Chapter 1

Introduction

In God we trust; all others must bring data.
—William Edwards Deming

Quantitative social science is an interdisciplinary field encompassing a large number of disciplines, including economics, education, political science, public policy, psychology, and sociology. In quantitative social science research, scholars analyze data to understand and solve problems about society and human behavior. For example, researchers examine racial discrimination in the labor market, evaluate the impact of new curricula on students' educational achievements, predict election outcomes, and analyze social media usage. Similar data-driven approaches have been taken up in other neighboring fields such as health, law, journalism, linguistics, and even literature. Because social scientists directly investigate a wide range of real-world issues, the results of their research have enormous potential to directly influence individual members of society, government policies, and business practices.

Over the last couple of decades, quantitative social science has flourished in a variety of areas at an astonishing speed. The number of academic journal articles that present empirical evidence from data analysis has soared. Outside academia, many organizations—including corporations, political campaigns, news media, and government agencies—increasingly rely on data analysis in their decision-making processes. Two transformative technological changes have driven this rapid growth of quantitative social science. First, the Internet has greatly facilitated the *data revolution*, leading to a spike in the amount and diversity of available data. Information sharing makes it possible for researchers and organizations to disseminate numerous data sets in digital form. Second, the *computational revolution*, in terms of both software and hardware, means that anyone can conduct data analysis using their personal computer and favorite data analysis software.

As a direct consequence of these technological changes, the sheer volume of data available to quantitative social scientists has rapidly grown. In the past, researchers largely relied upon data published by governmental agencies (e.g., censuses, election outcomes, and economic indicators) as well as a small number of data sets collected by research groups (e.g., survey data from national election studies and hand-coded data sets about war occurrence and democratic institutions). These data sets still play an important role in empirical analysis. However, the wide variety of new data has significantly expanded the horizon of quantitative social science research. Researchers are designing and conducting randomized experiments and surveys on their own. Under pressure to increase transparency and accountability, government agencies are making more data publicly available online. For example, in the United States, anyone can download detailed data on campaign contributions and lobbying activities to their personal computers. In Nordic countries like Sweden, a wide range of registers, including income, tax, education, health, and workplace, are available for academic research.

New data sets have emerged across diverse areas. Detailed data about consumer transactions are available through electronic purchasing records. International trade data are collected at the product level between many pairs of countries over several decades. Militaries have also contributed to the data revolution. During the Afghanistan war in the 2000s, the United States and international forces gathered data on the geo-location, timing, and types of insurgent attacks and conducted data analysis to guide counterinsurgency strategy. Similarly, governmental agencies and nongovernmental organizations collected data on civilian casualties from the war. Political campaigns use data analysis to devise voter mobilization strategies by targeting certain types of voters with carefully selected messages.

These data sets also come in varying forms. Quantitative social scientists are analyzing digitized texts as data, including legislative bills, newspaper articles, and the speeches of politicians. The availability of social media data through websites, blogs, tweets, SMS messaging, and Facebook has enabled social scientists to explore how people interact with one another in the online sphere. Geographical information system (GIS) data sets are also widespread. They enable researchers to analyze the legislative redistricting process or civil conflict with attention paid to spatial location. Others have used satellite imagery data to measure the level of electrification in rural areas of developing countries. While still rare, images, sounds, and even videos can be analyzed using quantitative methods for answering social science questions.

Together with the revolution of information technology, the availability of such abundant and diverse data means that anyone, from academics to practitioners, from business analysts to policy makers, and from students to faculty, can make data-driven discoveries. In the past, only statisticians and other specialized professionals conducted data analysis. Now, everyone can turn on their personal computer, download data from the Internet, and analyze them using their favorite software. This has led to increased demands for accountability to demonstrate policy effectiveness. In order to secure funding and increase legitimacy, for example, nongovernmental organizations and governmental agencies must now demonstrate the efficacy of their policies and programs through rