Determine how often the shuffled sum-of-products exceeds the observed value (439,141).

The following notebook in Python provides the solution:

Note 72: Notebook: Testing association of compliance and cholesterol values

- Download zip with notebook + data file
- Interact

```
# Load the Numpy library for arrays.
import numpy as np
# Load the Pandas library for loading and selecting data.
import pandas as pd
# Plotting library.
import matplotlib.pyplot as plt

# Set up the random number generator
rnd = np.random.default_rng()

# Data as arrays.
df = pd.read_csv('data/cholost.csv')
compliance = np.array(df['percent_compliance'])
cholesterol = np.array(df['cholesterol_decrease'])

# Rename the two sequences to match the description in the text.
x = compliance
y = cholesterol

# Step 1 above
actual_prod = x * y
# Note: actual_sum = 439,141
actual_sum = np.sum(actual_prod)
# Show the result.
actual_sum
```

```
np.float64(439140.75)
```

```
# Set the number of trials
n_trials = 10_000

# An empty array to store the trial results.
results = np.zeros(n_trials)

# Do 10,000 experiments (step 4)
for i in range(n_trials):
```

```
    # Step 2 above.
    y_random = rnd.permuted(y)
    # Step 3 above.
    fake_prod = x * y_random
    fake_sum = np.sum(fake_prod)
    # Step 3 above
    results[i] = fake_sum
    # Step 4 above

# Step 5 above
k = np.sum(results >= actual_sum)
kk = k / n_trials

print('Proportion product sums >= observed:', kk)
```

```
Proportion product sums >= observed: 0.0
```

> **i** End of notebook: Testing association of compliance and cholesterol values
>
> `compliance_cholesterol` starts at Note 72.

Interpretation: 10,000 simulated random shufflings never produced a sum-of-products as high as the observed value. Hence we conclude that random chance is a very unlikely explanation for the observed correlation.

29.5 The correlation coefficient

We have been using the *sum of products* as a measure of straight-line (linear) association between two variables. We found that the sum of products for the athletic and IQ variables (`ath` and `iq`) was 95759, and we showed that the sum of products is higher for pairings of `ath` and `iq` with a higher degree of linear association. Now let us consider compliance and cholesterol above; the sum of products for `compliance` and `cholesterol` was 439141. Notice the `compliance` and `cholesterol` sum of products is much higher than for `ath` and `iq` — but, for various reasons that we explain below, that does not tell us whether the *association of the scores* is higher for the compliance / cholesterol values than the athletic / IQ scores. We would like a version of the sum of products procedure that could give us a measure of linear association that we could compare between different pairs of sequences. This is the purpose of the *correlation coefficient*.

The first problem that makes it hard to compare the sum of product scores is that the sum of products depends on the number of elements in the sequence. `ath` (and `iq`) have 10 elements, whereas `compliance` (and `cholesterol`) have 164 elements. The sum of products is a sum; all other things being equal, the more values you sum, the higher the sum; all other things being equal, the sum of 164 elements will be greater than the sum of 10 elements. We can solve this problem by dividing the sum of products by the number of elements in that sum, to get the *mean* of the products. The mean of product values for the athletic / IQ scores is 95759 / 10 = 9575.9. The mean of products for compliance / cholesterol is 439141 / 164 = 2677.69.

We have solved one problem in comparing the measure by taking the mean of products — the value no longer depends on the number of elements. We have other problems. For example, consider the IQ scores. We have used the raw IQ scores, on their usual scale, where 100 is the average IQ for the general population. Our raw IQ scores have a mean of 109.1. But we could also have scored IQ in another way. For example, we could have recorded the *difference* from 100 instead of the raw value, to get the values in column 2 of Table 29.8. Call these the "IQ-100" scores. IQ-100 scores have a mean of 9.1.

Table 29.8: Means of Products: IQ, IQ-100 and Athletic Scores

1 Athletic score	2 Original IQ	3 Col 1 x Col 2	4 IQ-100	5 Col 2 x Col 4
97	114	11058	14	1358
94	120	11280	20	1880
93	107	9951	7	651
90	113	10170	13	1170
87	118	10266	18	1566
86	101	8686	1	86
86	109	9374	9	774
85	110	9350	10	850
81	100	8100	0	0
76	99	7524	-1	-76
SUMS		95759		8259
MEANS		9575.9		825.9

Notice that the mean product for the IQ-100 scores is much smaller than the mean for the product with the original IQ values, because the IQ element in the product is smaller, so the products will be smaller, and the mean product will be smaller. However, it might be obvious that the *association* of the IQ and athletic scores remains the same after we have subtracted 100 from the IQ; higher IQ values on either scale are still associated with higher athletic scores. Think of the association as a straight-line relationship in a scatter plot, with the athletic scores on the x-axis, and the IQ scores on the y-axis. Figure 29.1 has the original scatter plot with the athletic scores on the x-axis and the original IQ scores on the y-axis. The points appear

to scatter around some straight line we could draw. Figure 29.6 shows the scatterplot of the original IQ score against the athletic scores on the left plot (as for Figure 29.1) and the IQ-100 scores against the athletic scores in the right plot. As you can see, subtracting 100 from the IQ values just moved the points on the graph down by 100 on the y-axis — but the slope of the line relating the two must be the same. The slope is a measure of how much the IQ score increases as the athletic score increases — it is therefore a good measure of the *linear association* between athletic score and IQ score. Because the slope of our potential line should not change between the original IQ score and the IQ-100 score, a good measure of association should not change, when we do this subtraction. That is not true of the mean product that is much smaller after the subtraction.

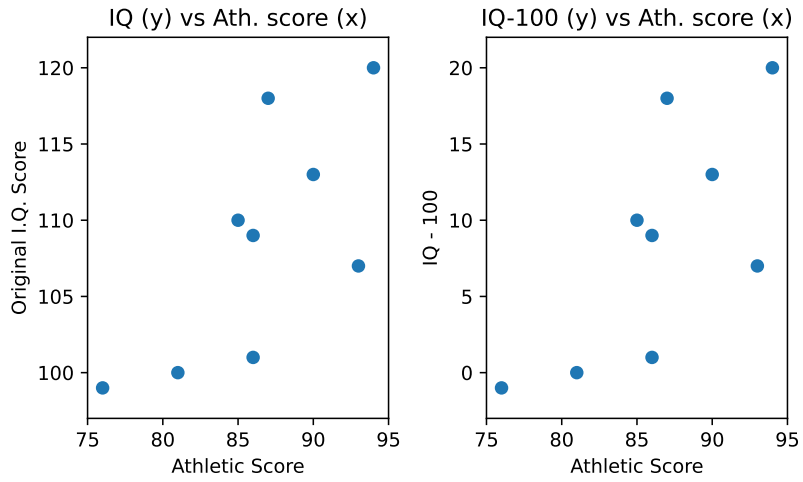


Figure 29.6: Scatter plot of IQ Score, IQ-100 as function of Athletic Score

So, the mean product will change as we add or subtract values to one or both of the sequences. You might also be able to see that the mean product will change if we multiply or divide the values in the sequence by some number. In summary, the mean product depends on *position* or *location* of the values in the sequence (on the x and y axes), and the *scale* or *units* of the values.

We would like some measure of association that does not depend on the location and scale of the sequences. A simple approach is to put each sequence on a *standard* scale, by converting the original sequence values to *standard scores* (see Section 16.9).

We do this by:

1. Subtracting the mean value from each element in the sequence to give the *deviations*.
2. Dividing the deviations by a measure of spread — the *standard deviation* (see Section 16.7).

See Table 29.9 for an example of applying this algorithm to the IQ scores. We get the deviations by subtracting the mean IQ (109.1), and the standard scores by dividing the deviations by the IQ standard deviation (7.02).

Table 29.9: IQ, deviations and standard scores

Original IQ	IQ deviations	Standard scores
114	4.9	0.7
120	10.9	1.55
107	-2.1	-0.3
113	3.9	0.56
118	8.9	1.27
101	-8.1	-1.15
109	-0.1	-0.01
110	0.9	0.13
100	-9.1	-1.3
99	-10.1	-1.44

After we apply this procedure to both sequences, they are both in the same *standard* units. The mean product of these standard units is therefore a *standard* measure of association, called the *Pearson product-moment correlation coefficient*, or just the *correlation coefficient*. We use the letter r for the value of the correlation coefficient. Table 29.10 shows the mean of products calculation on standard scores athletic ability and IQ, to get the r value.

Table 29.10: r for Athletic and IQ scores

1 Athletic score	2 Original IQ	3 Standardized athletic score	4 Standardized IQ	5 Col 3 x Col 4
97	114	1.6	0.7	1.11
94	120	1.09	1.55	1.69
93	107	0.92	-0.3	-0.28
90	113	0.42	0.56	0.23
87	118	-0.08	1.27	-0.11
86	101	-0.25	-1.15	0.29
86	109	-0.25	-0.01	0
85	110	-0.42	0.13	-0.05
81	100	-1.09	-1.3	1.42
76	99	-1.93	-1.44	2.78
Mean product (r value)				0.71

Because the sequences for the correlation coefficient r are in the same units, we can compare correlation coefficients between different sequences, to give a standard measure of association. For example, we can run the same calculation on the compliance and cholesterol values, to get the r value (correlation coefficient) for these sequences. Let's do that in Python:

> **Note 73:** Notebook: Correlation coefficient for compliance and cholesterol
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np
import pandas as pd

df = pd.read_csv('data/cholost.csv')
compliance = np.array(df['percent_compliance'])
cholesterol = np.array(df['cholesterol_decrease'])

# Compliance as standard scores.
dev_compliance = compliance - np.mean(compliance)  # Deviations.
stdev_compliance = np.sqrt(np.mean(dev_compliance ** 2))  # Standard deviation.
# We could also have used np.std to do the calculation above.
std_compliance = dev_compliance / stdev_compliance  # Standard scores.

# Cholesterol as standard scores.
dev_cholesterol = cholesterol - np.mean(cholesterol)  # Deviations.
stdev_cholesterol = np.sqrt(np.mean(dev_cholesterol ** 2))  # Standard deviation.
# We could also have used np.std to do the calculation above.
std_cholesterol = dev_cholesterol / stdev_cholesterol  # Standard scores.

# r value is mean of products for standardized scores.
r = np.mean(std_compliance * std_cholesterol)

print('r for compliance / cholesterol is:', np.round(r, 2))
```

```
r for compliance / cholesterol is: 0.68
```

> **End of notebook:** Correlation coefficient for compliance and cholesterol
>
> `r_for_cholesterol` starts at Note 73.

Notice that the r value is higher for the athletic and IQ scores (0.71) than it is for compliance and cholesterol (0.68). Because we have measured the association in standard units, we can compare these values, and say there is a slightly stronger linear association for our athletic and IQ scores, than there is for our compliance and cholesterol scores.

29.5.1 Correlations are symmetrical

You might be able to see from the definition of the calculation above, that the correlation coefficient r between `ath` and `iq` must be the same as the correlation coefficient (r) between `iq` and `ath` — and in general the correlation coefficient between some sequence x and another y is equal to the correlation coefficient between y and x.

Why? Remember that the correlation coefficient is the mean of the product of the standard scores.

Let us first calculate the standard scores for `ath` and `iq`:

```
std_ath = (ath - np.mean(ath)) / np.std(ath)
std_iq = (iq - np.mean(iq)) / np.std(iq)
```

Now calculate the correlation coefficient.

```
print('r value for "ath" and "iq":', np.mean(std_ath * std_iq))
```

```
r value for "ath" and "iq": 0.7093117740309458
```

But multiplication is commutative — changing the order does not change the result — so $a * b = b * a$. This means that the `std_ath * std_id` gives the same array as `std_id * std_ath`, and:

```
print('r value for "iq" and "ath":', np.mean(std_iq * std_ath))
```

```
r value for "iq" and "ath": 0.7093117740309458
```

29.5.2 The correlation coefficient in Python

NumPy has a function `np.corrcoef` for calculating the correlation coefficient between two sequences directly.

Here is `np.corrcoef` in action:

```
corr_arr = np.corrcoef(ath, iq)
corr_arr
```

```
array([[1.        , 0.70931177],
       [0.70931177, 1.        ]])
```

Notice that `np.corrcoef` returns a *two-dimensional* array of the correlations between all possible pairs of the two sequences. This results in a 2 by 2 array, where the top-left value is the correlation coefficient r between `ath` and `ath` — the r value for a sequence and itself is always 1. The bottom right value is the r value for `iq` and `iq` — again, this must be 1. The top right value is the r between `ath` and `iq`, and the bottom left value is the r between `iq` and `ath`, and as you saw above, these must be equal. We are usually interested in the r between the two sequences we send to `np.corrcoef`, so we usually just want the top-right (or bottom-left) value.

We can select the first row of the 2D array like this:

```
# First row of the 2D array.
corr_arr[0]
```

```
array([1.        , 0.70931177])
```

We can select the first row and the first column by giving two indices between the square brackets, where the first index is for the row (select the first (0) row), and the second is for the column (select the second (1) column):

```
# Select the first row, second column of the 2D array.
# This is the top-right value.
corr_arr[0, 1]
```

```
np.float64(0.7093117740309459)
```

Therefore, to get the correlation coefficient between two sequences, we can do:

```
# Select top-right value from correlation coefficient array.
np.corrcoef(ath, iq)[0, 1]
```

```
np.float64(0.7093117740309459)
```

29.5.3 Test linear association with the correlation coefficient

Above we used the sum-of-products calculation to see if the actual observed sum of products was unusual in the null-world. We simulated sum of product values in the null-world by permuting one of the sequences, and recalculating the sum of products.

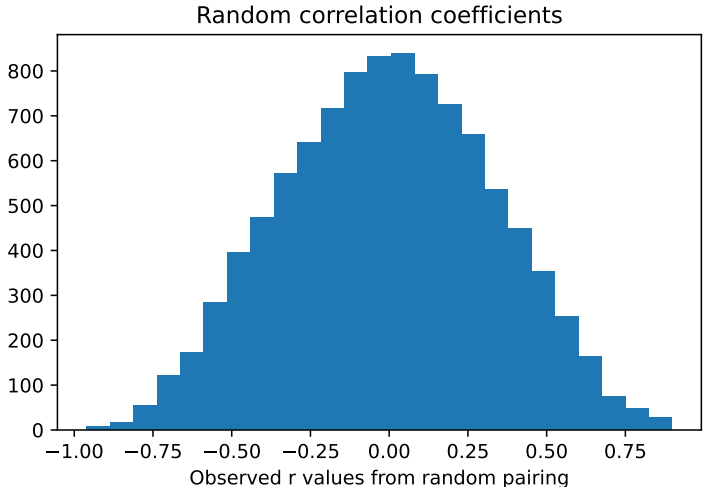
We could use the correlation coefficient instead of the sum of products as our measure of association, and simulate correlation coefficients in the null-world. We will come to the same conclusion using sum of products as we would for correlation coefficients because they are equivalent when comparing between two different orderings of the same sequences of values. The only advantage of the correlation coefficient for this case is that NumPy has the ready-made function `np.corrcoef` to do the calculation. Here is the test for linear association using the correlation coefficient instead of the sum of products. Compare the procedure below to Section 29.4.1.

> Note 74: Notebook: Association of athletic and IQ scores with r
>
> - Download zip with notebook + data file
> - Interact

```
# Load the Numpy library for arrays.
import numpy as np
# Load the Pandas library for loading and selecting data.
import pandas as pd
# Plotting library.
import matplotlib.pyplot as plt

# Set up the random number generator
rnd = np.random.default_rng()

# Read the data file containing athletic and IQ scores.
ath_iq_df = pd.read_csv('data/athletic_iq.csv')

# Turn athletic and IQ scores into arrays.
ath = np.array(ath_iq_df['athletic_score'])
iq = np.array(ath_iq_df['iq_score'])

# Calculate, select correlation coefficient.
actual_r = np.corrcoef(ath, iq)[0, 1]

# Set the number of trials for the null-world simulation.
n_trials = 10_000
```

```
# An empty array to store the trial results.
results = np.zeros(n_trials)

# Do 10,000 experiments.
for i in range(n_trials):
    # Shuffle the IQ scores so we can pair them against athletic scores.
    shuffled = rnd.permuted(iq)
    # Calculate the correlation coefficient.
    fake_r = np.corrcoef(ath, shuffled)[0, 1]
    # Keep track of the result in results array.
    results[i] = fake_r
    # End one trial, go back and repeat until 10000 trials are complete.

# Obtain a histogram of the trial results.
plt.hist(results, bins=25)
plt.title('Random correlation coefficients')
plt.xlabel('Observed r values from random pairing')

# Determine in how many trials the random r value was greater than
# the observed r value.
k = np.sum(results >= actual_r)
# Convert to a proportion.
kk = k / n_trials
# Print the result.
print('Proportion of random pairings giving r >= observed:', kk)
```

```
Proportion of random pairings giving r >= observed: 0.0111
```



End of notebook: Association of athletic and IQ scores with r

`athlete_iq_cc` starts at Note 74.

Notice we get a very similar proportion (p-value) with the correlation coefficient to the one from sum of products in Section 29.4.1; in fact, the p-values only differ because the p-value is itself somewhat random, as it derives from (many) random samples.

29.6 Testing for a relationship between counted-data variables

29.6.1 Example: Drinking Beer And Being In Favor of Selling Beer

Earlier in this chapter, we analyzed measured values for athletic ability and I.Q. We can use them in their original "cardinal" form, or split them up into "high" and "low" groups. Often, however, the individual observations are recorded only as categories such "yes" or "no," which makes it more difficult to ascertain the existence of a relationship. Consider the poll responses in Table 29.11 to two public-opinion survey questions: "Do you drink beer?" and "Are you in favor of a local option on the sale of beer?".[3]

Table 29.11: Results of Observed Sample For beer poll

Do you favor local option on the sale of beer?	Do you drink beer?		
	Yes	No	**Total**

[3]These data are based on an example in (Dixon and Massey Jr 1983, 278, example 13-5), in which the problem is tackled conventionally with a chi-square test.

Favor	45	20	**65**
Don't favor	7	6	**13**
Total	**52**	**26**	**78**

Here is the statistical question: Is a person's opinion on "local option" related to whether or not they drink beer? Our resampling solution begins by noting that there are seventy-eight respondents, sixty-five of whom approve local option and thirteen of whom do not. Therefore write "approve" on sixty-five index cards and "not approve" on thirteen index cards. Now take *another* set of seventy-eight index cards, preferably of a different color, and write "yes" on fifty-two of them and "no" on twenty-six of them, corresponding to the numbers of people who do and do not drink beer in the sample. Now lay them down in random *pairs*, one from each pile.

If there is a high association between the variables, then real life observations will bunch up in the two diagonal cells in the upper left and lower right in Table 29.11. (Ignore the "total" data for now.) Put another way, people filling in the upper left and lower right cells are people with views compatible with their drinking habits (drink-yes / favor, or drink-no / don't favor). Conversely, people filling in the lower left and upper right cells have views incompatible with their drinking habits (drink-yes, don't favor, or drink-no, favor). Adding up the upper left / lower right diagonal gives us the total number of *compatible* responses, and adding the lower left / upper right diagonal gives us the *incompatible* total. To get an index of how strongly the table data represents compatible responses, we can subtract the incompatible total (lower left plus upper right) from the compatible total (upper left plus lower right) for the observed data: (45 + 6) - (20 + 7) = 24. Then compare this difference to the comparable differences found in random trials. The proportion of times that the simulated-trial difference exceeds the observed difference is the probability that the observed difference of +24 might occur by chance, even if there is no relationship between the two variables. (Notice that, in this case, we are working on the assumption that beer drinking is *positively* associated with approval of local option and not the inverse. We are interested only in differences that are equal to or exceed +24 when the northeast-southwest diagonal is subtracted from the northwest-southeast diagonal.)

We can carry out a resampling test with this procedure:

- **Step 1.** Write "approve" on 65 and "disapprove" on 13 red index cards, respectively; write "Drink" and "Don't drink" on 52 and 26 white cards, respectively.
- **Step 2.** Pair the two sets of cards randomly. Count the numbers of the four possible pairs: (1) "approve-drink," (2) "disapprove-don't drink," (3) "disapprove-drink," and (4) "approve-don't drink." Record the number of these combinations, as in Table 29.12, where columns 1-4 correspond to the four data cells in Table 29.11.
- **Step 3.** Add (column 1 plus column 4), then add (column 2 plus column 3), and subtract the result in the second parenthesis from the result in the first parenthesis. If the difference is equal to or greater than 24, record "yes," otherwise "no."

- **Step 4.** Repeat steps 2 and 3 perhaps a hundred times.
- **Step 5.** Calculate the proportion "yes," which estimates the probability that an association this great or greater would be observed by chance.

Table 29.12: Results of one random trial of the beer poll problem

Trial	1 Approve Yes	2 Approve No	3 Disapprove Yes	4 Disapprove No	5 (Col 1 + Col 4) - (Col 2 + Col 3)
1	43	22	9	4	47 - 31 = 16

Table 29.12 shows the result of one such trial. When we run a series of ten trials such trials, it indicates that the observed difference is very often exceeded, which suggests that we have no good evidence here for a relationship between beer drinking and opinion.

The Python notebook below does this repetitively. From the "actual" sample results we know that 52 respondents drink beer and 26 do not. We create the array "drink" with 52 1's for those who drink beer, and 26 0's for those who do not. We also create the array "sale" with 65 1's (approve) and 13 0's (disapprove). In the actual sample, 51 of the 78 respondents had "consistent" responses to the two questions — that is, people who both favor the sale of beer *and* drink beer, or who are against the sale of beer *and* do not drink beer. We want to randomly pair the responses to the two questions to compare against that observed result to test the relationship.

> **Note 75: Notebook: Does opinion match behavior in a poll?**
>
> - Download notebook
> - Interact

To accomplish this aim, we repeat the following procedure 10,000 times using a `for` loop. We shuffle `drink` to `random_drink` so that the responses are randomly ordered. Now when we subtract the corresponding elements of the two arrays, a 0 will appear in each element of the new array `diffs` for which there was consistency in the response of the two questions. We therefore count (`sum`) the times that `diffs` equals 0 and place this result in the array `consistent` and the number of times `diffs` does not equal 0, and place this result in `inconsistent`. Find the difference (`consistent` minus `inconsistent`), and record the result in the `results` array for each trial.

`results` stores, for each trial, the number of consistent responses minus inconsistent responses. To determine whether the results of the actual sample indicate a relationship between the responses to the two questions, we check how often the random trials had a difference (between consistent and inconsistent responses) as great as 24, the value in the observed sample.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

# Constitute the set of 52 beer drinkers, and the set of 26 non-drinkers,
# where 1 represents beer drinker, and 0 represents non-drinker.
drink = np.repeat([1, 0], [52, 26])
# Show the result
drink
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

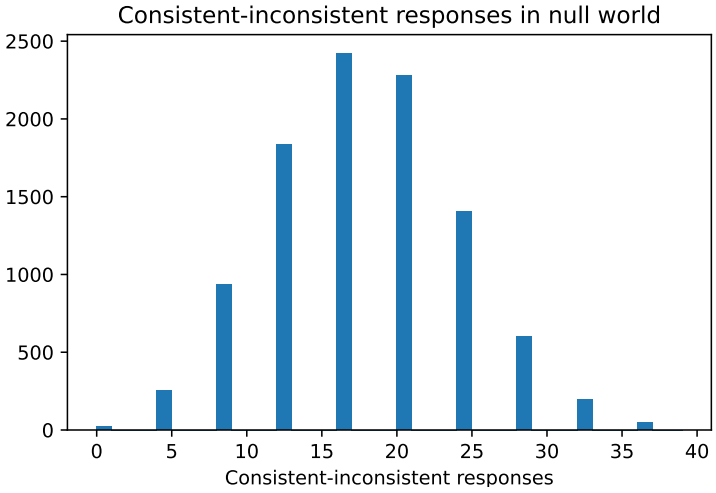
```
# The same set of individuals classified by whether they favor or
# don't favor the sale of beer, where 1 represents those who favor
# local beer sale, and 0 represents those who do not.
sale = np.repeat([1, 0], [65, 13])
# Show the result
sale
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# Number of repeats.
n_trials = 10_000

# Array to store the results of each trial.
results = np.zeros(n_trials)

# Repeat the experiment 10,000 times.
for i in range(n_trials):
    # Shuffle the drink labels..
    random_drink = rnd.permuted(drink)
    # Note - random_drink is now an array like this:
    # [1, 0, 1, 1, 1, 0, 0, 1 ...]
```

```
    # Subtract the favor/don't favor set from the drink/don't drink set.
    # Consistent responses are someone who drinks favoring the sale of beer (a
    # 1 and a 1) or someone who doesn't drink opposing the sale of beer.
    # When subtracted, consistent responses *(and only consistent responses)*
    # produce a 0.
    diffs = random_drink - sale
    # Count the number of consistent responses (those equal to 0).
    consistent = np.sum(diffs == 0)
    # Count the "inconsistent" responses (those not equal to 0).
    inconsistent = np.sum(diffs != 0)
    # Find the difference.
    diff_diff = consistent - inconsistent
    # Keep track of the results of each trial.
    results[i] = diff_diff
    # End one trial, go back and repeat until all 10,000 trials are complete.
# Produce a histogram of the trial result.
plt.hist(results, bins=np.arange(0, 40))
plt.title('Consistent-inconsistent responses in null world')
plt.xlabel('Consistent-inconsistent responses')

# Count differences >= observed.
k = np.sum(results >= 24)
# Proportion.
kk = k / n_trials
# Print result.
print('Proportion null-world differences >= observed:', kk)
```

```
Proportion null-world differences >= observed: 0.2251
```



ℹ End of notebook: Does opinion match behavior in a poll?

`consistent_drink` starts at Note 75.

The actual results showed a difference of 24. In the histogram we see that a difference that large or larger happened just by chance pairing — without any relationship between the two variables — about 23 percent of the time. Hence, we conclude that there is little evidence of a relationship between the two variables.

Though the test just described may *generally* be appropriate for data of this sort, it may well not be appropriate in some particular case. Let's consider a set of data where even if the test showed that an association existed, we would not believe the test result to be meaningful.

Suppose the survey results had been as presented in Table 29.13. We see that non-beer drinkers have a higher rate of approval of allowing beer drinking, which does not accord with experience or reason. Hence, without additional explanation we would not believe that a meaningful relationship exists among these variables even if the test showed one to exist. (Still another reason to doubt that a relationship exists is that the absolute differences are too small to mean anything to anyone — there is only a 6% difference in disapproval between drink and don't drink groups. On both grounds, then, it makes sense simply to act as if there were no difference between the two groups and to run *no test*.)

Table 29.13: Beer poll in which results are not in accord with expectation

	% Approve	% Disapprove	Total
Beer Drinkers	71%	29%	100%
Non-Beer Drinkers	77%	23%	100%

The lesson to be learned from this is that one should inspect the data carefully before applying a statistical test, and only test for "significance" if the apparent relationships accord with theory, general understanding, and common sense.

29.6.2 Example: do athletes really have "slumps"?

Are successive events in a series independent, or is there a relationship between them?

The important concept of independent events was introduced earlier. Various scientific and statistical decisions depend upon whether or not a series of events is independent. But how does one know whether or not the events are independent? Let us consider a baseball example.

Baseball players and their coaches believe that on some days and during some weeks a player will bat better than on other days and during other weeks. And team managers and coaches act on the belief that there are periods in which players do poorly — slumps — by temporarily replacing the player with another after a period of poor performance. The underlying belief is that a series of failures indicates a temporary (or permanent) change in the player's capacity to play well, and it therefore makes sense to replace him until the evil spirit passes on, either of its own accord or by some change in the player's style.

But even if his hits come randomly, a player will have runs of good luck and runs of bad luck just by chance — just as does a card player. The problem, then, is to determine whether (a) the runs of good and bad batting are merely runs of chance, and the probability of success for each event remains the same throughout the series of events — which would imply that the batter's ability is the same at all times, and coaches should not take recent performance heavily into account when deciding which players should play; or (b) whether a batter really does have a tendency to do better at some times than at others, which would imply that there is some relationship between the occurrence of success in one trial event and the probability of success in the next trial event, and therefore that it is reasonable to replace players from time to time.

Let's analyze the batting of a player we shall call "Slug." Here are the results of Slug's first 100 times at bat during the 1987 season ("H" = hit, "X" = out):

X X X X X X H X X H X H H X X X X X X X X H X X X X X H H X X X X
X H X X H X H X X X H H X X X X X H X H X X X X H H X H H X X X X X X X X X
X H X X X H X X H X X H X H X X H X X X H X X X.

Now, do Slug's hits tend to come in bunches? That would be the case if he really did have a tendency to do better at some times than at others. Therefore, let us compare Slug's results with those of a deck of cards or a set of random numbers that we know has no tendency to do better at some times than at others.

During this period of 100 times at bat, Slug has averaged one hit in every four times at bat — a .250 batting average. This average is the same as the chance of one card suit's coming up. We designate hearts as "hits" and prepare a deck of 100 cards, twenty-five "H"s (hearts, or "hit") and seventy-five "X"s (other suit, or "out"). Here is the sequence in which the 100 randomly-shuffled cards fell:

X X H X X X X H H X X X H H H X X X X X H X X X H X X H X X X X H X H H X X
X X X X X X X H X X X X X X H H X X X X X H H H X X X X X H X H X H X X H
X H X X X X X X X X X H X X X X X X X H H H X X.

Now we can compare whether or not Slug's hits are bunched up more than they would be by random chance; we can do so by *counting the clusters* (also called "runs") of *one or more* consecutive hits or outs for Slug and for the cards. Notice a cluster (run) can be length 1. So "H H X X X H X H" contains five clusters — they are "H H", "X X X", "H" "X" "H". Notice too that number of clusters will be highest *when there is no clumping* — that is - when "H" and "X" always alternate, as in "H X H X H X ...". Slug had forty-three clusters, which is more than the thirty-seven clusters in the cards; it therefore does not seem that there is a tendency for Slug's hits to clump together. (A larger number of clusters indicates a lower tendency to cluster.)

Of course, the single trial of 100 cards shown above might have an unusually high or low number of clusters. To be safer, lay out, (say,) ten trials of 100 cards each, and compare Slug's number of clusters with the various trials. The proportion of trials with more clusters than Slug's indicates whether or not Slug's hits have a tendency to bunch up. (But caution: This proportion cannot be interpreted directly as a probability.) Now the steps:

- **Step 1.** Constitute a bucket with 3 slips of paper that say "out" and one that says "hit." Or use a table of random numbers and say "01-25" = hits (H), "26-00" = outs (X), Slug's long-run average.
- **Step 2.** Sample 100 slips of paper, with replacement, record "hit" or "out" each time, or write a series of "H's" or "X's" corresponding to 100 numbers, each selected randomly between 1 and 100.
- **Step 3.** Count the number of "clusters," that is, the number of "runs" of the same event, "H"s or "X"s.
- **Step 4.** Compare the outcome in step 3 with Slug's outcome, 43 clusters. If 43 or fewer; write "yes," otherwise "no."
- **Step 5.** Repeat steps 2-4 a hundred times.
- **Step 6.** Compute the proportion "yes." This estimates the probability that Slug's record is not characterized by more "slumps" than would be caused by chance. A very low proportion of "yeses" indicates longer (and hence fewer) "streaks" and "slumps" than would result by chance.

In Python, we can do this experiment 10000 times.

Note 76: Notebook: Do Slugs hits and misses tend to cluster?

- Download notebook
- Interact

```
import numpy as np
import matplotlib.pyplot as plt

# Set up the random number generator.
rnd = np.random.default_rng()
```

First let us work out how to calculate the number of clusters as defined above.

This is Slug's record:

```
slug = np.array([
    'X', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'X', 'H', 'X', 'H', 'H', 'X', 'X',
    'X', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'X',
    'X', 'X', 'H', 'H', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'X', 'H', 'X', 'H',
    'X', 'X', 'X', 'H', 'H', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'H', 'X', 'X',
    'X', 'X', 'H', 'H', 'X', 'H', 'H', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
    'X', 'X', 'H', 'X', 'X', 'X', 'H', 'X', 'X', 'H', 'X', 'X', 'H', 'X', 'H',
    'X', 'X', 'H', 'X', 'X', 'X', 'H', 'X', 'X', 'X'
])
```

Here's how we can calculate the number of runs:

```
slug_runs = 0
current_element = 'start-marker'  # Something that is always != to the elements.
for element in slug:
    if element != current_element:  # The previous cluster has finished.
        slug_runs = slug_runs + 1  # Record we are in a new cluster.
        current_element = element  # Give the starting element for the new cluster.
# Show the count.
slug_runs
```

```
43
```

For practice, let us count the number of clusters in the random set of cards above:

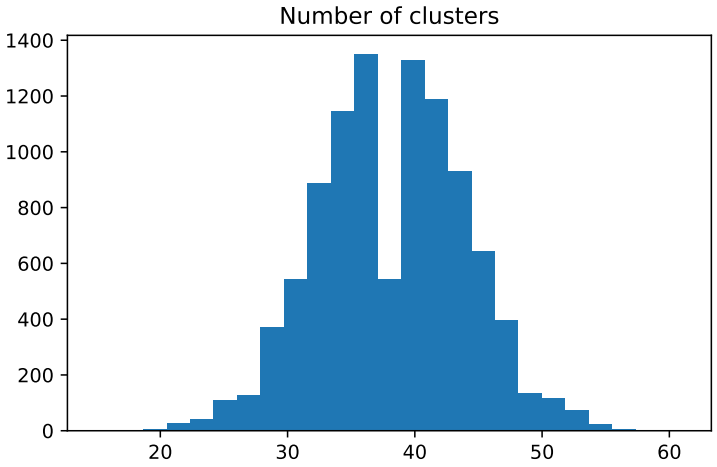
```
random_hand = np.array([
    'X', 'X', 'H', 'X', 'X', 'X', 'X', 'H', 'H', 'X', 'X', 'X', 'H', 'H', 'H',
    'X', 'X', 'X', 'X', 'X', 'H', 'X', 'X', 'X', 'H', 'X', 'X', 'H', 'X', 'X',
    'X', 'X', 'H', 'X', 'H', 'H', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
    'H', 'X', 'X', 'X', 'X', 'X', 'X', 'H', 'H', 'X', 'X', 'X', 'X', 'X', 'H',
    'H', 'H', 'X', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'H', 'X', 'H', 'X', 'X',
    'H', 'X', 'H', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'H', 'X', 'X',
    'X', 'X', 'X', 'X', 'X', 'H', 'H', 'H', 'X', 'X'
])
random_runs = 0
current_element = 'start-marker'  # Something that is always != to the elements.
for element in random_hand:
    if element != current_element:  # The previous cluster has finished.
        random_runs = random_runs + 1  # Record we are in a new cluster.
        current_element = element  # Give the starting element for the new cluster.
# Show the result.
random_runs
```

```
37
```

We will use that code in the procedure below.

```
# Set the number of trials.
n_trials = 10_000

# An empty array to store the trials.
results = np.zeros(n_trials)

for i in range(n_trials):
    # Sample 100 "at-bats" with Slug's own probabilities.
    hits_misses = rnd.choice(['H', 'X'], p=[0.25, 0.75], size=100)
    # How many runs (of any length >=1) are there in the 100 at-bats?
    fake_runs = 0
    current_element = 'start-marker'  # Something always != to the elements.
    for element in hits_misses:
        if element != current_element:  # The previous cluster has finished.
            fake_runs = fake_runs + 1  # Record we are in a new cluster.
            current_element = element  # Give the starting element for the new cluster.
    # Record the result.
    results[i] = fake_runs
```

```
plt.hist(results, bins=25)
plt.title('Distribution of number of clusters in random hit/miss arrays')
plt.title('Number of clusters')
```



Examining the histogram, we see that 43 runs is not at all an unusual occurrence.

> ℹ End of notebook: Do Slugs hits and misses tend to cluster?
>
> `slugs_slumps` starts at Note 76.

The manager wants to look at this matter in a somewhat different fashion, however. He insists that the existence of slumps is proven by the fact that the player sometimes does not get a hit for an abnormally long period of time. One way of testing whether or not the coach is right is by comparing an average player's longest slump in a 100-at-bat season with the longest run of outs in the first card trial. Assume that Slug is a player picked *at random*. Then compare Slug's longest slump — say, 10 outs in a row — with the longest cluster of a single simulated 100-at-bat trial with the cards, 9 outs. This result suggests that Slug's apparent slump might well have resulted by chance.

The estimate can be made more accurate by taking the *average* longest slump (cluster of outs) in ten simulated 400-at-bat trials. But notice that we do *not* compare Slug's slump against the *longest* slump found in ten such simulated trials. We want to know the longest cluster of outs that would be found under *average* conditions, and the hand with the *longest* slump is *not* average or typical. Determining whether to compare Slug's slump with the *average* longest slump or with the *longest* of the ten longest slumps is a decision of crucial importance. There are no mathematical or logical rules to help you. What is required is hard, clear thinking.

Experience can help you think clearly, of course, but these decisions are not easy or obvious even to the most experienced statisticians.

The coach may then refer to the protracted slump of one of the twenty-five players on his team to prove that slumps really occur. But, of twenty-five random 100-at-bat trials, one will contain a slump longer than any of the other twenty-four, and that slump will be considerably longer than average. A fair comparison, then, would be between the longest slump of his longest-slumping player, and the longest run of outs found among twenty-five random trials. In fact, the longest run among twenty-five hands of 100 cards was fifteen outs in a row. And, if we had set some of the hands for lower (and higher) batting averages than .250, the longest slump in the cards would have been even longer.

Research by Roberts and his students at the University of Chicago shows that in fact slumps do *not* exist, as I conjectured in the first publication of this material in 1969. (Of course, a batter feels as if he has a better chance of getting a hit at some times than at other times. After a series of successful at-bats, sandlot players and professionals alike feel confident — just as gamblers often feel that they're on a "streak." But there seems to be no connection between a player's performance and whether he feels hot or cold, astonishing as that may be.)

Averages over longer periods may vary systematically, as Ty Cobb's annual batting average varied non-randomly from season to season, Roberts found. But short-run analyses of day-to-day and week-to-week individual and team performances in most sports have shown results similar to the outcomes that a lottery-type random-number machine would produce.

Remember, too, the study by Gilovich, Vallone, and Twersky of basketball mentioned in Chapter 14. To repeat, their analyses "provided no evidence for a positive correlation between the outcomes of successive shots." That is, knowing whether a shooter has or has not scored on the previous sheet — or in any previous sequence of shots — is useless for predicting whether they will score again.

The species *homo sapiens* apparently has a powerful propensity to believe that one can find a pattern even when there is no pattern to be found. Two decades ago I (JLS) cooked up several series of random numbers that looked like weekly prices of publicly-traded stocks. Players in the experiment were told to buy and sell stocks as they chose. Then I repeatedly gave them "another week's prices," and allowed them to buy and sell again. The players did all kinds of fancy calculating, using a wild variety of assumptions — although there was no possible way that the figuring could help them.

When I stopped the game before completing the 10 buy-and-sell sessions they expected, subjects would ask that the game go on. Then I would tell them that there was no basis to believe that there were patterns in the data, because the "prices" were just randomly-generated numbers. Winning or losing therefore did not depend upon the subjects' skill. Nevertheless, they demanded that the game not stop until the 10 "weeks" had been played, so they could find out whether they "won" or "lost."

This study of batting illustrates how one can test for independence among various trials. The trials are independent if each observation is randomly chosen with replacement from the universe, in which case there is no reason to believe that one observation will be related to the observations directly before and after; as it is said, "the coin has no memory."

The year-to-year level of Lake Michigan is an example in which observations are *not* independent. If Lake Michigan is very high in one year, it is likely to be higher than average the following year because some of the high level carries over from one year into the next.[4] We could test this hypothesis by writing down whether the level in each year from, say, 1860 to 1975 was higher or lower than the median level for those years. We would then count the number of runs of "higher" and "lower" and compare the number of runs of "black" and "red" with a deck of that many cards; we would find fewer runs in the lake level than in an average hand of 116 (1976-1860) cards, though this test is hardly necessary. (But are the *changes* in Lake Michigan's level independent from year to year? If the level went up last year, is there a better than 50-50 chance that the level will also go up this year? The answer to this question is not so obvious. One could compare the numbers of runs of ups and downs against an average hand of cards, just as with the hits and outs in baseball.)

Exercise for students: How could one check whether the successive numbers in a random-number table are independent?

29.7 Exercises

You can find solutions for problems at Appendix A.

29.7.1 Exercise: voter participation

Table 29.14 shows voter participation rates in the various states in the 1844 presidential election. Should we conclude that there was a negative relationship between the participation rate and the vote spread between the parties in the election? (Adapted from (Noreen 1989, 20, Table 2-4):

Table 29.14: Voter Participation In 1844 Presidential Election

State	Participation	Spread
Maine	67.5	13
New Hampshire	65.6	19
Vermont	65.7	18
Massachusetts	59.3	12
Rhode Island	39.8	20

[4]Example from (Wallis and Roberts 1956, 565 – 556).

State	Participation	Spread
Connecticut	76.1	5
New York	73.6	1
New Jersey	81.6	1
Pennsylvania	75.5	2
Delaware	85.0	3
Maryland	80.3	5
Virginia	54.5	6
North Carolina	79.1	5
Georgia	94.0	4
Kentucky	80.3	8
Tennessee	89.6	1
Louisiana	44.7	3
Alabama	82.7	8
Mississippi	89.7	13
Ohio	83.6	2
Indiana	84.9	2
Illinois	76.3	12
Missouri	74.7	17
Arkansas	68.8	26
Michigan	79.3	6

The observed correlation coefficient between voter participation and spread is -0.425. Is this more negative that what might occur by chance, if no correlation exists?

Here's a notebook to get you started.

Note 77: Notebook: Voter participation in 1844 election

- Download zip with notebook + data file
- Interact

```
import numpy as np
import pandas as pd

rnd = np.random.default_rng()

voter_df = pd.read_csv('data/election_1844.csv')
participation = np.array(voter_df['Participation'])
spread = np.array(voter_df['Spread'])
```

ℹ End of notebook: Voter participation in 1844 election

`voter_participation_exercise` starts at Note 77.

See: Section A.6.

29.7.2 Exercise: association of runs and strikeouts

We would like to know whether, among major-league baseball players, home runs (per 500 at-bats) and strikeouts (per 500 at-bat's) are correlated. For this exercise, you should use the sum-of-products procedure as used above for I.Q. and athletic ability — multiplying the elements within each pair. The next exercise uses the more "sophisticated" measure, the correlation coefficient.

The data for 18 randomly-selected players in the 1989 season are as follows, as they would appear in the first lines of the notebook.

ℹ Note 78: Notebook: Homeruns and strikeout sum of products.

- Download notebook
- Interact

```
import numpy as np

rnd = np.random.default_rng()

homeruns = np.array([14, 20, 0, 38, 9, 38, 22, 31, 33,
                     11, 40, 5, 15, 32, 3, 29, 5, 32])
strikeout = np.array([135, 153, 120, 161, 138, 175, 126, 200, 205,
                      147, 165, 124, 169, 156, 36, 98, 82, 131])

# Exercise - complete this program.
```

ℹ End of notebook: Homeruns and strikeout sum of products.

`homerun_sop_exercise` starts at Note 78.

See: Section A.7.

29.7.3 Exercise: runs, strikeouts, correlation coefficient

In the previous example relating strikeouts and home runs, we used the procedure of multiplying the elements within each pair. Now we use a more "sophisticated" measure, the correlation coefficient, which is simply a standardized form of the multiplicands, but sufficiently well known that we calculate it with a pre-set command.

Exercise: Write a program that uses the correlation coefficient to test the significance of the association between home runs and strikeouts. You can use the starting notebook for the previous exercise.

See: Section A.8.

29.7.4 Exercise: money and exchange rate

All the other things equal, an increase in a country's money supply is inflationary and should have a negative impact on the exchange rate for the country's currency. The data in the following table (Table 29.15) were computed using data from tables in the 1983/1984 *Statistical Yearbook of the United Nations*. The table shows the first 15 rows.

Table 29.15: % change in exchange rates and money supply 1983-1984

	Exchange rate	Money supply
Australia	0.089	0.035
Belgium	0.134	0.003
Botswana	0.351	0.085
Burma	0.064	0.155
Burundi	0.064	0.064
Canada	0.062	0.209
Chile	0.465	0.126
China	0.411	0.555
Costa Rica	0.100	0.100
Cyprus	0.158	0.044
Denmark	0.140	0.351
Ecuador	0.242	0.356
Fiji	0.093	0.000
Finland	0.124	0.164
France	0.149	0.090
Germany	0.156	0.061
Greece	0.302	0.202
Hungary	0.133	0.049
India	0.187	0.184
Indonesia	0.080	0.132

	Exchange rate	Money supply
Italy	0.167	0.124
Jamaica	0.504	0.237
Japan	0.081	0.069
Jordan	0.092	0.010
Kenya	0.144	0.141
Korea	0.040	0.006
Kuwait	0.038	-0.180
Lebanon	0.619	0.065
Madagascar	0.337	0.244
Malawi	0.205	0.203
Malaysia	0.037	-0.006
Malta	0.003	0.003
Mauritania	0.180	0.192
Mauritius	0.226	0.136
Mexico	0.338	0.599
Morocco	0.076	0.076
Netherlands	0.158	0.078
New Zealand	0.370	0.098
Nigeria	0.079	0.082
Norway	0.177	0.242
Papua	0.075	0.209
Philippines	0.411	0.035
Portugal	0.288	0.166
Romania	-0.029	0.039
Rwanda	0.059	0.083
Samoa	0.348	0.118
Saudi Arabia	0.023	0.023
Seychelles	0.063	0.031
Singapore	0.024	0.030
Solomon Is	0.101	0.526
Somalia	0.481	0.238
South Africa	0.624	0.412
Spain	0.107	0.086
Sri Lanka	0.051	0.141
Switzerland	0.186	0.186
Tunisia	0.193	0.068
Turkey	0.573	0.181
UK	0.255	0.154
USA	0.000	0.156
Vanatuva	0.008	0.331
Yemen	0.253	0.247

	Exchange rate	Money supply
Yugoslavia	0.685	0.432
Zaire	0.343	0.244
Zambia	0.457	0.094
Zimbabwe	0.359	0.164

Are changes in the exchange rates and in money supplies related to each other? That is, are they correlated?

Should the algorithm of non-computer resampling steps be similar to the algorithm for I.Q. and athletic ability shown in the text? One can also work with the correlation coefficient rather then the sum-of-products method, and expect to get the same result.

1. Write a series of non-computer resampling steps to solve this problem.
2. Write a Python program to implement those steps.

Here's a notebook to get you started on part 2:

> **Note 79: Notebook: Exchange rates and money supply**
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np
import pandas as pd

rnd = np.random.default_rng()

exchange_df = pd.read_csv('data/exchange_rates.csv')
exchange_rates = np.array(exchange_df['exchange_rate'])
money_supply = np.array(exchange_df['money_supply'])
```

> **End of notebook: Exchange rates and money supply**
>
> `exchange_rates_exercise` starts at Note 79.

See: Section A.9.

30 How Large a Sample?

30.1 Issues in determining sample size

Sometime in the course of almost every study — preferably early in the planning stage — the researcher must decide how large a sample to take. Deciding the size of sample to take is likely to puzzle and distress you at the beginning of your research career. You have to decide somehow, but there are no simple, obvious guides for the decision.

For example, one of the first studies I worked on was a study of library economics (Fussler and Simon 1961), which required taking a sample of the books from the library's collections. Sampling was expensive, and we wanted to take a correctly sized sample. But how large should the sample be? The longer we searched the literature, and the more people we asked, the more frustrated we got because there just did not seem to be a clear-cut answer. Eventually we found out that, even though there are some fairly rational ways of fixing the sample size, most sample sizes in most studies are fixed simply (and irrationally) by the amount of money that is available or by the sample size that similar research has used in the past.

The rational way to choose a sample size is by weighing the benefits you can expect in information against the cost of increasing the sample size. In principle you should continue to increase the sample size until the benefit and cost of an additional sampled unit are equal.[1]

The benefit of additional information is not easy to estimate even in applied research, and it is extraordinarily difficult to estimate in basic research. Therefore, it has been the practice of researchers to set up target goals of the *degree of accuracy* they wish to achieve, or to consider various degrees of accuracy that might be achieved with various sample sizes, and then to balance the degree of accuracy with the cost of achieving that accuracy. The bulk of this chapter is devoted to learning how the sample size is related to accuracy in simple situations.

In complex situations, however, and even in simple situations for beginners, you are likely to feel frustrated by the difficulties of relating accuracy to sample size, in which case you cry out to a supervisor, "Don't give me complicated methods, just give me a rough number based on your greatest experience." My inclination is to reply to you, "Sometimes life is hard and there is no shortcut." On the other hand, perhaps you can get more information than misinformation out of knowing sample sizes that have been used in other studies. Table 30.1 shows the middle

[1] Schlaifer (1961) attacks the sample-size problem in the wider context of decision making, costs, and benefits. The statistically knowledgeable reader can find an excellent discussion of sample size in Hansen *et al.* (1953).

(modal), 25th percentile, and 75th percentile scores for — please keep this in mind — *National Opinion Surveys* Table 30.2 shows how subgroup analyses affect sample size. The source for both tables is *Applied Sampling*, by Seymour Sudman (1976, 86–87) copyright Academic Press, reprinted by permission.

Pretest sample sizes are smaller, of course, perhaps 25-100 observations. Samples in research for Master's and Ph.D. theses are likely to be closer to a pretest than to national samples.

Table 30.1: Common sample sizes for national and regional studies by subject

Subject Matter	National			Regional		
	Mode	Q3	Q1	Mode	Q3	Q1
Financial	1000+	—	—	100	400	50
Medical	1000+	1000+	500	1000+	1000+	250
Other Behavior	1000+	—	—	700	1000	300
Attitudes	1000+	1000+	500	700	1000	400
Laboratory Experiments	—	—	—	100	200	50

Table 30.2: Typical sample sizes for studies of human and institutional populations

Subgroup analysis	People or households		Institutions	
	National	Special	National	Special
None or few	1000-1500	200-500	200-500	50-200
Average	1500-2500	500-1000	500-1000	200-500
Many	2500+	1000+	1000+	500+

Once again, the sample size ought to depend on the proportions of the sample that have the characteristics you are interested in, the extent to which you want to learn about subgroups as well as the universe as a whole, and of course the purpose of your study, the value of the information, and the cost. Also, keep in mind that the *added* information that you obtain from an additional sample observation tends to be smaller as the sample size gets larger. You must quadruple the sample to halve the error.

Now let us consider some specific cases. The first examples taken up here are from the descriptive type of study, and the latter deal with sample sizes in relationship research.

30.2 Some practical examples

30.2.1 Example: what proportion of homes are listening to a radio station?

What proportion of the homes in Countryville watch television station WCNT's ten o'clock news program? That is the question your phone survey aims to answer, and you want to know how many randomly selected homes you must phone to obtain a sufficiently large sample.

Begin by guessing the likeliest answer, say 30 percent in this case. Do not worry if you are off by 5 per cent or even 10 per cent; and you will probably not be further off than that. Select a first-approximation sample size of perhaps 400; this number is selected from my general experience, but it is just a starting point. Then proceed through the first 400 numbers in a random-number table, marking down a *yes* for numbers 1-3 and *no* for numbers 4-10 (because 3/10 was your estimate of the proportion listening). Then add the number of *yes* and *no*. Carry out perhaps ten sets of such trials, the results of which are in Table 30.3.

Table 30.3: Ten example trials from phone survey simulation

Trial number	Number "yes"	Number "no"	% difference from expected mean of 30% (120 "yes")
1	115	285	1.25
2	119	281	0.25
3	116	284	1.00
4	114	286	1.50
5	107	293	3.25
6	116	284	1.00
7	132	268	3.00
8	123	277	0.75
9	121	279	0.25
10	114	286	1.50
Mean			1.375

Based on these ten trials, you can estimate that if you take a sample of 400 and if the "real" viewing level is 30 percent, your average percentage error will be 1.375 percent on either side of 30 percent. That is, with a sample of 400, half the time your error will be greater than 1.375 percent if 3/10 of the universe is listening.

Now you must decide whether the estimated error is small enough for your needs. If you want greater accuracy than a sample of 400 will give you, increase the sample size, using this important rule of thumb: To cut the error in half, you must *quadruple* the sample size. In other words, if you want a sample that will give you an error of only 0.6875 percent on the

average, you must increase the sample size to 1,600 interviews. Similarly, if you cut the sample size to 100, the average error will be only 2.75 percent (double 1.375 percent) on either side of 30 percent. If you distrust this rule of thumb, run ten or so trials on sample sizes of 100 or 1,600, and see what error you can expect to obtain on the average.

If the "real" viewership is 20 percent or 40 percent, instead of 30 percent, the accuracy you will obtain from a sample size of 400 will not be very different from an "actual" viewership of 30 percent, so do not worry about that too much, as long as you are in the right general vicinity.

Accuracy is *slightly* greater in smaller universes but *only* slightly. For example, a sample of 400 would give *perfect* accuracy if Countryville had only 400 residents. And a sample of 400 will give *slightly* greater accuracy for a town of 800 residents than for a city of 80,000 residents. But, beyond the point at which the sample is a *large fraction* of the total universe, there is no difference in accuracy with increases in the size of universe. This point is very important. For any given level of accuracy, *identical* sample sizes give the same level of accuracy for Podunk (population 8,000) or New York City (population 8 million). The *ratio* of the sample size to the population of Podunk or New York City means nothing at all, even though it intuitively seems to be important.

The size of the sample must depend upon which population or sub-populations you wish to describe. For example, Alfred Kinsey's sample size for the classic "Sexual Behavior in the Human Male" (1948) would have seemed large, by customary practice, for generalizations about the United States population as a whole. But, as Kinsey explains: "... the chief concern of the present study is an understanding of the sexual behavior of *each segment of the population*, and that it is only secondarily concerned with generalization for the population as a whole." (1948, 82, italics added). Therefore Kinsey's sample had to include sub-samples large enough to obtain the desired accuracy in *each* of these sub-universes. The U.S. Census offers a similar illustration. When the U.S. Bureau of the Census aims to estimate only a total or an average for the United States as a whole — as, for example, in the Current Population Survey estimate of unemployment — a sample of perhaps 50,000 is big enough. But the decennial census aims to make estimates for all the various communities in the country, estimates that require adequate sub-samples in each of these sub-universes; such is the justification for the decennial census' sample size of so many millions. Television ratings illustrate both types of purpose. Nielsen ratings, for example, are sold primarily to national network advertisers. These advertisers on national television networks usually sell their goods all across the country and are therefore interested primarily in the total United States viewership for a program, rather than in the viewership in various demographic subgroups. The appropriate calculations for Nielsen sample size will therefore refer to the total United States sample. But other organizations sell rating services to *local* television and radio stations for use in soliciting advertising over the local stations rather than over the network as a whole. Each local sample must then be large enough to provide reasonable accuracy, and, considered as a whole, the samples for the local stations therefore add up to a much larger sample than the Nielsen and other nationwide samples.

The problem may be handled with the following Python program. This program represents viewers with the string `'viewers'` and non-viewers as `'not viewers'`. It then asks `rnd.choice` to choose randomly between `'viewer'` and `'not viewer'` with a 30% (p=0.3) chance of getting a `'viewer'` and a 70% chance of getting a `'not viewer'`. It gets a sample of 400 such numbers, counts (with `np.sum` the "viewers" then finds how much this sample diverges from the expected number of viewers (30% of 400 = 120). It repeats this procedure 10000 times, and then calculates the average divergence.

Note 80: Notebook: Number of viewers

- Download notebook
- Interact

```
import numpy as np

# set up the random number generator
rnd = np.random.default_rng()
```

```
# set the number of trials
n_trials = 10000

# an empty array to store the results
results = np.zeros(n_trials)

# What are the options to choose from?
options = ['viewer', 'not viewer']

# do n_trials trials
for i in range(n_trials):

    # Choose 'viewer' 30% of the time.
    a = rnd.choice(options, size=400, p=[0.3, 0.7])

    # count the viewers
    b = np.sum(a == 'viewer')

    # how different from expected?
    c = 120 - b

    # absolute value of the difference
    d = np.abs(c)
```

```
    # express as a proportion of sample
    e = d / 400

    # keep score of the result
    results[i] = e

# find the mean divergence
k = np.mean(results)

# Show the result
k
```

```
np.float64(0.018184000000000002)
```

> **i** End of notebook: Number of viewers
>
> `viewer_numbers` starts at Note 80.

It is a simple matter to go back and try a sample size of (say) 1600 rather than 400, and examine the effect on the mean difference.

30.2.2 Example: average weight gain for pig rations

This example, like Section 30.2.1, illustrates the choice of sample size for estimating a summarization statistic. Later examples deal with sample sizes for probability statistics.

Hark back to the pig-ration problems presented earlier (e.g. Section 24.0.1), and consider the following set of pig weight-gains recorded for ration A: 31, 34, 29, 26, 32, 35, 38, 34, 31, 29, 32, 30. Assume that our purpose now is to estimate the average weight gain for ration A, so that the feed company can advertise to farmers how much weight gain to expect from ration A. If the universe is made up of pig weight-gains like those we observed, we can simulate the universe with, say, 1 million weight gains of thirty-one pounds, 1 million of thirty-four pounds, and so on for the twelve observed weight gains. Or, more conveniently, as accuracy will not be affected much, we can make up a universe of say, thirty cards for each thirty-one-pound gain, thirty cards for each thirty-four-pound gains and so forth, yielding a deck of 30 x 12 = 360 cards. Then shuffle, and, just for a starting point, try sample sizes of twelve pigs. The means of the samples for twenty such trials are as in Table 30.4.

Table 30.4: Simulated average weight gains from pig ration A

Trial	Mean	Absolute deviation of trial mean from actual mean	Trial	Mean	Absolute deviation of trial mean from actual mean
1	31.77	.02	11	32.10	.35
2	32.27	1.52	12	30.67	1.08
3	31.75	.00	13	32.42	.67
4	30.83	.92	14	30.67	1.08
5	30.52	1.23	15	32.25	.50
6	31.60	.15	16	31.60	.15
7	32.46	.71	17	32.33	.58
8	31.10	.65	18	33.08	1.33
9	32.42	.35	19	33.01	1.26
10	30.60	1.15	20	30.60	1.15
Mean				31.75	

Now ask yourself whether a sample size of twelve pigs gives you enough accuracy. If we sort the absolute deviations, we find the middle two values (the 10th and 11th values) of 20 are 0.67 and 0.71, so the median is the average of these values: 0.69. There is a .5 chance that the mean for any given sample will be more than 0.69 points from the mean of the universe that generates such samples, which in this situation is 31.75 pounds. Is this close enough? That is up to you to decide in light of the purposes for which you are running the experiment. (The logic of the inference you make here is inevitably murky, and use of the term "real mean" can make it even murkier, as is seen in the discussion in Chapter 26 — Chapter 28 on confidence intervals.)

To see how accuracy is affected by larger samples, try a sample size of forty-eight "pigs" dealt from the same deck. (But, if the sample size were to be much larger than forty-eight, you might need a "universe" greater than 360 cards.) The results of twenty trials are in Table 30.5.

Table 30.5: Simulated average weight gain from 48 pigs

Trial	Mean	Absolute deviation of trial mean from actual mean	Trial	Mean	Absolute deviation of trial mean from actual mean
1	31.80	.05	11	31.93	.18

Trial	Mean	Absolute deviation of trial mean from actual mean	Trial	Mean	Absolute deviation of trial mean from actual mean
2	32.27	.52	12	32.40	.65
3	31.82	.07	13	31.32	.43
4	31.39	.36	14	32.07	.68
5	31.22	.53	15	32.03	.28
6	31.88	.13	16	31.95	.20
7	31.37	.38	17	31.75	.00
8	31.48	.27	18	31.11	.64
9	31.20	.55	19	31.96	.21
10	32.01	.26	20	31.32	.43
Mean				31.75	

The median of mean absolute deviations in Table 30.5 is 0.32. In half the trials with a sample size of forty-eight the difference between the sample mean and the "real" mean of 31.75 will be .32 pound, smaller than with the 0.69 of samples of 12 pigs. Again, is this too little accuracy for you? If so, increase the sample size further.

The attentive reader of this example may have been troubled by this question: How do you know what kind of a distribution of values is contained in the universe before the sample is taken? The answer is that you guess, just as in Section 30.2.1 you guessed at the mean of the universe. If you guess wrong, you will get either more accuracy or less accuracy than you expected from a given sample size, but the results will not be fatal; if you obtain more accuracy than you wanted, you have wasted some money, and, if you obtain less accuracy, your sample dispersion will tell you so, and you can then augment the sample to boost the accuracy. But an error in guessing will not introduce error into your final results.

The guess should be based on something, however. One source for guessing is your general knowledge of the likely dispersion; for example, if you were estimating male heights in Rhode Island, you would be able to guess what proportion of observations would fall within 2 inches, 4 inches, 6 inches, and 8 inches, perhaps, of the real value. Or, much better yet, a very small pretest will yield quite satisfactory estimates of the dispersion.

Here is a Python program that will let you try different sample sizes, and then take bootstrap samples to determine the range of sampling error. You set the sample size by setting the `sampsize` variable. Above I noted that we could sample without replacement from a "deck" of thirty "31" 's, thirty "34" 's, etc, as a substitute for creating a universe of a million "31" 's, a million "34" 's, etc. We can achieve the same effect if we replace each card after we sample it; this is equivalent to creating a "deck" of an infinite number of "31" 's, "34" 's, etc. That is

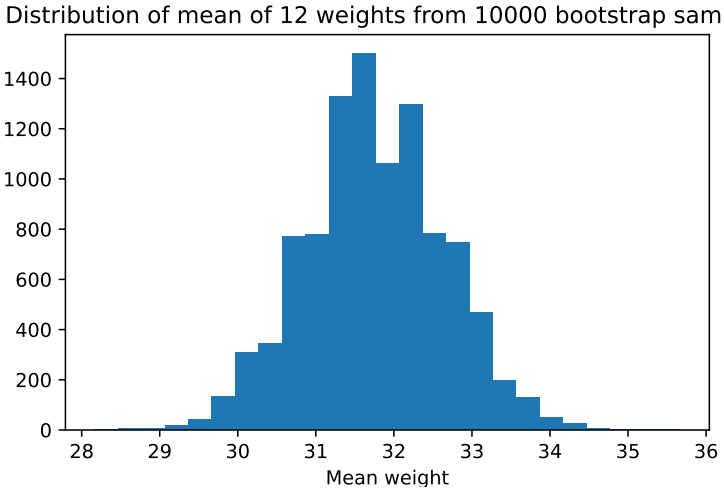
what the `rnd.choice` command does, below. Note that the sample size is determined by the value of the `sampsize` variable, which you set at the beginning. From here on the program takes the `mean` of each sample, keeps score of that result in the `results` array and produces a histogram. The `quantile` function will also tell you what values enclose 90% of all sample results, excluding those below the 5th percentile and above the 95th percentile.

Here is a notebook for a sample size of 12.

> **Note 81: Notebook: Sampling error for pig ration weight gain via bootstrap**
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

sampsize = 12

gains = np.array([31, 34, 29, 26, 32, 35, 38, 34, 32, 31, 30, 29])

n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    fake_gains = rnd.choice(gains, size=sampsize)
    results[i] = np.mean(fake_gains)

plt.hist(results, bins=25)
plt.title('Distribution of mean of ' + str(sampsize) +
          ' weights from ' + str(n_trials) + ' bootstrap samples')
plt.xlabel('Mean weight')

# Find values such that to 5%, 95% of values are below given value.
# (These are the 5% and 95% percentile values).
hi_lo_range = np.quantile(results, [0.05, 0.95])

print('5% and 95% percentiles of bootstrap means:', np.round(hi_lo_range, 2))
```

```
5% and 95% percentiles of bootstrap means: [30.25 33.25]
```



End of notebook: Sampling error for pig ration weight gain via bootstrap

`sampling_error_bootstrap` starts at Note 81.

30.3 Example: sample size for inference on fruit fly sex difference

This is the first example of sample-size estimation for *probability* (testing) statistics, rather than the summarization statistics dealt with above.

Recall the problem of the sex of fruit-fly offspring discussed in Section 21.2.1. The question now is, how large a sample is needed to determine whether the radiation treatment results in a sex ratio other than a 50-50 male-female split?

The first step is, as usual, difficult but necessary. As the researcher, you must *guess* what the sex ratio will be if the treatment *does* have an effect. Let's say that you use all your general knowledge of genetics and of this treatment and that you guess the sex ratio will be 75 percent males and 25 percent females *if* the treatment alters the ratio from 50-50.

In table of random numbers from 00-99 let "01-25" stand for females and "26-00" for males. Take twenty successive pairs of numbers for each trial, and run perhaps fifty trials, as in Table 30.6.

Table 30.6: Simulated numbers of males and females for 75/25% universe

Trial	Females	Males	Trial	Females	Males	Trial	Females	Males
1	4	16	18	7	13	34	4	16

Trial	Females	Males	Trial	Females	Males	Trial	Females	Males
2	6	14	19	3	17	35	6	14
3	6	14	20	7	13	36	3	17
4	5	15	21	4	16	37	8	12
5	5	15	22	4	16	38	4	16
6	3	17	23	5	15	39	3	17
7	7	13	24	8	12	40	6	14
8	6	14	25	4	16	41	5	15
9	3	17	26	1	19	42	2	18
10	2	18	27	5	15	43	8	12
11	6	14	28	3	17	44	4	16
12	1	19	29	8	12	45	6	14
13	6	14	30	8	12	46	5	15
14	3	17	31	5	15	47	3	17
15	1	19	32	3	17	48	5	15
16	5	15	33	4	16	49	3	17
17	5	15				50	5	15

In Section 21.2.1 with a sample of twenty flies that contained fourteen or more males, we found only an 8% probability that such an extreme sample would result from a 50-50 universe. Therefore, if we observe such an extreme sample, we rule out a 50-50 universe.

Now Table 30.6 tells us that, if the ratio is *really* 75 to 25, then a sample of twenty will show fourteen or more males forty-two of fifty times (84 percent of the time). If we take a sample of twenty flies and if the ratio is really 75-25, we will make the correct decision by deciding that the split is not 50-50, 84 percent of the time.

Perhaps you are not satisfied with reaching the right conclusion only 84 percent of the time. In that case, still assuming that the ratio will really be 75-25 if it is not 50-50, you need to take a sample larger than twenty flies. How much larger? That depends on how much surer you want to be. Follow the same procedure for a sample size of perhaps eighty flies. First work out for a sample of eighty, as was done in Section 21.2.1 for a sample of twenty, the number of males out of eighty that you would need to find for the odds to be, say, 9 to 1 that the universe is not 50-50; your estimate turns out to be forty-eight males. Then run fifty trials of eighty flies each on the basis of 75-25 probability, and see how often you would not get as many as forty-eight males in the sample. Table 30.7 shows the results we got. No trial was anywhere near as low as forty-eight, which suggests that a sample of eighty is larger than necessary if the split is really 75-25.

Table 30.7: Simulated males / females from 80 flies for 75/25% universe

Trial	Females	Males	Trial	Females	Males	Trial	Females	Males
1	21	59	18	13	67	34	21	59
2	22	58	19	19	61	35	17	63
3	13	67	20	17	63	36	22	58
4	15	65	21	17	63	37	19	61
5	22	58	22	18	62	38	21	59
6	21	59	23	26	54	39	21	59
7	13	67	24	20	60	40	21	59
8	24	56	25	16	64	41	21	59
9	16	64	26	22	58	42	18	62
10	21	59	27	16	64	43	19	61
11	20	60	28	21	59	44	17	63
12	19	61	29	22	58	45	13	67
13	21	59	30	21	59	46	16	64
14	17	63	31	22	58	47	21	59
15	22	68	32	19	61	48	16	64
16	22	68	33	10	70	49	17	63
17	17	63				50	21	59

It is obvious that, if the split you guess at is 60 to 40 rather than 75 to 25, you will need a bigger sample to obtain the "correct" result with the same probability. For example, run some eighty-fly random-number trials with 1-40 representing males and 51-100 representing females. Table 30.8 shows that only twenty-four of fifty (48 percent) of the trials reach the necessary cut-off at which one would judge that a sample of eighty really does not come from a universe that is split 50-50; therefore, a sample of eighty is not big enough if the split is 60-40.

Table 30.8: Simulated males / females from 80 flies for 60/40% universe

Trial	Females	Males	Trial	Females	Males	Trial	Females	Males
1	35	45	18	32	48	34	35	45
2	36	44	19	28	52	35	36	44
3	35	45	20	32	48	36	29	51
4	35	45	21	33	47	37	36	44
5	36	44	22	37	43	38	36	44
6	36	44	23	36	44	39	31	49
7	36	44	24	31	49	40	29	51
8	34	46	25	27	53	41	30	50
9	34	46	26	30	50	42	35	45
10	29	51	27	31	49	43	32	48

Trial	Females	Males	Trial	Females	Males	Trial	Females	Males
11	29	51	28	33	47	44	30	50
12	32	48	29	37	43	45	37	43
13	29	51	30	30	50	46	31	49
14	31	49	31	31	49	47	36	44
15	28	52	32	32	48	48	34	64
16	33	47	33	34	46	49	29	51
17	36	44				50	37	43

To review the main principles of this example: First, the closer together the two possible universes from which you think the sample might have come (50-50 and 60-40 are closer together than are 50-50 and 75-25), the larger the sample needed to distinguish between them. Second, the surer you want to be that you reach the right decision based upon the sample evidence, the larger the sample you need.

The problem may be handled with the following Python notebook. We construct a benchmark universe that is 60-40 male-female, and take samples of size 80, observing whether the numbers of males and females differs enough in these resamples to rule out a 50-50 universe. Recall that we need at least 48 of 80 males to say that the proportion of males is *not* 50%.

Note 82: Notebook: Sample size for detecting fruitfly sex difference

- Download notebook
- Interact

```
import numpy as np

rnd = np.random.default_rng()

n_trials = 10_000

# Results for each trial.
results = np.zeros(n_trials)

# Do 10,000 trials
for i in range(n_trials):
    # Generate 80 "flies" with 0.6 chance of male, 0.4 of female>
    flies = rnd.choice(['male', 'female'], size=80, p=[0.6, 0.4])
    # Count the males.
    n_males = np.sum(flies == 'male')
    # Keep score.
```

```
    results[i] = n_males

# How many of the trials produced more than 48 males?
k = np.sum(results >= 48)
# Convert to a proportion
kk = k / n_trials

print('Proportion of 60/40 trials giving >= 48 males:', kk)
```

```
Proportion of 60/40 trials giving >= 48 males: 0.5534
```

If the result `kk` is close to 1, we then know that samples of size 80 will almost always produce samples with enough males to avoid misleading us into thinking that they could have come from a universe in which males and females are split 50-50.

> **i** End of notebook: Sample size for detecting fruitfly sex difference
>
> `flies_sample_size` starts at Note 82.

30.3.1 Example: sample size for an internet-provider poll

Referring back to Section 21.2.3, on the internet provider poll, how large a sample *should* you have taken? Pretend that the data have not yet been collected. You need *some* estimate of how the results will turn out before you can select a sample size. But you have not the foggiest idea how the results will turn out. Therefore, go out and take a very small sample, maybe ten people, to give you some idea of whether people will split quite evenly or unevenly. Seven of your ten initial interviews say they are for the internet provider contract. How large a sample do you now need to provide an answer of which you can be fairly sure?

Using the techniques of the previous chapter, we can estimate that from a sample of fifty people at least thirty-two would have to vote the same way for you to believe that the odds are at least 19 to 1 that the sample does not misrepresent the universe, that is, that the sample does not show a majority different from that of the whole universe if you polled everyone.

We do this by repeating the experiment in Section 21.2.3. In that experiment, we were doing simulated trials in the 50:50 world, and looking at the proportion of simulated trials where the count of yes votes was >= 30. We found the proportion of trials was about 10%. We are interested to know the count for which we get less than 5%. We can do this by checking the proportions for >=31, >= 32, and so on. It turns out that counts >= 32 occur a bit less than 5% of the time in the 50:50 world.

Note 83: Notebook: Contract poll looking for >=32

- Download notebook
- Interact

This Python notebook generates samples of 50 simulated voters on the assumption that only 50 percent are in favor of the contract. Then it counts the number of generated samples where 32 or more of the 50 respondents said they were in favor of the contract.

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

n_trials = 10_000

yeses = np.zeros(n_trials)

for i in range(n_trials):
    answers = rnd.choice(['No', 'Yes'], size=50)
    yeses[i] = np.sum(answers == 'Yes')

k = np.sum(yeses >= 32)
kk = k / n_trials

print('Proportion >= 32:', np.round(kk, 2))
```

```
Proportion >= 32: 0.03
```

End of notebook: Contract poll looking for >=32

`contract_poll_32` starts at Note 83.

We know, therefore, that if we see a voter "yes" count >= 32, there is only a small (>5%) chance that arose from the 50:50 world.

Therefore, designate numbers 1-30 as *no* and 31-00 as *yes* in the random-number table (that is, 70 percent, as in your estimate based on your presample of ten), work through a trial sample size of fifty, and count the number of *yeses*. Run through perhaps ten or fifteen trials, and reckon how often the observed number of *yeses* is >= 32 (the number you must exceed for a result you can rely on). In Table 30.9 we see that a sample of fifty respondents, from a

universe split 70-30, will show that many *yeses* a preponderant proportion of the time — in fact, in fifteen of fifteen experiments; therefore, the sample size of fifty is large enough if the split is "really" 70-30.

Table 30.9: Number of "yes" votes out of 50 for 30% in favor universe

Trial	No	Yes	Trial	No	Yes
1	13	37	9	15	35
2	14	36	10	9	41
3	18	32	11	15	35
4	10	40	12	15	35
5	13	37	13	9	41
6	15	35	14	16	34
7	14	36	15	17	33

The following Python program takes samples of size 50 from a universe that is 70% "yes." It then observes how often such samples produce more than 31 "yeses" — the number we must get if we are to be sure enough that the sample is not from a 50/50 universe.

Note 84: Notebook: Sample size for an internet contract poll

- Download notebook
- Interact

```
import numpy as np

rnd = np.random.default_rng()

# Number of trials.
n_trials = 10_000

# Make array to store results for each trial.
results = np.zeros(n_trials)

# Do 10,00 trials
for i in range(n_trials):
    # Generate 50 voters with 70% chance of "yes".
    voters = rnd.choice(["yes", "no"], size=50, p=[0.7, 0.3])
    # Count the "yeses".
    n_yes = np.sum(voters == 'yes')
    # Keep score of the result.
```

```
    results[i] = n_yes

# Count how often the sample result >= our 32 cutoff (recall that samples
# with 31 or fewer "yeses" cannot be ruled out of a 50/50 universe).
k = np.sum(results >= 32)
# Convert to a proportion
kk = k / n_trials

print('p of 50 voter samples in 70:30 universe >= 32 "yes":', kk)
```

```
p of 50 voter samples in 70:30 universe >= 32 "yes": 0.859
```

> End of notebook: Sample size for an internet contract poll
>
> `poll_sample_size` starts at Note 84.

If `kk` is close to 1, we can be confident that this sample will be large enough to avoid a result that we might mistakenly think comes from a 50/50 universe (provided that the real universe is 70% favorable).

30.3.2 Example: how large a sample for pig rations?

How large a sample is needed to determine whether there is any difference between the two pig rations in Section 24.0.1? The first step is to guess the results of the tests. You estimate that the average for ration A will be a weight gain of 32 pounds. You further guess that twelve pigs on ration A might gain 36, 35, 34, 33, 33, 32, 32, 31, 31, 30, 29 and 28 pounds. This set of guesses has an equal number of pigs above and below the average and more pigs close to the average than farther away. That is, there are more pigs at 33 and 31 pounds than at 36 and 28 pounds. This would seem to be a reasonable distribution of pigs around an average of 32 pounds. In similar fashion, you guess an average weight gain of 28 pounds for ration B and a distribution of 32, 31, 30, 29, 29, 28, 28, 27, 27, 26, 25, and 24 pounds.

Let us review the basic strategy. We want to find a sample size large enough so that a large proportion of the time it will reveal a difference between groups big enough to be accepted as not attributable to chance. First, then, we need to find out how big the difference must be to be accepted as evidence that the difference is not attributable to chance. We do so from trials with samples of the given size from the benchmark universe. We state that a difference larger than the benchmark universe will usually produce is not attributable to chance.

In this case, let us try samples of 12 pigs on each ration. First we draw two samples from a *combined* benchmark universe made up of the results that we have guessed will come from ration A and ration B. (The procedure is the same as was followed in Section 24.0.1).

> **Note 85: Notebook: Sample size for pig rations with bootstrap**
>
> - Download notebook
> - Interact

First we need to get the measured data from the data file using the Pandas library:

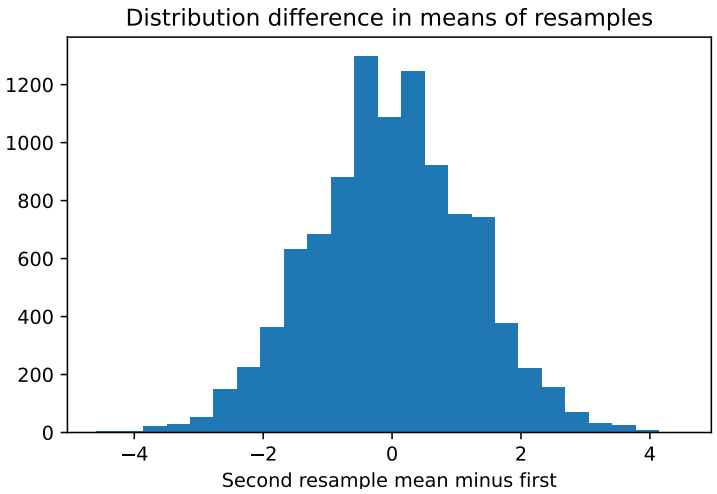
```
import numpy as np
import matplotlib.pyplot as plt

# set up the random number generator
rnd = np.random.default_rng()

# Estimated weights for ration A.
a_weights = np.array([36, 35, 34, 33, 33, 32, 32, 31, 31, 30, 29, 28])
# Estimated weights for ration B.
b_weights = np.array([32, 31, 30, 29, 29, 28, 28, 27, 27, 26, 25, 24])

# Make a combined (benchmark) universe from the weights.
both = np.concatenate([a_weights, b_weights])

# Set the number of trials
n_trials = 10_000

# An empty array to store the trial results.
results = np.zeros(n_trials)

# Do 10,000 experiments.
for i in range(n_trials):
    # Take a "resample" of 12 with replacement from both and put it in fake_a
    fake_a = rnd.choice(both, size=12)
    # Likewise to make fake_b
    fake_b = rnd.choice(both, size=12)
    # Mean of the first "resample" sample.
    fake_a_mean = np.mean(fake_a)
    # Mean of the second "resample" sample.
    fake_b_mean = np.mean(fake_b)
    # Calculate the difference between the two resamples.
    fake_diff = fake_a_mean - fake_b_mean
```

```
    # Keep track of each trial result.
    results[i] = fake_diff
    # End one experiment, go back and repeat until all trials are complete,
    # then proceed.
# Produce a histogram of trial results.
plt.hist(results, bins=25)
plt.xlabel('Second resample mean minus first')
plt.title('Distribution difference in means of resamples')

# Get the 95% percentile.  Only 5% of results are above this value, by chance.
q_95 = np.quantile(results, 0.95)

print('95% quantile for resampled mean difference:', q_95)
```

```
95% quantile for resampled mean difference: 2.0
```

```
# Check the quantile.
k = np.sum(results > q_95)
kk = k / n_trials

print('Proportion > 95% quantile:', kk)
```

```
Proportion > 95% quantile: 0.0439
```

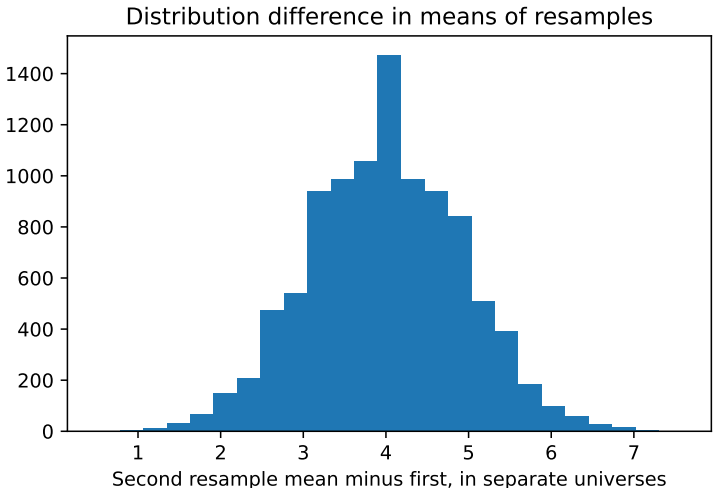


We find that in only of the trials the difference between the two observed groups of 12 pigs was more than 2. Now we investigate how often samples of 12 pigs, drawn from the *separate* universes, will show a mean difference larger than 2 pounds. We do so by making up a deck of 25 or 50 cards for *each* of the 12 hypothesized A's and each of the 12 B's, with the ration name and the weight gain written on it — that is, a deck of, say, 300 cards for each ration. Then from each deck we draw a set of 12 cards at random, record the group averages, and find the difference.

Here is the same work done with more runs on the computer. In this version we are sampling from the separate A and B universes we have estimated.

```
# A new empty array to store the trial results.
results = np.zeros(n_trials)

# Do 10,000 experiments.
for i in range(n_trials):
    # Take a "resample" of 12 with replacement from A and put it in fake_a.
    # Notice we are sampling from "a_weights" this time.
    fake_a = rnd.choice(a_weights, size=12)
    # Likewise to make fake_b
    # Notice we are sampling from "b_weights" this time.
    fake_b = rnd.choice(b_weights, size=12)
    # Mean of the first "resample" sample.
    fake_a_mean = np.mean(fake_a)
    # Mean of the second "resample" sample.
    fake_b_mean = np.mean(fake_b)
    # Calculate the difference between the two resamples.
    fake_diff = fake_a_mean - fake_b_mean
    # Keep track of each trial result.
    results[i] = fake_diff
    # End one experiment, go back and repeat until all trials are complete,
    # then proceed.

# Produce a histogram of trial results.
plt.hist(results, bins=25)
plt.xlabel('Second resample mean minus first, in separate universes')
plt.title('Distribution difference in means of resamples')

# What proportion of the differences are implausible in the combined universe?
k = np.sum(results >= q_95)
kk = k / n_trials

print('p for separate universe results >= 95% quantile:', kk)
```

```
p for separate universe results >= 95% quantile: 0.9865
```



If `kk` is close to one, we know that the sample size is large enough that samples drawn from the universes we have hypothesized will not mislead us into thinking that they could come from the same universe.

Therefore, two samples of twelve pigs each are clearly large enough, and, in fact, even smaller samples might be sufficient if the universes are really like those we guessed at. If, on the other hand, the differences in the guessed universes had been smaller, then twelve-pig groups would have seemed too small and we would then have had to try out larger sample sizes, say forty-eight pigs in each group and perhaps 200 pigs in each group if forty-eight were not enough. And so on until the sample size is large enough to promise the accuracy we want. (In that case, the decks would also have to be much larger, of course.)

If we had guessed different universes for the two rations, then the sample sizes required would have been larger or smaller. If we had guessed the averages for the two samples to be closer together, then we would have needed larger samples. Also, if we had guessed the weight gains *within* each universe to be less spread out, the samples could have been smaller and vice versa.

ℹ End of notebook: Sample size for pig rations with bootstrap

`sample_size_rations` starts at Note 85.

30.4 Step-wise sample-size determination

Often it is wisest to determine the sample size as you go along, rather than fixing it firmly in advance. In sequential sampling, you *continue* sampling until the split is sufficiently even to make you believe you have a reliable answer.

Related techniques work in a series of jumps from sample size to sample size. Step-wise sampling makes it less likely that you will take a sample that is much larger than necessary. For example, in the internet contract survey case, if you took a sample of perhaps fifty you could see whether the split was as wide as 32-18, which you figure you need for 9 to 1 odds that your answer is right. If the split were not that wide, you would sample another fifty, another 100, or however large a sample you needed until you reached a split wide enough to satisfy you that your answer was reliable and that you really knew which way the entire universe would vote.

Step-wise sampling is not always practical, however, and the internet contract survey example is unusually favorable for its use. One major pitfall is that the *early* responses to a mail survey, for example, do *not* provide a random sample of the whole, and therefore it is a mistake simply to look at the early returns when the split is not wide enough to justify a verdict. If you have listened to early news reports of election returns, you know how misleading the reports from the first precincts can be if we regard them as a fair sample of the whole.[2]

Stratified sampling is another device that helps reduce the sample size required, by balancing the amounts of information you obtain in the various strata. (Cluster sampling does not reduce the sample size. Rather, it aims to reduce the cost of obtaining a sample that will produce a given level of accuracy.)

30.5 Summary

Sample sizes are too often determined on the basis of convention or of the available budget. A more rational method of choosing the size of the sample is by balancing the diminution of error expected with a larger sample, and its value, against the cost of increasing the sample

[2]See J. Lorie and H. Roberts (1951, 155–57) for more discussion of the limitations of sequential sampling. Hansen *et al* (1953, 78), warn against the danger of increasing the sample size in this fashion:

> A fairly obvious and flagrant way of arriving at biased results is to examine the returns from an initial sample to determine whether they appear acceptable to the investigator; if they do, he uses the results as they are; if they do not, he discards the sample results and draws a new sample, perhaps by a different method, in the hope that he will obtain a result more nearly like the one he expected. Such an approach can be utilized to obtain almost any results desired, or can "prove" any point even when unbiased or consistent methods of selecting the sample and making the individual estimates are used if the initial results are subject to relatively large sampling errors.

size. The relationship of various sample sizes to various degrees of accuracy can be estimated with resampling methods, which are illustrated here.

31 Bayesian Analysis by Simulation

> This branch of mathematics [probability] is the only one, I believe, in which good writers frequently get results entirely erroneous. (Peirce 1923, Doctrine of Chances, II)

Bayesian analysis is a way of thinking about problems in probability and statistics that can help one reach otherwise-difficult decisions. It also can sometimes be used in science. The range of its recommended uses is controversial, but this chapter deals only with those uses of Bayesian analysis that are uncontroversial.

Better than defining Bayesian analysis in formal terms is to demonstrate its use. We shall start with the simplest sort of problem, and proceed gradually from there.

31.1 Simple decision problems

31.1.1 Assessing the Likelihood That a Used Car Will Be Sound

Consider a problem in estimating the soundness of a used car one considers purchasing (after (Wonnacott and Wonnacott 1990, 93–94)). Seventy percent of the cars are known to be OK on average, and 30 percent are faulty. Of the cars that *are* really OK, a mechanic correctly identifies 80 percent as "OK" but says that 20 percent are "faulty"; of those that are faulty, the mechanic correctly identifies 90 percent as faulty and says (incorrectly) that 10 percent are OK.

We wish to know the probability that if the mechanic *says* a car is "OK," it *really* is faulty. Phrased differently, what is the probability of a car being faulty if the mechanic said it was OK?

We can get the desired probabilities directly by simulation without knowing Bayes' rule, as we shall see. But one must be able to model the physical problem correctly in order to proceed with the simulation; this requirement of a clearly visualized model is a strong point in favor of simulation.

1. Note that we are only interested in outcomes where the mechanic approved a car.
2. For each car, generate a label of either "faulty" or "working" with probabilities of 0.3 and 0.7, respectively.

3. For each *faulty car*, we generate one of two labels, "approved" or "not approved" with probabilities 0.1 and 0.9, respectively.
4. For each *working car*, we generate one of two labels, "approved" or "not approved" with probabilities 0.7 and 0.3, respectively.
5. Out of all cars "approved", count how many are "faulty". The ratio between these numbers is our answer.

In-place operators in Python

In the code that follows, we are going to use a new and neat feature of Python, called *in-place* operations.
We often find ourselves in the situation where we have some variable, say `a`, and we want to do some operation on the value of `a` *and then store the result in* `a` *again*.
Of course, we could do something like this:

```
# The initial value of "a".
a = 10
```

```
# Set a to get the value of a plus 1.
a = a + 1
# Show the result.
a
```

```
11
```

Because this is such a common kind of thing to do, Python has special operators to do this for us, called *in-place* operators. For the example above, there is an *in-place addition* operator `+=` that takes the value of the variable on the left hand side, adds the value on the right hand side, and stores the result in the variable on the left hand side. For example:

```
# The initial value of "b".
b = 10
```

```
# Set b to get the value of b plus 1.
# Notice the in-place operator "+=".
b += 1
# Show the result.
b
```

```
11
```

There are also in-place operators for subtraction (e.g. `c -= 3`), multiplication (e.g. `d *= 2`), and division (`e /= 2`).
You will see us using the in-place `+=` in the code below.

Here is the whole simulation of the car / mechanic problem:

> **Note 86:** Notebook: Bayesian analysis of cars and mechanics
>
> - Download notebook
> - Interact

```
import numpy as np

# Set up the random number generator.
rnd = np.random.default_rng()

n_trials = 10000  # number of cars

# Counters for number of approved, number of approved and faulty
approved = 0
approved_and_faulty = 0

for i in range(n_trials):

    # Decide whether the car is faulty or working, with a probability of
    # 0.3 and 0.7 respectively
    car = rnd.choice(['faulty', 'working'], p=[0.3, 0.7])

    if car == 'faulty':
        # What the mechanic says of a faulty car
        mechanic_says = rnd.choice(['approved', 'not approved'], p=[0.1, 0.9])
    else:
        # What the mechanic says of a working car
        mechanic_says = rnd.choice(['approved', 'not approved'], p=[0.7, 0.3])

    if mechanic_says == 'approved':
        approved += 1

        if car == 'faulty':
            approved_and_faulty += 1

k = approved_and_faulty / approved

print('Proportion of faulty cars of cars approved:', np.round(k, 2))
```

```
Proportion of faulty cars of cars approved: 0.06
```

The answer looks to be somewhere between 5 and 6%. The code clearly follows the description step by step, but it is also quite slow. If we can improve the code, we may be able to do our simulation with more cars, and get a more accurate answer.

Let's use arrays to store the states of all cars in the lot simultaneously:

```
# Number of cars; we made this number larger by a factor of 100
n_trials = 1_000_000

# Generate an array with as many entries as there are cars, each
# being either 'working' or 'faulty'
cars = rnd.choice(['working', 'faulty'], p=[0.7, 0.3], size=n_trials)

# Count how many cars are working
n_working = np.sum(cars == 'working')

# All the rest are faulty
n_faulty = n_trials - n_working

# Create a new array in which to store what a mechanic says
# about the car: 'approved' or 'not approved'. Use
# dtype=object because we need to store strings.
mechanic_says = np.zeros(n_trials, dtype=object)

# We start with the working cars; what does the mechanic say about them?
# Generate 'approved' or 'not approved' labels with the given probabilities.
mechanic_says[cars == 'working'] = rnd.choice(
    ['approved', 'not approved'], p=[0.8, 0.2], size=n_working
)

# Similarly, for each faulty car, generate 'approved'/'not approved'
# labels with the given probabilities.
mechanic_says[cars == 'faulty'] = rnd.choice(
    ['approved', 'not approved'], p=[0.1, 0.9], size=n_faulty
)

# Identify all cars that were approved
# This produces a binary mask, an array that looks like:
# [True, False, False, True, ... ]
approved = (mechanic_says == 'approved')

# Identify cars that are faulty AND were approved
faulty_but_approved = (cars == 'faulty') & approved
```

```
# Count the number of cars that are faulty but approved, as well as
# the total number of cars that were approved
n_faulty_but_approved = np.sum(faulty_but_approved)
n_approved = np.sum(approved)

# Calculate the ratio, which is the answer we seek
k = n_faulty_but_approved / n_approved

print('Proportion of faulty cars of cars approved:', np.round(k, 2))
```

```
Proportion of faulty cars of cars approved: 0.05
```

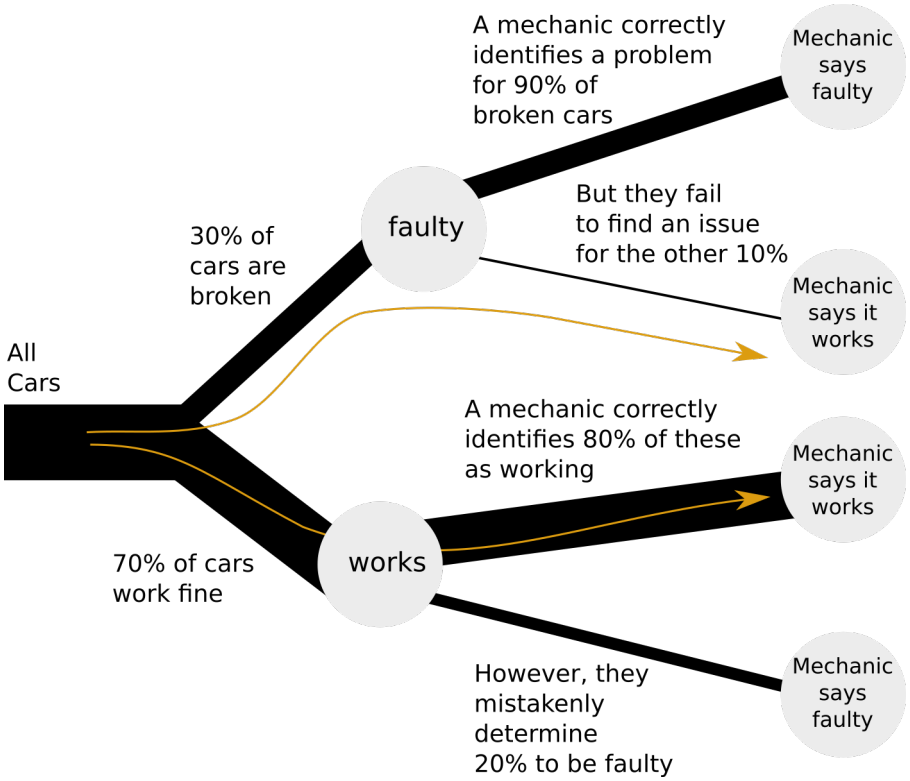
The code now runs much faster, and with a larger number of cars we see that the answer is closer to a 5% chance of a car being broken after it has been approved by a mechanic.

> **i** End of notebook: Bayesian analysis of cars and mechanics
>
> `bayes_cars` starts at Note 86.

31.1.2 Calculation without simulation

Simulation forces us to model our problem clearly and concretely in code. Such code is most often easier to reason about than opaque statistical methods. Running the simulation gives a good sense of what the correct answer should be. Thereafter, we can still look into different — sometimes more elegant or accurate — ways of modeling and solving the problem.

Let's examine the following diagram of our car selection:



We see that there are two paths, highlighted, that results in a car being approved by a mechanic. Either a car can be working, and correctly identified as such by a mechanic; or the car can be broken, while the mechanic mistakenly determines it to be working. Our question only pertains to these two paths, so we do not need to study the rest of the tree.

In the long run, in our simulation, about 70% of the cars will end with the label "working", and about 30% will end up with the label "faulty". We just took 10000 sample cars above but, in fact, the larger the number of cars we take, the closer we will get to 70% "working" and 30% "faulty". So, with many samples, we can think of 70% of these samples flowing down the "working" path, and 30% flowing along the "faulty" path.

Now, we want to know, of all the cars approved by a mechanic, how many are faulty:

$$\frac{\text{cars}_{\text{faulty}}}{\text{cars}_{\text{approved}}}$$

We follow the two highlighted paths in the tree:

1. Of a large sample of cars, 30% are faulty. Of these, 10% are approved by a mechanic. That is, 30% * 10% = 3% of all cars.
2. Of all cars, 70% work. Of these, 80% are approved by a mechanic. That is, 70% * 80% = 56% of all cars.

The percentage of faulty cars, out of approved cars, becomes:

$$3\%/(56\% + 3\%) = 5.08\%$$

Notation-wise, it is a bit easier to calculate these sums using proportions rather than percentages:

1. Faulty cars approved by a mechanic: 0.3 * 0.1 = 0.03
2. Working cars approved by a mechanic: 0.7 * 0.8 = 0.56

Fraction of faulty cars out of approved cars: 0.03 / (0.03 + 0.56) = 0.0508

We see that every time the tree branches, it filters the cars: some go to one branch, the rest to another. In our code, we used the AND (`&`) operator to find the intersection between faulty AND approved cars, i.e., to filter out from all faulty cars only the cars that were ALSO approved.

31.2 Probability interpretation

31.2.1 Probability from proportion

In these examples, we often calculate proportions. In the given simulation:

- How many cars are approved by a mechanic? 59/100.
- How many of those 59 were faulty? 3/59.

We often also count how commonly events occur: "it rained 4 out of the 10 days".

An extension of this idea is to *predict* the probability of an event occurring, based on what we had seen in the past. We can say "out of 100 days, there was some rain on 20 of them; we therefore estimate that the probability of rain occurring is 20/100". Of course, this is not a complex or very accurate weather model; for that, we'd need to take other factors—such as season—into consideration. Overall, the more observations we have, the better our probability estimates become. We discussed this idea previously in "The Law of Large Numbers".

31.2.1.1 Ratios of proportions

At our mechanic's yard, we can ask "how many red cars here are faulty"? To calculate that, we'd first count the number of red cars, then the number of those red cars that are also broken, then calculate the ratio: `red_cars_faulty / red_cars`.

We could just as well have worked in percentages: `percentage_of_red_cars_broken / percentage_of_cars_that_are_red`, since that is `(red_cars_broken / 100) / (red_cars / 100)`—the same ratio calculated before.

Our point is that the denominator doesn't matter when calculating ratios, so we could just as well have written:

(red_cars_broken / all_cars) / (red_cars / all_cars)

or

$$P(\text{cars that are red and that are broken})/P(\text{red cars})$$

31.2.2 Probability relationships: conditional probability

Here's one way of writing the probability that a car is broken:

$$P(\text{car is broken})$$

We can shorten "car is broken" to B, and write the same thing as:

$$P(B)$$

Similarly, we could write the probability that a car is red as:

$$P(R)$$

We might also want to express the *conditional probability*, as in the probability that the car is broken, *given that* we already know that the car is red:

$$P(\text{car is broken GIVEN THAT car is red})$$

That is getting getting pretty verbose, so we will shorten this as we did above:

$$P(B \text{ GIVEN THAT } R)$$

To make things even more compact, we write "GIVEN THAT" as a vertical bar | — so the whole thing becomes:

$$P(B|R)$$

We read this as "the probability that the car is broken given that the car is red". Such a probability is known as a *conditional probability*. We discuss these in more detail in Section 8.13.

In our original problem, we ask what the chance is of a car being broken given that a mechanic approved it. As discussed under "Ratios of proportions", it can be calculated with:

$$P(\text{car broken} \mid \text{mechanic approved}) =$$
$$P(\text{car broken and mechanic approved})/P(\text{mechanic approved})$$

We have already used B to mean "broken" (above), so let us use A to mean "mechanic approved". Then we can write the statement above in a more compact way:

$$P(B|A) = P(B \text{ and } A)/P(A)$$

To put this generally, conditional probabilities for two events X and Y can be written as:

$P(X|Y) = P(X \text{ and } Y)/P(Y)$

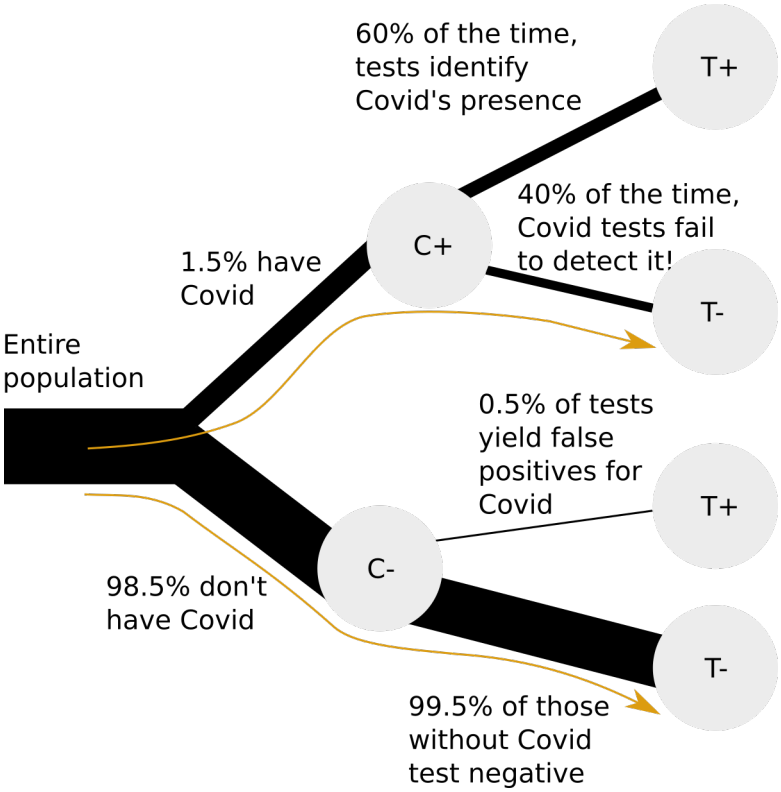
Where (again) and means that *both* events occur.

31.2.3 Example: conditional probability

Let's discuss a very relevant example. You get a Covid test, and the test is negative. Now, you would like to know what the chance is of you having Covid.

We have the following information:

- 1.5% of people in your area have Covid
- The false positive rate of the tests (i.e., that they detect Covid when it is absent) is very low at 0.5%
- The false negative rate (i.e., that they fail to detect Covid when it is present) is quite high at 40%



Again, we start with our simulation.

> Note 87: Notebook: Bayesian analysis of Covid test result
>
> - Download notebook
> - Interact

```
import numpy as np

rnd = np.random.default_rng()

# The number of people.
n_trials = 1_000_000

# For each person, generate a True or False label,
# indicating that they have / don't have Covid.
person_has_covid = rnd.choice(
    [True, False], p=[0.015, 0.985],
    size=n_trials
)
```

```
# Calculate the numbers of people with and without Covid.
n_with_covid = np.sum(person_has_covid)
n_without_covid = n_trials - n_with_covid

# In this array, we will store, for each person, whether they
# had a positive or a negative test.
test_result = np.zeros(n_trials, dtype=bool)

# Draw test results for people with Covid.
test_result[person_has_covid] = rnd.choice(
    [True, False], p=[0.6, 0.4],
    size=n_with_covid
)

# Draw test results for people without Covid.
# ~person_has_covid` flips all Boolean values from FALSE to TRUE
# and from TRUE to FALSE.
test_result[~person_has_covid] = rnd.choice(
    [True, False], p=[0.005, 0.995],
    size=n_without_covid
)

# Get the Covid statuses of all those with negative tests
# (`test_result` is a Boolean mask, like `[True, False, False, True, ...]`,
# and `~test_result` flips all Boolean values to `[False, True, True, False, ...]`.
covid_status_negative_test = person_has_covid[~test_result]

# Now, count how many with Covid had a negative test results.
n_with_covid_and_negative_test = np.sum(covid_status_negative_test)

# And how many people, overall, had negative test results.
n_with_negative_test = len(covid_status_negative_test)

k = n_with_covid_and_negative_test / n_with_negative_test

print('Proportion with Covid of those with negative test:', np.round(k, 4))
```

```
Proportion with Covid of those with negative test: 0.0061
```

This gives around 0.006 or 0.6%.

End of notebook: Bayesian analysis of Covid test result

`bayes_covid` starts at Note 87.

Now that we have a rough indication of what the answer should be, let's try and calculate it directly, based on the tree of information shown earlier.

We will use these abbreviations:

- C^+ means Covid positive (you do actually have Covid).
- C^- means Covid negative (you do *not* actually have Covid).
- T^+ means the Covid *test* was positive.
- T^- means the Covid *test* was negative.

For example $P(C^+|T^-)$ is the probability (P) that you do actually have Covid (C^+) *given that* ($|$) the test was negative (T^-).

We would like to know the probability of having Covid *given that* your test was negative ($P(C^+|T^-)$). Using the conditional probability relationship from above, we can write:

$$P(C^+|T^-) = P(C^+ \text{ and } T^-)/P(T^-)$$

We see from the tree diagram that $P(C^+ \text{ and } T^-) = P(T^-|C^+) * P(C^+) = .4 * .015 = 0.006$.

We observe that $P(T^-) = P(T^- \text{ and } C^-) + P(T^- \text{ and } C^+)$, i.e. that we can obtain a negative test result through two paths, having Covid or not having Covid. We expand these further as conditional probabilities:

$P(T^- \text{ and } C^-) = P(T^-|C^-) * P(C^-)$

and

$P(T^- \text{ and } C^+) = P(T^-|C^+) * P(C^+)$.

We can now calculate

$$P(T^-) = P(T^-|C^-) * P(C^-) + P(T^-|C^+) * P(C^+)$$

$$= .995 * .985 + .4 * .015 = 0.986$$

The answer, then, is:

$P(C^+|T^-) = 0.006/0.986 = 0.0061$ or 0.61%.

This matches very closely our simulation result, so we have some confidence that we have done the calculation correctly.

31.2.4 Estimating Driving Risk for Insurance Purposes

Another sort of introductory problem, following after (Feller 1968, p 122):

> **Note 88: Notebook: Bayesian analysis for insurance premium**
>
> - Download notebook
> - Interact

A mutual insurance company charges its members according to the risk of having an car accident. It is known that there are two classes of people — 80 percent of the population with good driving judgment and with a probability of .06 of having an accident each year, and 20 percent with poor judgment and a probability of .6 of having an accident each year. The company's policy is to charge \$100 for each percent of risk, i. e., a driver with a probability of .6 should pay 60*\$100 = \$6000.

If nothing is known of a driver except that they had an accident last year, what fee should they pay?

Another way to phrase this question is: given that a driver had an accident last year, what is the probability of them having an accident overall?

We will proceed as follows:

1. Generate a population of N people. Label each as `good driver` or `poor driver`.
2. Simulate the last year for each person: did they have an accident or not?
3. Select only the ones that had an accident last year.
4. Among those, calculate what their average risk is of making an accident. This will indicate the appropriate insurance premium.

```
import numpy as np

rnd = np.random.default_rng()

n_trials = 100_000
cost_per_percent = 100

people = rnd.choice(
    ['good driver', 'poor driver'], p=[0.8, 0.2],
    size=n_trials
)

good_driver = (people == 'good driver')
poor_driver = ~good_driver
```

```
# Did they have an accident last year?
had_accident = np.zeros(n_trials, dtype=bool)
had_accident[good_driver] = rnd.choice(
    [True, False], p=[0.06, 0.94],
    size=np.sum(good_driver)
)
had_accident[poor_driver] = rnd.choice(
    [True, False], p=[0.6, 0.4],
    size=np.sum(poor_driver)
)

ppl_with_accidents = people[had_accident]
n_good_driver_accidents = np.sum(ppl_with_accidents == 'good driver')
n_poor_driver_accidents = np.sum(ppl_with_accidents == 'poor driver')
n_all_with_accidents = n_good_driver_accidents + n_poor_driver_accidents

avg_risk_percent = ((n_good_driver_accidents * 0.06 +
                     n_poor_driver_accidents * 0.6) /
                    n_all_with_accidents * 100)

premium = avg_risk_percent * cost_per_percent

print('Premium is:', np.round(premium))
```

```
Premium is: 4462.0
```

The answer should be around 4450 USD.

> ℹ End of notebook: Bayesian analysis for insurance premium
>
> `bayes_accidents` starts at Note 88.

31.2.5 Screening for Disease

This is a classic Bayesian problem (quoted by Tversky and Kahneman (1982, 154), from Cascells *et al.* (1978, 999)):

> If a test to detect a disease whose prevalence is 1/1000 has a false positive rate of 5%, what is the chance that a person found to have a positive result actually has the disease, assuming you know nothing about the person's symptoms or signs?

Tversky and Kahneman note that among the respondents — students and staff at Harvard Medical School — "the most common response, given by almost half of the participants, was 95%" — very much the wrong answer.

To obtain an answer by simulation, we may rephrase the question above with (hypothetical) absolute numbers as follows:

If a test to detect a disease whose prevalence has been estimated to be about 100,000 in the population of 100 million persons over age 40 (that is, about 1 in a thousand) has been observed to have a false positive rate of 60 in 1200 observations, and never gives a negative result if a person really has the disease, what is the chance that a person found to have a positive result actually has the disease, assuming you know nothing about the person's symptoms or signs?

If the raw numbers are not available, the problem can be phrased in such terms as "about 1 case in 1000" and "about 5 false positives in 100 cases."

One may obtain an answer as follows:

1. Construct bucket A with 999 white beads and 1 black bead, and bucket B with 95 green beads and 5 red beads. A more complete problem that also discusses false negatives would need a third bucket.

2. Pick a bead from bucket A. If black, record "T," replace the bead, and end the trial. If white, continue to step 3.

3. If a white bead is drawn from bucket A, select a bead from bucket B. If red, record "F" and replace the bead, and if green record "N" and replace the bead.

4. Repeat steps 2-4 perhaps 10,000 times, and in the results count the proportion of "T"s to ("T"s plus "F"s) ignoring the "N"s).

   Of course 10,000 draws would be tedious, but even after a few hundred draws a person would be likely to draw the correct conclusion that the proportion of "T"s to ("T"s plus "F"s) would be small. And it is easy with a computer to do 10,000 trials very quickly.

   Note that the respondents in the Cascells *et al.* study were not naive; the medical staff members were supposed to understand statistics. Yet most doctors and other personnel offered wrong answers. If simulation can do better than the standard deductive method, then simulation would seem to be the method of choice. And only one piece of training for simulation is required: Teach the habit of saying "I'll simulate it" and then actually doing so.

31.3 Fundamental problems in statistical practice

Box and Tiao (1992) begin their classic exposition of Bayesian statistics with the analysis of a famous problem first published by Fisher (1959, 18).

> …there are mice of two colors, black and brown. The black mice are of two genetic kinds, homozygotes (*BB*) and heterozygotes (*Bb*), and the brown mice are of one kind (*bb*). It is known from established genetic theory that the probabilities associated with offspring from various matings are as listed in Table 31.1.

(See (Box and Tiao 1992, 12–14)).

Table 31.1: Probabilities for Genetic Character of Mice Offspring

	BB (black)	Bb (black)	bb (brown)
BB mated with bb	0	1	0
Bb mated with bb	0	½	½
Bb mated with Bb	¼	½	¼

Suppose we have a "test" mouse which has been produced by a mating between two (*Bb*) mice and is black. What is the genetic kind of this mouse?

To answer that, we look at the information in the last line of the table: it shows that the probabilities of a test mouse is of kind *BB* and *Bb* are precisely known, and are 1/3 and 2/3 respectively ((1/4)/(1/4 + 1/2) vs (1/2)/(1/4 + 1/2)). We call this our "prior" estimate — in other words, our estimate before seeing data.

Suppose the test mouse is now mated with a brown mouse (of kind *bb*) and produces seven black offspring. Before, we thought that it was more likely for the parent to be of kind *Bb* than of kind *BB*. But if that were true, then we would have expected to have seen some brown offspring (the probability of mating *Bb* with *bb* resulting in brown offspring is given as 0.5). Therefore, we sense that it may now be more likely that the parent was of type *BB* instead. How do we quantify that?

One can calculate, as Fisher (1959, 19) did, the probabilities after seeing the data (we call this the *posterior* probability). This is typically done using using Bayes' rule.

But instead of doing that, let's take the easy route out and simulate the situation instead.

1. We begin, as do Box and Tiao, by restricting our attention to the third line in Table 31.1. We draw a mouse with label 'BB', 'Bb', or 'bb', using those probabilities. We were told that the "test mouse" is black, so if we draw 'bb', we try again. (Alternatively, we could draw 'BB' and 'Bb' with probabilities of 1/3 and 2/3 respectively.)

2. We now want to examine the offspring of the test mouse when mated with a brown "bb" mouse. Specifically, we are only interested in cases where all offspring were black. We will store the genetic kind of the parents of such offspring so that we can count them later.

   If our test mouse is "BB", we already know that all their offspring will be black ("Bb"). Thus, store "BB" in the parent list.

3. If our test mouse is "Bb", we have a bit more work to do. Draw seven offspring from the middle row of Table 31.1. If all the offspring are black, store "Bb" in the parent list.

4. Repeat steps 1-3 perhaps 10000 times.

5. Now, out of all parents count the numbers of "BB" vs "Bb".

We will do a naïve implementation that closely follows the logic described above, followed by a slightly optimized version.

> **Note 89: Notebook: A problem of black and brown mice**
>
> - Download notebook
> - Interact

```
import numpy as np

rnd = np.random.default_rng()

n_trials = 100_000

# Make an array to store results for each trial.
# The results are strings, so use dtype=object.
# Many of these we will not set, for example, for brown mice (see below).
parents = np.zeros(n_trials, dtype=object)

for i in range(n_trials):
    test_mouse = rnd.choice(['BB', 'Bb', 'bb'], p=[0.25, 0.5, 0.25])

    # The test mouse is black; since we drew a brown mouse skip this trial
    if test_mouse == 'bb':
        # continue has the effect of aborting this iteration of the loop
        # and going back to start the next iteration.  If the code gets
        # to "continue", none of the rest of the loop (indented) code
        # will run.
        continue
```

```
    # If the test mouse is 'BB', all 7 children are guaranteed to
    # be 'Bb' black.
    # Therefore, add 'BB' to the parent list.
    if test_mouse == 'BB':
        parents[i] = 'BB'

    # If the parent mouse is 'Bb', we draw 7 children to
    # see whether all of them are black ('Bb').
    # The probabilities come from the middle row of the table.
    if test_mouse == 'Bb':
        children = rnd.choice(['Bb', 'bb'], p=[0.5, 0.5], size=7)
        if np.all(children == 'Bb'):
            parents[i] = 'Bb'

# Now, count how many parents were 'BB' vs 'Bb'
n_parents_BB = np.sum(parents == 'BB')
n_parents_Bb = np.sum(parents == 'Bb')
n_B = n_parents_BB + n_parents_Bb

p_BB = n_parents_BB / n_B
p_Bb = n_parents_Bb / n_B

print('p_BB =', np.round(p_BB, 3))
```

```
p_BB = 0.985
```

```
print('p_Bb =', np.round(p_Bb, 3))
```

```
p_Bb = 0.015
```

```
print('Ratio =', np.round(p_BB / p_Bb, 1))
```

```
Ratio = 66.0
```

We see that all the offspring being black considerably changes the situation! We started with the odds being 2:1 in favor of Bb vs BB. The "posterior" or "after the evidence" ratio is closer to 64:1 in favor of *BB*! (1973, pp. 12-14)

Let's tune the code a bit to run faster. Instead of doing the trials one mouse at a time, we will do the whole bunch together.

To do this, we will use two-dimensional arrays.

So far, nearly all the arrays we have used are one-dimensional. A one-dimensional array is a sequence of values. Let us generate a one-dimensional array with `rnd.choice`, as we have many times in this book, and in this chapter.

```
# A one-dimensional array, with five elements.
one_d = rnd.choice([1, 2], size=5)
one_d
```

```
array([2, 2, 1, 1, 1])
```

However, we can also generate arrays with more than one dimension. In particular we can generate arrays with two dimensions. An array with two dimensions has rows and columns, much like a Pandas data frame. However, unlike data frames, two-dimensional arrays have no row or column names. Here is a two-dimensional array we create with `rnd.choice`, by passing two values to the size argument:

```
# A two-dimensional array with five rows and three columns.
two_d = rnd.choice([1, 2], size=(5, 3))
two_d
```

```
array([[1, 1, 2],
       [1, 2, 1],
       [1, 2, 2],
       [1, 1, 1],
       [2, 2, 2]])
```

As usual, we can apply Boolean comparison operations to this array, to get a two-dimensional Boolean array:

```
is_2 = two_d == 2
is_2
```

```
array([[False, False,  True],
       [False,  True, False],
       [False,  True,  True],
       [False, False, False],
       [ True,  True,  True]])
```

Numpy thinks of two-dimensional arrays as having two *axes*, where the first axis (axis at position 0) is the row axis, and the second axis (at position 1) is the column axis.

Many Numpy functions have an `axis` argument that asks the function to apply its operation along a particular axis. For example, we might want to ask whether `all` the values in *each column* (across axis position 1) are equal to 2. We can do this using `np.all`:

```
all_equal_2 = np.all(is_2, axis=1)
all_equal_2
```

```
array([False, False, False, False,  True])
```

Notice that we get one answer for each row (axis=0), where the answer is `np.all` across the columns, for that row.

```
n_trials = 1_000_000

# In n_trials trials, pair two Bb mice and generate a child.
test_mice = rnd.choice(['BB', 'Bb', 'bb'], p=[0.25, 0.5, 0.25], size=n_trials)

# The resulting test mouse is black, so filter out all brown ones.
test_mice = test_mice[test_mice != 'bb']
n_test_mice = len(test_mice)

# Each test mouse will now be mated with a brown mouse, producing 7 offspring.
# We then store whether all the offspring were black or not.
all_offspring_black = np.zeros(n_test_mice, dtype=bool)

# If a test mouse is 'BB', we are assured that all its offspring
# will be black.
all_offspring_black[test_mice == 'BB'] = True

# If a test mouse is 'Bb', we have to generate its offspring and
# see whether they are all black or not.
test_mice_Bb = (test_mice == 'Bb')
n_test_mice_Bb = np.sum(test_mice_Bb)

# Generate all offspring of all 'Bb' test mice.
# This gives a 2-dimensional array, with one row per Bb mouse,
# and 7 columns, one for each child.
offspring = rnd.choice(
    ['Bb', 'bb'], p=[0.5, 0.5], size=(n_test_mice_Bb, 7)
```

```
)
# Check whether all children (columns) are Bb, for each row.
all_offspring_black[test_mice_Bb] = np.all(offspring == 'Bb', axis=1)

# Find the genetic types of the parents of all-black offspring.
parents = test_mice[all_offspring_black]

# Calculate what fraction of parents were 'BB' vs 'Bb'.
parents_BB = (parents == 'BB')
parents_Bb = (parents == 'Bb')
n_B = np.sum(all_offspring_black)

p_BB = np.sum(parents_BB) / n_B
p_Bb = np.sum(parents_Bb) / n_B

print('p_BB = ', np.round(p_BB, 3))
```

```
p_BB =  0.984
```

```
print('p_Bb = ', np.round(p_Bb, 3))
```

```
p_Bb =  0.016
```

```
print('Ratio = ', np.round(p_BB / p_Bb, 1))
```

```
Ratio =  63.4
```

This yields a similar result, but in much shorter time — which means we can increase the number of trials and get a more accurate result.

> **i** End of notebook: A problem of black and brown mice
>
> `box_tao_mice` starts at Note 89.

Creating the correct simulation procedure is not trivial, because Bayesian reasoning is subtle — a reason it has been the cause of controversy for more than two centuries. But it certainly is not easier to create a correct procedure using analytic tools (except in the cookbook sense of plug-and-pray). And the difficult mathematics that underlie the analytic method (see e.g. (Box and Tiao 1992, Appendix A1.1)) make it almost impossible for the statistician to fully

understand the procedure from beginning to end. If one is interested in insight, the simulation procedure might well be preferred.[1]

31.4 Problems based on normal and other distributions

This section should be skipped by all except advanced practitioners of statistics.

Much of the work in Bayesian analysis for scientific purposes treats the combining of prior distributions having Normal and other standard shapes with sample evidence which may also be represented with such standard functions. The mathematics involved often is formidable, though some of the calculational formulas are fairly simple and even intuitive.

These problems may be handled with simulation by replacing the Normal (or other) distribution with the original raw data when data are available, or by a set of discrete sub-universes when distributions are subjective.

Measured data from a continuous distribution present a special problem because the probability of any one observed value is very low, often approaching zero, and hence the probability of a given set of observed values usually cannot be estimated sensibly; this is the reason for the conventional practice of working with a continuous distribution itself, of course. But a simulation necessarily works with discrete values. A feasible procedure must bridge this gulf.

The logic for a problem of Schlaifer's (1961, example 17.1) will only be sketched out. The procedure is rather novel, but it has not heretofore been published and therefore must be considered tentative and requiring particular scrutiny.

31.4.1 An Intermediate Problem in Conditional Probability

Schlaifer employs a quality-control problem for his leading example of Bayesian estimation with Normal sampling. A chemical manufacturer wants to estimate the amount of yield of a crucial ingredient X in a batch of raw material in order to decide whether it should receive special handling. The yield ranges between 2 and 3 pounds (per gallon), and the manufacturer has compiled the distribution of the last 100 batches.

[1] We can use a similar procedure to illustrate an aspect of the Bayesian procedure that Box and Tiao emphasize, its sequentially-consistent character. First let us carry out the above procedure but observe only three black balls in a row. The program to be used is the same except for the insertion of "3" for "7" where "7" appears. We then estimate the probability for BB, which turns out to be about 1/5 instead of about 1/65. We then substitute for bucket A a bucket A' with appropriate numbers of black Bb's and black BB's, to represent the "updated" prior probability. We may then continue by substituting "4" for "3" above (to attain a total of seven observed black balls), and find that the probability is about what it was when we observed 7 black balls in a single sample (1/65). This shows that the Bayesian procedure accumulates information without "leakage" and with consistency.

The manufacturer currently uses the decision rule that if the mean of nine samples from the batch (which vary only because of measurement error, which is the reason that he takes nine samples rather than just one) indicates that the batch mean is greater than 2.5 gallons, the batch is accepted. The first question Schlaifer asks, as a sampling-theory waystation to the more general question, is the likelihood that a given batch with any given yield — say 2.3 gallons — will produce a set of samples with a mean as great or greater than 2.5 gallons.

We are told that the manufacturer has in hand nine samples from a given batch; they are 1.84, 1.75, 1.39, 1.65, 3.53, 1.03, 2.73, 2.86, and 1.96, with a mean of 2.08. Because we are also told that the manufacturer considers the extent of sample variation to be the same at all yield levels, we may — if we are again working with 2.3 as our example of a possible universe — therefore add (2.3 minus 2.08 =) 0.22 to each of these nine observations, so as to constitute a bootstrap-type universe; we do this on the grounds that this is our best guess about the constitution of that distribution with a mean at (say) 2.3.

We then repeatedly draw samples of nine observations from this distribution (centered at 2.3) to see how frequently its mean exceeds 2.5. This work is so straightforward that we need not even state the steps in the procedure.

31.4.2 Estimating the Posterior Distribution

Next we estimate the posterior distribution. Figure 31.1 shows the prior distribution of batch yields, based on 100 previous batches.

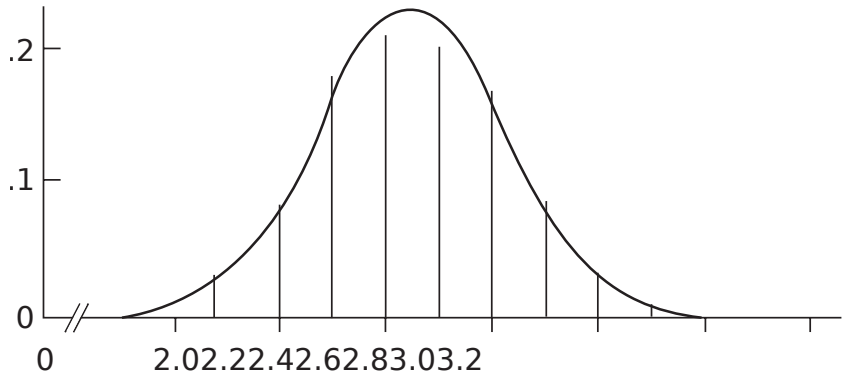


Figure 31.1: Posterior distribution of batch yields

Notation: S_m = set of batches (where total S = 100) with a particular mean m (say, m = 2.1). x_i = particular observation (say, $x_3 = 1.03$). s = the set of x_i .

We now perform for each of the S_m (categorized into the tenth-of-gallon divisions between 2.1 and 3.0 gallons), each corresponding to one of the yields ranging from 2.1 to 3.0, the same sort of sampling operation performed for $S_{m=2.3}$ in the previous problem. But now, instead of using the manufacturer's decision criterion of 2.5, we construct an interval of arbitrary width

around the sample mean of 2.08 — say at .1 intervals from 2.03 to 2.13 — and then work with the weighted proportions of sample means that fall into this interval.

1. Using a bootstrap-like approach, we presume that the sub-universe of observations related to each S_m equals the mean of that S_m — say, 2.1) plus (minus) the mean of the x_i (equals 2.05) added to (subtracted from) each of the nine x_i, say, 1.03 + .05 = 1.08. For a distribution centered at 2.3, the values would be (1.84 + .22 = 2.06, 1.75 + .22 = 1.97…).
2. Working with the distribution centered at 2.3 as an example: Constitute a universe of the values (1.84+.22=2.06, 1.75 + .22 = 1.97…). Here we may notice that the variability in the sample enters into the analysis at this point, rather than when the sample evidence is combined with the prior distribution; this is in contrast to conventional Bayesian practice where the posterior is the result of the prior and sample means weighted by the reciprocals of the variances (see e.g. (Box and Tiao 1992, 17 and Appendix A1.1)).
3. Draw nine observations from this universe (with replacement, of course), compute the mean, and record.
4. Repeat step 2 perhaps 1000 times and plot the distribution of outcomes.
5. Compute the percentages of the means within (say) .5 on each side of the sample mean, i. e. from 2.03–2.13. The resulting number — call it UP^i — is the un-standardized (un-normalized) effect of this sub-distribution in the posterior distribution.
6. Repeat steps 1-5 to cover each other possible batch yield from 2.0 to 3.0 (2.3 was just done).
7. Weight each of these sub-distributions — actually, its UP^i — by its prior probability, and call that WP^i -.
8. Standardize the WP^i s to a total probability of 1.0. The result is the posterior distribution. The value found is 2.283, which the reader may wish to compare with a theoretically-obtained result (which Schlaifer does not give).

This procedure must be biased because the numbers of "hits" will differ between the two sides of the mean for all sub-distributions except that one centered at the same point as the sample, but the extent and properties of this bias are as-yet unknown. The bias would seem to be smaller as the interval is smaller, but a small interval requires a large number of simulations; a satisfactorily narrow interval surely will contain relatively few trials, which is a practical problem of still-unknown dimensions.

Another procedure — less theoretically justified and probably more biased — intended to get around the problem of the narrowness of the interval, is as follows:

5. (**5a.**) Compute the percentages of the means on each side of the sample mean, and note the smaller of the two (or in another possible process, the difference of the two). The resulting number — call it UP^i — is the un-standardized (un-normalized) weight of this sub-distribution in the posterior distribution.

Another possible criterion — a variation on the procedure in 5a — is the *difference* between the two tails; for a universe with the same mean as the sample, this difference would be zero.

31.5 Conclusion

All but the simplest problems in conditional probability are confusing to the intuition even if not difficult mathematically. But when one tackles Bayesian and other problems in probability with experimental simulation methods rather than with logic, neither simple nor complex problems need be difficult for experts or beginners.

This chapter shows how simulation can be a helpful and illuminating way to approach problems in Bayesian analysis.

Simulation has two valuable properties for Bayesian analysis:

1. It can provide an effective way to handle problems whose analytic solution may be difficult or impossible.
2. Simulation can provide insight to problems that otherwise are difficult to understand fully, as is peculiarly the case with Bayesian analysis.

Bayesian problems of updating estimates can be handled easily and straightforwardly with simulation, whether the data are discrete or continuous. The process and the results tend to be intuitive and transparent. Simulation works best with the original raw data rather than with abstractions from them via percentages and distributions. This can aid the understanding as well as facilitate computation.

References

Ani Adhikari, John DeNero, and David Wagner. 2021. *Computational and Inferential Thinking: The Foundations of Data Science*. https://inferentialthinking.com. https://inferentialthinking.com.

Arbuthnot, John. 1710. "An Argument for Divine Providence, Taken from the Constant Regularity Observ'd in the Births of Both Sexes. By Dr. John Arbuthnott, Physitian in Ordinary to Her Majesty, and Fellow of the College of Physitians and the Royal Society." *Philosophical Transactions of the Royal Society of London* 27 (328): 186–90. https://royalsocietypublishing.org/doi/pdf/10.1098/rstl.1710.0011.

Barnett, Vic. 1982. *Comparative Statistical Inference*. 2nd ed. Wiley Series in Probability and Mathematical Statistics. Chichester: John Wiley & Sons. https://archive.org/details/comparativestati0000barn.

Box, George E. P., and George C. Tiao. 1992. *Bayesian Inference in Statistical Analysis*. New York: Wiley & Sons, Inc. https://www.google.co.uk/books/edition/Bayesian_Inference_in_Statistical_Analys/T8Askeyk1k4C.

Brooks, Charles Ernest Pelham. 1928. "Periodicities in the Nile Floods." *Memoirs of the Royal Meteorological Society* 2 (12): 9--26. https://www.rmets.org/sites/default/files/papers/brooksmem2-12.pdf.

Bulmer, M. G. 1979. *Principles of Statistics*. New York, NY: Dover Publications, inc. https://archive.org/details/principlesofstat0000bulm.

Burnett, Ed. 1988. *The Complete Direct Mail List Handbook: Everything You Need to Know about Lists and How to Use Them for Greater Profit*. Englewood Cliffs, New Jersey: Prentice Hall. https://archive.org/details/completedirectma00burn.

Cascells, Ward, Arno Schoenberger, and Thomas B. Grayboys. 1978. "Interpretation by Physicians of Clinical Laboratory Results." *New England Journal of Medicine* 299: 999–1001. https://www.nejm.org/doi/full/10.1056/NEJM197811022991808.

Catling, HW, and RE Jones. 1977. "A Reinvestigation of the Provenance of the Inscribed Stirrup Jars Found at Thebes." *Archaeometry* 19 (2): 137–46.

Chung, James H, and Donald AS Fraser. 1958. "Randomization Tests for a Multivariate Two-Sample Problem." *Journal of the American Statistical Association* 53 (283): 729–35. https://www.jstor.org/stable/pdf/2282050.pdf.

Cipolla, C. M. 1981. *Fighting the Plague in Seventeenth-Century Italy*. Merle Curti Lectures. Madison, Wisconsin: University of Wisconsin Press. https://books.google.co.uk/books?id=Ct/_OJYgnKCsC.

Cobb, George W. 2007. "The Introductory Statistics Course: A Ptolemaic Curriculum?" *Technology Innovations in Statistics Education* 1 (1). https://escholarship.org/uc/item/6

hb3k0nz.

Coleman, William. 1987. "Experimental Physiology and Statistical Inference: The Therapeutic Trial in Nineteenth Century Germany." In *The Probabilistic Revolution: Volume 2: Ideas in the Sciences*, edited by Lorenz Krüger, Gerd Gigerenzer, and Mary S. Morgan. An MIT Press Classic. MIT Press. https://books.google.co.uk/books?id=SLftmgEACAAJ.

Cook, Earl. 1976. "Limits to Exploitation of Nonrenewable Resources." *Science* 191 (4228): 677–82. https://www.jstor.org/stable/pdf/1741483.pdf.

Davenport, Thomas H, and DJ Patil. 2012. "Data Scientist: The Sexiest Job of the 21st Century." *Harvard Business Review* 90 (10): 70–76. https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century.

Deshpande, Jayant V, AP Gore, and A Shanubhogue. 1995. *Statistical Analysis of Nonnormal Data*. Taylor & Francis. https://www.google.co.uk/books/edition/Statistical_Analysis_of_Nonnormal_Data/sS0on2XqwwoC.

Dixon, Wilfrid J, and Frank J Massey Jr. 1983. "Introduction to Statistical Analysis."

Donoho, David. 2017. "50 Years of Data Science." *Journal of Computational and Graphical Statistics* 26 (4): 745–66. http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf.

Dunleavy, Kieron, Stefania Pittaluga, John Janik, Nicole Grant, Margaret Shovlin, Richard Little, Robert Yarchoan, Seth Steinberg, Elaine S. Jaffe, and Wyndham H. Wilson. 2006. "Novel Treatment of Burkitt Lymphoma with Dose-Adjusted EPOCH-Rituximab: Preliminary Results Showing Excellent Outcome." *Blood* 108 (11): 2736–36. https://doi.org/10.1182/blood.V108.11.2736.2736.

Dwass, Meyer. 1957. "Modified Randomization Tests for Nonparametric Hypotheses." *The Annals of Mathematical Statistics*, 181–87. https://www.jstor.org/stable/pdf/2237031.pdf.

Efron, Bradley. 1979. "Bootstrap Methods; Another Look at the Jackknife." *The Annals of Statistics* 7 (1): 1–26. http://www.econ.uiuc.edu/~econ508/Papers/efron79.pdf.

Efron, Bradley, and Robert J Tibshirani. 1993. "An Introduction to the Bootstrap." In *Monographs on Statistics and Applied Probability*, edited by David R Cox, David V Hinkley, Nancy Reid, Donald B Rubin, and Bernard W Silverman. Vol. 57. New York: Chapman & Hall.

Feller, William. 1968. *An Introduction to Probability Theory and Its Applications: Volume i*. 3rd ed. Vol. 1. New York: John Wiley & Sons. https://www.google.co.uk/books/edition/An_Introduction_to_Probability_Theory_an/jbkdAQAAMAAJ.

Feynman, Richard P., and Ralph Leighton. 1988. *What Do You Care What Other People Think? Further Adventures of a Curious Character.* New York, NY: W. W. Norton; Company, Inc. https://archive.org/details/whatdoyoucarewha0000feyn_x5w7.

Fisher, Ronald Aylmer. 1935. *The Design of Experiments*. 1st ed. Edinburgh: Oliver and Boyd Ltd. https://archive.org/details/in.ernet.dli.2015.502684.

———. 1959. "Statistical Methods and Scientific Inference." https://archive.org/details/statisticalmetho0000fish.

———. 1960. *The Design of Experiments*. 7th ed. Edinburgh: Oliver and Boyd Ltd. https://archive.org/details/designofexperime0000rona_q7u5.

Fussler, Herman Howe, and Julian Lincoln Simon. 1961. *Patterns in the Use of Books in Large*

Research Libraries. Chicago: University of Chicago Library.

Gardner, Martin. 1985. *Mathematical Magic Show*. Penguin Books Ltd, Harmondsworth.

———. 2001. *The Colossal Book of Mathematics*. W.W. Norton & Company Inc., New York. https://archive.org/details/B-001-001-265.

Gilovich, Thomas, Robert Vallone, and Amos Tversky. 1985. "The Hot Hand in Basketball: On the Misperception of Random Sequences." *Cognitive Psychology* 17 (3): 295–314. https://www.joelvelasco.net/teaching/122/Gilo.Vallone.Tversky.pdf.

Gnedenko, Boris Vladimirovich, I Aleksandr, and Akovlevich Khinchin. 1962. *An Elementary Introduction to the Theory of Probability*. New York, NY, USA: Dover Publications, Inc. https://archive.org/details/gnedenko-khinchin-an-elementary-introduction-to-the-theory-of-probability.

Goldberg, Samuel. 1986. *Probability: An Introduction*. Courier Corporation. https://www.google.co.uk/books/edition/Probability/CmzFx9rB_FcC.

Graunt, John. 1759. "Natural and Political Observations Mentioned in a Following Index and Made Upon the Bills of Mortality." In *Collection of Yearly Bills of Mortality, from 1657 to 1758 Inclusive*, edited by Thomas Birch. London: A. Miller. https://archive.org/details/collectionyearl00hebegoog.

Hald, Anders. 1990. *A History of Probability and Statistics and Their Applications Before 1750*. New York: John Wiley & Sons. https://archive.org/details/historyofprobabi0000hald.

Hamilton, Max. 1960. "A Rating Scale for Depression." *Journal of Neurology, Neurosurgery, and Psychiatry* 23 (1): 56. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC495331/pdf/jnnpsyc00273-0060.pdf.

Hansen, Morris H, William N Hurwitz, and William G Madow. 1953. "Sample Survey Methods and Theory. Vol. I. Methods and Applications." https://archive.org/details/SampleSurveyMethodsAndTheoryVol1.

Hodges Jr, Joseph Lawson, and Erich Leo Lehmann. 1970. *Basic Concepts of Probability and Statistics*. 2nd ed. San Francisco, California: Holden-Day, Inc. https://archive.org/details/basicconceptsofp0000unse_m8m9.

Hollander, Myles, Douglas A Wolfe, and Eric Chicken. 2013. *Nonparametric Statistical Methods*. 3rd ed. Wiley Series in Probability and Statistics: Applied Probability and Statistics. New York: John Wiley & Sons, Inc. https://books.google.com/books?id=Y5s3AgAAQBAJ.

Hyndman, Rob J, and Yanan Fan. 1996. "Sample Quantiles in Statistical Packages." *The American Statistician* 50 (4): 361–65. https://www.jstor.org/stable/pdf/2684934.pdf.

Kahn, Harold A, and Christopher T Sempos. 1989. *Statistical Methods in Epidemiology*. Vol. 12. Monographs in Epidemiology and Biostatistics. New York: Oxford University Press. https://www.google.co.uk/books/edition/Statistical_Methods_in_Epidemiology/YERYAgAAQBAJ.

Kinsey, Alfred C, Wardell B Pomeroy, and Clyde E Martin. 1948. "Sexual Behavior in the Human Male." *W. B. Saunders Company*. https://books.google.co.uk/books?id=pfMKrY3VvigC.

Kornberg, Arthur. 1991. *For the Love of Enzymes: The Odyssey of a Biochemist*. Cambridge, Massachusetts: Harvard University Press. https://archive.org/details/forloveofenzymes00

arth.
Kotz, Samuel, and Norman Lloyd Johnson. 1992. *Breakthroughs in Statistics*. New York: Springer-Verlag.
Lee, Peter M. 2012. *Bayesian Statistics: An Introduction*. 4th ed. Wiley Online Library. https://www.york.ac.uk/depts/maths/histstat/pml1/bayes/book.htm.
Lorie, James Hirsch, and Harry V Roberts. 1951. *Basic Methods of Marketing Research*. McGraw-Hill.
Lyon, Herbert L, and Julian Lincoln Simon. 1968. "Price Elasticity of the Demand for Cigarettes in the United States." *American Journal of Agricultural Economics* 50 (4): 888–95.
Martineau, Adrian R, David A Jolliffe, Richard L Hooper, Lauren Greenberg, John F Aloia, Peter Bergman, Gal Dubnov-Raz, et al. 2017. "Vitamin D Supplementation to Prevent Acute Respiratory Tract Infections: Systematic Review and Meta-Analysis of Individual Participant Data." *Bmj* 356.
McCabe, George P, and Linda Doyle McCabe. 1989. *Instructor's Guide with Solutions for Introduction to the Practice of Statistics*. New York: W. H. Freeman.
Mosteller, Frederick. 1987. *Fifty Challenging Problems in Probability with Solutions*. Courier Corporation.
Mosteller, Frederick, and Robert E. K. Rourke. 1973. *Sturdy Statistics: Nonparametrics and Order Statistics*. Addison-Wesley Publishing Company.
Mosteller, Frederick, Robert E. K. Rourke, and George Brinton Thomas Jr. 1961. *Probability with Statistical Applications*. 2nd ed. https://archive.org/details/probabilitywiths0000most.
Noreen, Eric W. 1989. *Computer-Intensive Methods for Testing Hypotheses*. New York: John Wiley & Sons. https://archive.org/details/computerintensiv0000nore.
Peirce, Charles Sanders. 1923. *Chance, Love, and Logic: Philosophical Essays*. New York: Harcourt Brace & Company, Inc. https://www.gutenberg.org/files/65274/65274-h/65274-h.htm.
Piketty, Thomas. 2018. "Brahmin Left Vs Merchant Right: Rising Inequality & the Changing Structure of Political Conflict." 2018. https://www.prsinstitute.org/downloads/related/economics/RisingInequalityandtheChangingStructureofPoliticalConflict1.pdf.
Pitman, Edwin JG. 1937. "Significance Tests Which May Be Applied to Samples from Any Populations." *Supplement to the Journal of the Royal Statistical Society* 4 (1): 119–30. https://www.jstor.org/stable/pdf/2984124.pdf.
Raiffa, Howard. 1968. "Decision Analysis: Introductory Lectures on Choices Under Uncertainty." https://archive.org/details/decisionanalysis0000raif.
Rosner, Bernard. 1995. *Fundamentals of Biostatistics*. 4th ed. Belmont: Wadsworth Publishing Company. https://archive.org/details/fundamentalsofbi00rosn.
Ruark, Arthur Edward, and Harold Clayton Urey. 1930. *Atoms, Moleculues and Quanta*. New York, NY: McGraw-Hill book company, inc. https://archive.org/details/atomsmoleculesqu00ruar.
Russell, Bertrand. 1945. *A History of Western Philosophy*. New York: Simon; Schuster.
Savage, Leonard J. 1972. *The Foundations of Statistics*. New York: Dover Publications, Inc.

Savant, Marilyn vos. 1990. "Ask Marilyn." 1990. https://web.archive.org/web/20160318182523/http://marilynvossavant.com/game-show-problem.

Schlaifer, Robert. 1961. *Introduction to Statistics for Business Decisions*. New York: MacGraw-Hill. https://archive.org/details/introductiontost00schl.

Selvin, Steve. 1975. "Letters to the Editor." *The American Statistician* 29 (1): 67. http://www.jstor.org/stable/2683689.

Semmelweis, Ignác Fülöp. 1983. *The Etiology, Concept, and Prophylaxis of Childbed Fever*. Translated by K. Codell Carter. Madison, Wisconsin: University of Wisconsin Press. https://archive.org/details/etiologyconcepta0000unse.

Shurtleff, Dewey. 1970. "Some Characteristics Related to the Incidence of Cardiovascular Disease and Death: Framingham Study, 16-Year Follow-up." Section 26. Edited by William B. Kannel and Tavia Gordon. The Framingham Study: An Epidemiological Investigation of Cardiovascular Disease. Washington, D.C.: U.S. Government Printing Office. https://upload.wikimedia.org/wikipedia/commons/6/6d/The_Framingham_study_-_an_epidemiological_investigation_of_cardiovascular_disease_sec.26_1970_%28IA_framinghamstudye00kann_25%29.pdf.

Simon, Julian Lincoln. 1967. "Doctors, Smoking, and Reference Groups." *Public Opinion Quarterly* 31 (4): 646–47.

———. 1969. *Basic Research Methods in Social Science*. 1st ed. New York: Random House.

———. 1992. *Resampling: The New Statistics*. 1st ed. Arlington, VA: Resampling Stats Inc.

———. 1998. "The Philosophy and Practice of Resampling Statistics." 1998. http://www.juliansimon.org/writings/Resampling_Philosophy.

Simon, Julian Lincoln, David T Atkinson, and Carolyn Shevokas. 1976. "Probability and Statistics: Experimental Results of a Radically Different Teaching Method." *The American Mathematical Monthly* 83 (9): 733–39. https://www.jstor.org/stable/pdf/2318961.pdf.

Simon, Julian Lincoln, and Paul Burstein. 1985. *Basic Research Methods in Social Science*. 3rd ed. New York: Random House.

Simon, Julian Lincoln, and Allen Holmes. 1969. "A New Way to Teach Probability Statistics." *The Mathematics Teacher* 62 (4): 283–88.

Simon, Julian Lincoln, Manouchehr Mokhtari, and Daniel H Simon. 1996. "Are Mergers Beneficial or Detrimental? Evidence from Advertising Agencies." *International Journal of the Economics of Business* 3 (1): 69–82.

Simon, Julian Lincoln, and David M Simon. 1996. "The Effects of Regulations on State Liquor Prices." *Empirica* 23: 303–16.

Støvring, H. 1999. "On Radicke and His Method for Testing Mean Differences." *Journal of the Royal Statistical Society: Series D (The Statistician)* 48 (2): 189–201. https://www.jstor.org/stable/pdf/2681185.pdf.

Sudman, Seymour. 1976. *Applied Sampling*. New York: Academic Press. https://archive.org/details/appliedsampling0000unse.

Tukey, John W. 1977. *Exploratory Data Analysis*. Reading, MA, USA: Addison-Wesley.

Tversky, Amos, and Daniel Kahneman. 1982. "Evidential Impact of Base Rates." In *Judgement Under Uncertainty: Heuristics and Biases*, edited by Daniel Kahneman, Paul Slovic, and Amos Tversky. Cambridge: Cambridge University Press. https://www.google.co.uk

/books/edition/Judgment_Under_Uncertainty/_0H8gwj4a1MC.

Vazsonyi, Andrew. 1999. "Which Door Has the Cadillac." *Decision Line* 30 (1): 17–19. https://web.archive.org/web/20140413131827/http://www.decisionsciences.org/DecisionLine/Vol30/30_1/vazs30_1.pdf.

Wallis, Wilson Allen, and Harry V Roberts. 1956. *Statistics, a New Approach.* New York: The Free Press.

Whitworth, William Allen. 1897. *DCC Exercises in Choice and Chance.* Cambridge, UK: Deighton Bell; Co. https://archive.org/details/dccexerciseschoi00whit.

Winslow, Charles-Edward Amory. 1980. *The Conquest of Epidemic Disease: A Chapter in the History of Ideas.* Madison, Wisconsin: University of Wisconsin Press. https://archive.org/details/conquestofepidem0000wins_p3k0.

Wonnacott, Thomas H, and Ronald J Wonnacott. 1990. *Introductory Statistics.* 5th ed. New York: John Wiley & Sons.

Zhou, Qixing, Christopher E Gibson, and Robert H Foy. 2000. "Long-Term Changes of Nitrogen and Phosphorus Loadings to a Large Lake in North-West Ireland." *Water Research* 34 (3): 922–26. https://doi.org/10.1016/S0043-1354(99)00199-2.

A Exercise Solutions

A.1 Solution for paired differences exercise 24.3.1

We suggested that you ignored the pairing of the before and after samples, and that is what we will do here. Then we will extend the treatment to take the pairing into account.

Note 90: Notebook: Paired differences solution

- Download zip with notebook + data file
- Interact

```
import numpy as np
import pandas as pd

rnd = np.random.default_rng()

df = pd.read_csv('data/hamilton.csv')
before = np.array(df['score_before'])
after = np.array(df['score_after'])

observed_diff = np.mean(after) - np.mean(before)

# Let us start with a permutation test.
both = np.concatenate([before, after])
n_before = len(before)

# Samples in the null world.
n_trials = 10_000
results = np.zeros(n_trials)
for i in range(n_trials):
    shuffled = rnd.permuted(both)
    fake_before = shuffled[:n_before]
    fake_after = shuffled[n_before:]
    fake_diff = np.mean(fake_after) - np.mean(fake_before)
    results[i] = fake_diff
```

```
# We are interested in fake differences that are larger
# in magnitude than the observed difference (hence "abs").
# Here we have no prior hypothesis about which direction the difference
# will go.
k = np.sum(np.abs(results) >= np.abs(observed_diff))
kk = k / n_trials
print('Permutation p null-world abs >= abs observed:', kk)
```

```
Permutation p null-world abs >= abs observed: 0.2554
```

```
# Next a bootstrap test.
n_after = len(after)  # Of course, in our case, this will be == n_before
results = np.zeros(n_trials)
for i in range(n_trials):
    fake_before = rnd.choice(both, size=n_before)
    fake_after = rnd.choice(both, size=n_after)
    fake_diff = np.mean(fake_after) - np.mean(fake_before)
    results[i] = fake_diff

k = np.sum(np.abs(results) >= np.abs(observed_diff))
kk = k / n_trials
print('Bootstrap p null-world abs >= abs observed:', kk)
```

```
Bootstrap p null-world abs >= abs observed: 0.2217
```

Finally we consider the pairs. Here we *do* take the pairs into account. We have some reason to think that the patients or cars vary in some substantial way in their level of depression, or their tendency to break down, but we believe that the patients' *response* to treatment or the difference between the mechanics is the value of interest.

In that case, we are interested in the *differences* between the pairs. In the null world, these before / after (mechanic A / mechanic B) differences are random. In the null-world, where there is no difference between before/after or mechanics 1 and 2, we can flip the before / after (A / B) pairs and be in the same world.

Notice that flipping the before / after or A / B in the pair just changes the sign of the difference.

So we will simulate the effect of flipping the values in the pair, by choosing a random sign for the pair, where -1 means pair is flipped, and 1 means pair is in original order. We recalculated the mean difference with these random signs (flips) applied, and these will be our values in the null-world.

```
# A test of paired difference with sign flips for the null world.
differences = after - before
observed_mdiff = np.mean(differences)
n_both = len(differences)

results = np.zeros(n_trials)
for i in range(n_trials):
    # Choose random signs to perform random flips of the pairs.
    signs = rnd.choice([-1, 1], size=n_both)
    # Do flips.
    fake_differences = signs * differences
    # Calculate mean difference and store result.
    results[i] = np.mean(fake_differences)

k = np.sum(np.abs(results) >= np.abs(observed_mdiff))
kk = k / n_trials
print('Sign-flip p null-world abs >= abs observed:', kk)
```

```
Sign-flip p null-world abs >= abs observed: 0.0271
```

Notice that the sign-flip test, in which we preserve the information about the patients / cars, is much more convincing than the permutation or bootstrap tests above, where we choose to ignore that information.

This can occur when the values within the pairs (rows) are similar to each other, but less similar across different pairs (rows).

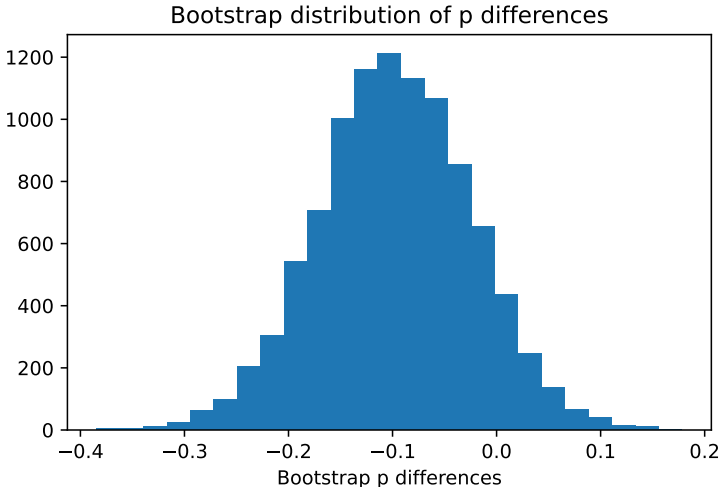
> **i** End of notebook: Paired differences solution
>
> `paired_differences_solution` starts at Note 90.

A.2 Solution to seatbelt proportions exercise 24.3.2

> **i** Note 91: Notebook: Seatbelt proportion solution
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

pittsburgh = np.repeat(['seatbelt', 'none'], [36, 36])
n_pitts = len(pittsburgh)
chicago = np.repeat(['seatbelt', 'none'], [77, 52])
n_chicago = len(chicago)

n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    fake_pitts = rnd.choice(pittsburgh, size=n_pitts)
    fake_chicago = rnd.choice(chicago, size=n_chicago)
    fake_p_pitts = np.sum(fake_pitts == 'seatbelt') / n_pitts
    fake_p_chicago = np.sum(fake_chicago == 'seatbelt') / n_chicago
    fake_p_diff = fake_p_pitts - fake_p_chicago
    results[i] = fake_p_diff

plt.hist(results, bins=25)
plt.title('Bootstrap distribution of p differences')
plt.xlabel('Bootstrap p differences')

p_limits = np.quantile(results, [0.025, 0.975])

print('95% percent limits for p differences:', p_limits)
```

```
95% percent limits for p differences: [-0.24386305  0.04521964]
```



End of notebook: Seatbelt proportion solution

`seatbelt_proportion_solution` starts at Note 91.

A.3 Solution for unemployment percentage 27.8.1

In a sample of 200 people, 7 percent are found to be unemployed. Determine a 95 percent confidence interval for the true population proportion.

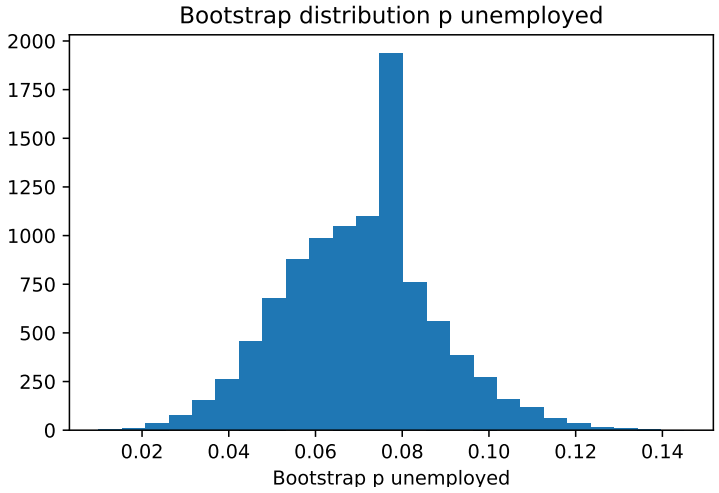
Note 92: Notebook: Unemployment percent solution

- Download notebook
- Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    fake_people = rnd.choice(['no job', 'job'], size=200, p=[0.07, 0.93])
```

```
    p_unemployed = np.sum(fake_people == 'no job') / 200
    results[i] = p_unemployed

plt.hist(results, bins=25)
plt.title('Bootstrap distribution p unemployed')
plt.xlabel('Bootstrap p unemployed')

p_limits = np.quantile(results, [0.025, 0.975])

print('95% percent limits for p differences:', p_limits)
```

```
95% percent limits for p differences: [0.035 0.105]
```



End of notebook: Unemployment percent solution

`unemployment_percent_solution` starts at Note 92.

A.4 Solution for battery lifetime 27.8.2

A sample of 20 batteries is tested, and the average lifetime is 28.85 months. Establish a 95 percent confidence interval for the true average value. The sample values (lifetimes in months) are listed below.

We use the "bootstrap" technique of drawing many bootstrap re-samples with replacement from the original sample, and observing how the re-sample means are distributed.

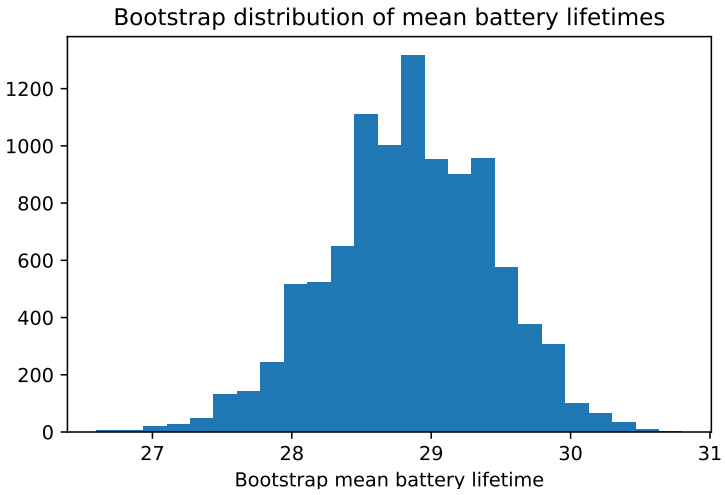
> **Note 93: Notebook: Battery lifetime solution**
>
> - Download notebook
> - Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()

lifetimes = np.array([30, 32, 31, 28, 31, 29, 29, 24, 30, 31,
                      28, 28, 32, 31, 24, 23, 31, 27, 27, 31])

print('Mean is:', np.mean(lifetimes))
```

```
Mean is: 28.85
```

```
n_lifetimes = len(lifetimes)
results = np.zeros(n_trials)

for i in range(n_trials):
    # Draw 20 lifetimes from "lifetimes, randomly and with replacement.
    fake_lifetimes = rnd.choice(lifetimes, size=n_lifetimes)
    # Find the average lifetime of the 20.
    fake_mean = np.mean(fake_lifetimes)
    # Keep score.
    results[i] = fake_mean

plt.hist(results, bins=25)
plt.title('Bootstrap distribution of mean battery lifetimes')
plt.xlabel('Bootstrap mean battery lifetime')

mean_limits = np.quantile(results, [0.025, 0.975])

print('95% percent limits for mean lifetimes:', mean_limits)
```

```
95% percent limits for mean lifetimes: [27.65 29.95]
```



End of notebook: Battery lifetime solution

`battery_lifetime_solution` starts at Note 93.

A.5 Solution for optical density 27.8.3

Note 94: Notebook: Optical density solution

- Download notebook
- Interact

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.default_rng()
```

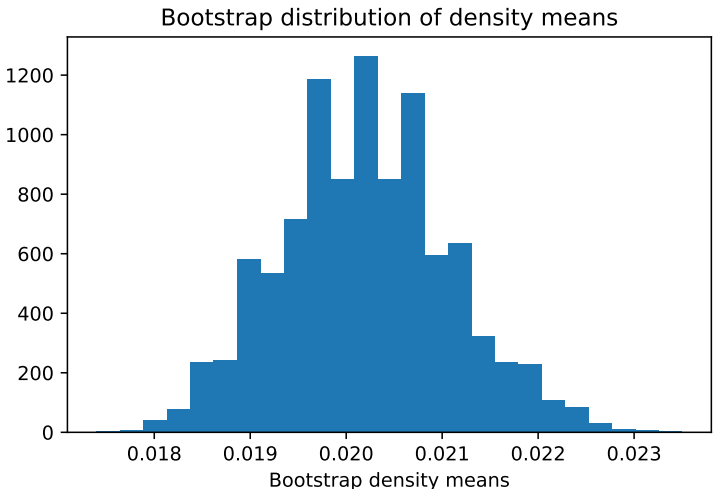
Suppose we have 10 measurements of Optical Density on a batch of HIV negative control samples:

```
density = np.array(
    [.02, .026, .023, .017, .022, .019, .018, .018, .017, .022])
```

Derive a 95 percent confidence interval for the sample mean. Are there enough measurements to produce a satisfactory answer?

```
n_density = len(density)

n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    fake_density = rnd.choice(density, size=n_density)
    results[i] = np.mean(fake_density)

plt.hist(results, bins=25)
plt.title('Bootstrap distribution of density means')
plt.xlabel('Bootstrap density means')

mean_limits = np.quantile(results, [0.025, 0.975])

print('95% percent limits for density mean:', mean_limits)
```

```
95% percent limits for density mean: [0.0185 0.022 ]
```



End of notebook: Optical density solution

`optical_density_solution` starts at Note 94.

A.6 Solution for voter participation 29.7.1

The observed correlation coefficient between voter participation and spread is moderate and negative. Is this more negative that what might occur by chance, if no correlation exists in some underlying population, from which this sample was taken?

1. Create two groups of paper cards: 25 with participation rates, and 25 with the spread values. Arrange the cards in pairs in accordance with the table, and compute the correlation coefficient between the shuffled participation and spread variables.
2. Shuffle one of the sets, say that with participation, and compute correlation between shuffled participation and spread.
3. Repeat step 2 many, say 1000, times. Compute the proportion of the trials in which correlation was at least as negative as that for the original data.

> **i** Note 95: Notebook: Voter participation in 1844 election
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np
import pandas as pd

rnd = np.random.default_rng()

voter_df = pd.read_csv('data/election_1844.csv')
participation = np.array(voter_df['Participation'])
spread = np.array(voter_df['Spread'])
```

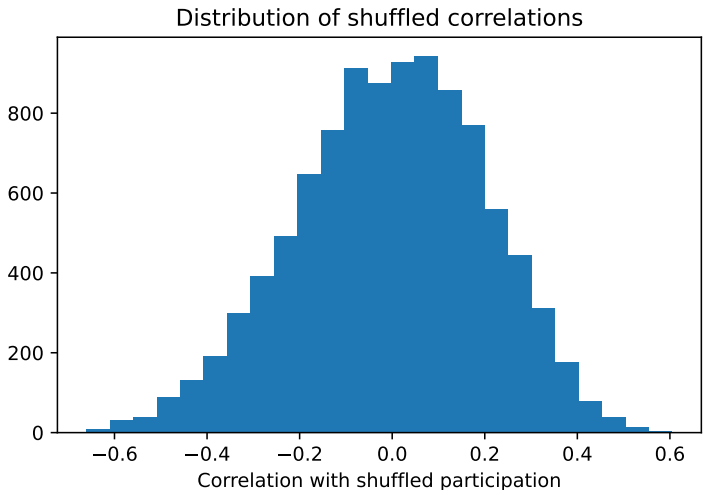
```
# Compute correlation.  It's -0.425.
actual_r = np.corrcoef(participation, spread)[0, 1]
actual_r
```

```
np.float64(-0.4249067562318352)
```

```
n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    # Shuffle the participation rates.
    shuffled = rnd.permuted(participation)
    # Compute re-sampled correlation.
```

```
    fake_r = np.corrcoef(shuffled, spread)[0, 1]
    # Keep the value in the results.
    results[i] = fake_r

plt.hist(results, bins=25)
plt.title('Distribution of shuffled correlations')
plt.xlabel('Correlation with shuffled participation')

# Count the trials when result <= observed.
k = np.sum(results <= actual_r)
# Compute the proportion of such trials.
kk = k / n_trials

print('Proportion of shuffled r <= observed:', np.round(kk, 2))
```

```
Proportion of shuffled r <= observed: 0.03
```



End of notebook: Voter participation in 1844 election

`voter_participation_solution` starts at Note 95.

From this we may conclude that the voter participation rates probably are negatively related to the vote spread in the election. The actual value of the correlation (-.425) cannot be explained by chance alone. In our Monte Carlo simulation of the null-hypothesis a correlation that negative is found only about 3 percent of the time.

See: Section A.6.

A.7 Solution for association of runs and strikeouts 29.7.2

We are looking at the correlation of home runs and strikeouts for major-league baseball players.

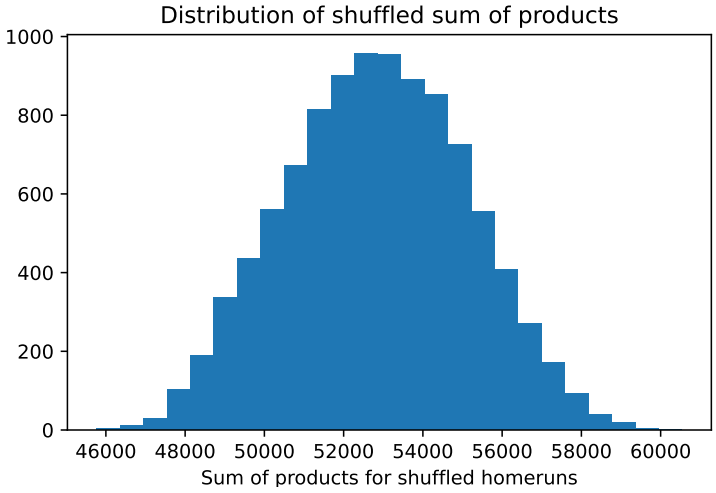
The instructions ask us to start here with the sum-of-products measure.

> Note 96: Notebook: Homeruns and strikeout sum of products.
>
> - Download notebook
> - Interact

```
import numpy as np

rnd = np.random.default_rng()

homeruns = np.array([14, 20, 0, 38, 9, 38, 22, 31, 33,
                     11, 40, 5, 15, 32, 3, 29, 5, 32])
strikeout = np.array([135, 153, 120, 161, 138, 175, 126, 200, 205,
                      147, 165, 124, 169, 156, 36, 98, 82, 131])
```

```
# The sum of products approach.
actual_sop = np.sum(homeruns * strikeout)

n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    shuffled_runs = rnd.permuted(homeruns)
    fake_sop = np.sum(shuffled_runs * strikeout)
    results[i] = fake_sop

plt.hist(results, bins=25)
plt.title('Distribution of shuffled sum of products')
plt.xlabel('Sum of products for shuffled homeruns')

k = np.sum(results >= actual_sop)
kk = k / n_trials

print('p shuffled sum of products >= actual:', np.round(kk, 3))
```

```
p shuffled sum of products >= actual: 0.003
```



Interpretation: In 10,000 simulations, random shuffling very rarely produced a value as high as observed. Therefore, we conclude that random chance could not reasonably be responsible for the observed degree of correlation.

ℹ End of notebook: Homeruns and strikeout sum of products.

`homerun_sop_solution` starts at Note 96.

A.8 Solution for runs, strikeouts and correlation coefficient 29.7.3

Again, we are looking at the correlation of home runs and strikeouts for major-league baseball players. This time we will use the correlation coefficient (r) measure.

ℹ Note 97: Notebook: Homeruns and strikeout correlation

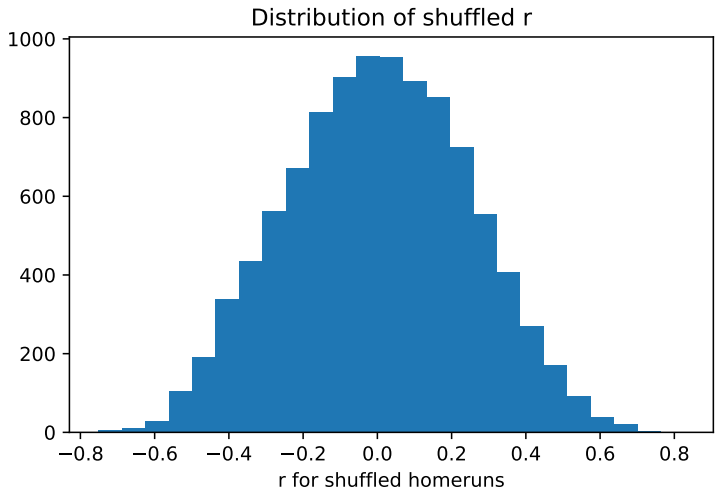
- Download notebook
- Interact

```
import numpy as np

rnd = np.random.default_rng()

homeruns = np.array([14, 20, 0, 38, 9, 38, 22, 31, 33,
                     11, 40, 5, 15, 32, 3, 29, 5, 32])
```

```
strikeout = np.array([135, 153, 120, 161, 138, 175, 126, 200, 205,
                      147, 165, 124, 169, 156, 36, 98, 82, 131])
```

```
# The correlation approach.
actual_r = np.corrcoef(homeruns, strikeout)[0, 1]

n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
    shuffled_runs = rnd.permuted(homeruns)
    fake_r = np.corrcoef(shuffled_runs, strikeout)[0, 1]
    results[i] = fake_r

plt.hist(results, bins=25)
plt.title('Distribution of shuffled r')
plt.xlabel('r for shuffled homeruns')

k = np.sum(results >= actual_r)
kk = k / n_trials

print('p shuffled r >= actual:', np.round(kk, 3))
```

```
p shuffled r >= actual: 0.003
```



Interpretation: a correlation coefficient as high as the observed value (.62) occurred only about 0.3% of the time by chance. Hence, provisionally, we choose to reject chance as an explanation for such a high value of the correlation coefficient.

Notice, we get the same answer for correlation coefficients as we do for sum of products. In fact, correlation coefficients must give us the same answer (apart from random variation from the permutation), as the tests of association are equivalent when we compare between different orderings of the same sequences.

> **i** End of notebook: Homeruns and strikeout correlation
>
> `homerun_correlation_solution` starts at Note 97.

A.9 Solution for money and exchange rate 29.7.4

> **i** Note 98: Notebook: Exchange rates and money supply
>
> - Download zip with notebook + data file
> - Interact

```
import numpy as np
import pandas as pd

rnd = np.random.default_rng()

exchange_df = pd.read_csv('data/exchange_rates.csv')
exchange_rates = np.array(exchange_df['exchange_rate'])
money_supply = np.array(exchange_df['money_supply'])
```

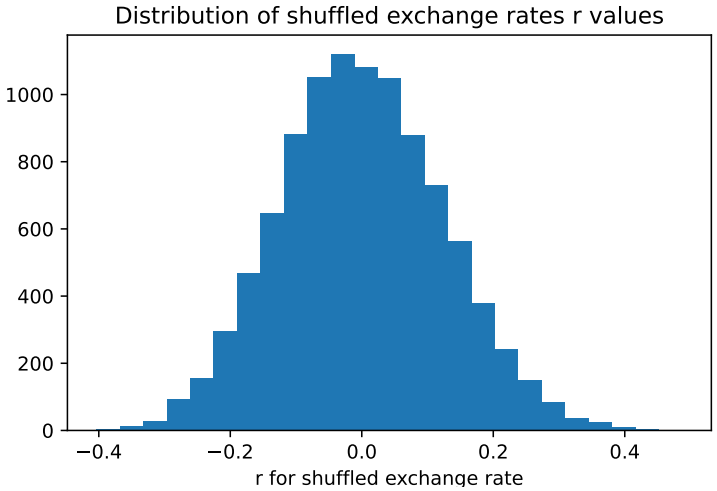
```
# Correlation.
actual_r = np.corrcoef(exchange_rates, money_supply)[0, 1]
actual_r
```

```
np.float64(0.4206068712932843)
```

```
n_trials = 10_000
results = np.zeros(n_trials)

for i in range(n_trials):
```

```
    shuffled_rates = rnd.permuted(exchange_rates)
    fake_r = np.corrcoef(shuffled_rates, money_supply)[0, 1]
    results[i] = fake_r

plt.hist(results, bins=25)
plt.title('Distribution of shuffled exchange rates r values')
plt.xlabel('r for shuffled exchange rate')

k = np.sum(results >= actual_r)
kk = k / n_trials

print('p shuffled r >= actual:', np.round(kk, 3))
```

```
p shuffled r >= actual: 0.0
```



End of notebook: Exchange rates and money supply

`exchange_rates_solution` starts at Note 98.

Interpretation: The observed correlation (.42) between the exchange rate and the money supply is seldom exceeded by random experiments with these data. Thus, the observed result 0.42 cannot be reasonably explained by chance alone and we conclude that it is statistically surprising.

B Technical Note to the Professional Reader

The material presented in this book fits together with the technical literature as follows: Though I (JLS) had proceeded from first principles rather than from the literature, I have from the start cited work by Chung and Fraser (1958) and Meyer Dwass (1957) They suggested taking samples of permutations in a two-sample test as a way of extending the applicability of Fisher's randomization test (1935; 1960, chap. III, section 21). Resampling with replacement from a single sample to determine sample statistic variability was suggested by Simon (1969). Independent work by Efron (1979) explored the properties of this technique (Efron termed it the "bootstrap") and lent it theoretical support. The notion of using these techniques routinely and in preference to conventional techniques based on Gaussian assumptions was suggested by Simon (1969) and by Simon, Atkinson, and Shevokas (1976).

C Acknowledgements

C.1 For the second edition

Many people have helped in the long evolution of this work. First was the late Max Beberman, who in 1967 immediately recognized the potential of resampling statistics for high school students as well as for all others. Louis Guttman and Joseph Doob provided important encouragement about the theoretical and practical value of resampling statistics. Allen Holmes cooperated with me in teaching the first class at University High School in Urbana, Illinois, in 1967. Kenneth Travers found and supervised several PhD students — David Atkinson and Carolyn Shevokas outstanding among them — who experimented with resampling statistics in high school and college classrooms and proved its effectiveness; Travers also carried the message to many secondary school teachers in person and in his texts. In 1973 Dan Weidenfield efficiently wrote the first program for the mainframe (then called "Simple Stats"). Derek Kumar wrote the first interactive program for the Apple II. Chad McDaniel developed the IBM version, with touchup by Henry van Kuijk and Yoram Kochavi. Carlos Puig developed the powerful 1990 version of the program. William E. Kirwan, Robert Dorfman, and Rudolf Lamone have provided their good offices for us to harness the resources of the University of Maryland and, in particular, the College of Business and Management. Terry Oswald worked day and night with great dedication on the program and on commercial details to start the marketing of RESAMPLING STATS. In mid-1989, Peter Bruce assumed the overall stewardship of RESAMPLING STATS, and has been proceeding with energy, good judgment, and courage. He has contributed to this volume in many ways, always excellently (including the writing and re-writing of programs, as well as explanations of the bootstrap and of the interpretation of p-values). Vladimir Koliadin wrote the code for several of the problems in this edition, and Cheinan Marks programmed the Windows and Macintosh versions of Resampling Stats. Toni York handled the typesetting and desktop publishing through various iterations, Barbara Shaw provided expert proofreading and desktop publishing services for the second printing of the second edition, and Chris Brest produced many of the figures. Thanks to all of you, and to others who should be added to the list.

D Code topics

This appendix gathers together links to parts of the book that cover various code topics in Python:

- Comments: Note 2 and Note 8.
- Variables: Section 4.7.
- Types of values in Python: Section 7.3.
  - Lists in Python: Section 7.3.2.
  - Arrays: Section 7.3.1.
- array length: Section 6.5.
- Indexing into arrays with integers: Section 6.6.
- Selecting multiple elements from arrays with slicing: Section 10.2.
- Strings — values containing text: Section 7.4.
- Randomness from your computer: Section 2.2.
- Numpy's random number generator: **?@nte-numpy-rng**.
- Types of brackets in Python: Note 4.
- Squaring values and arrays: Section 16.7.1.
- Introducing functions: Section 5.7.
- Named arguments to functions: Section 5.8.
- Functions and methods: Note 6.
- Ranges (continuous sequences) of integers: Section 5.9.
- `range` in Python (compared to `np.arange`): Section 5.10.
- Advanced ranges (for example, using floating point values), using `np.arange`: Note 45.
- Repeating elements of an array with `np.repeat`: Section 7.6}
- The *absolute* function (`np.abs`) to convert all values to positive: Note 38.
- `for` loops: Section 6.6.2.
- `if` statements: Section 8.10.
- Python modules and submodules: Note 24.
- Finding number of repeats using `np.bincount`: Section 11.6
- Combining Boolean arrays with `&`: Section 10.6
- `range` in Python `for` loops: Section 6.6.3
- Testing whether all elements of an array are the same using `np.all`: Note 30
- Plotting and histograms (and histogram bins and bin edges): Section 12.15.2.
- Shuffling arrays with `rnd.permuted`: Section 8.14.
- Choosing values at random with `np.choice`: Section 5.11.

- Counting `True` values with `sum`: Section 5.14.
- Making arrays by random resampling with `np.choice`: Section 5.12.
- Underscores in Python integers: Note 43.
- Building strings with format-strings: Section 21.1.1.
- What is an operator?: Note 51.
- Logical operators — *or*, *and*: Section 23.1.
- Concatenating arrays: Section 12.15.1.
- The Comma-Separated-Values (CSV) file format for data tables: Section 16.1.1.
- Introducing the Pandas Python data science library: Section 16.1.2.

E Errors and suggestions

We are hoping that you have arrived at this chapter because you have found an error in the book, or you have thought of an improvement that you would like to share.

There are two ways of letting us know about errors or fixes.

E.1 The easiest way — making an *issue*

The simplest method is to make a new "issue" using the Github website interface that we use to house the book. To do this, go to https://github.com/resampling-stats/resampling-with/issues, and click on the green "New Issue" button. Fill in as much information as you can think of about the error or fix or other improvement, and click "Submit new issue". In particular, please make sure to tell us the chapter and page number (from the PDF), or the web address of the text with the error. We will try to respond quickly.

E.2 Note on licensing

Please bear in mind that we have — with permission — released the book under a Creative Commons attribution / non-commercial license. In order to preserve our ability to keep using this license, we will assume that you will accept that license for any fixes or improvements that you send us. If that is not true, please let us know in your issue or pull-request (see below).

E.3 More advanced — making a *pull request*

If you have some more time, and experience, you might consider sending us your suggested changes as a *pull-request*.

As background, we have written this edition of the book in a set of computer text files. In these text files, we use conventions from a definition called Markdown to indicate things like **bold** and *italic* formatting, as well as links (like the one above), and other things.

We then use some software called Quarto to take the collection of text files, and build them into the book — both in HTML (web page) format, and in PDF format.

This is relevant, because we have stored all the text files and the instructions to build the book at the Github repository at https://github.com/resampling-stats/resampling-with. If you have some experience of using such repositories, we would be very grateful if you would consider making a *fork* of the repository and making a *pull-request* from this fork. If you don't know what these terms mean, don't worry, and consider submitting an issue instead (see above).

If you don't know these terms, and you are interested to learn, we suggest you use your search engine to look for "Github fork pull-request" (without quotes) to start your research.

Please have a look at the `README.md` file in the repository for some instructions, and feel free to contact us to ask for help, either by an issue (above) or by email.

If you do submit a pull-request, please note the comment above (Section E.2) about the book license.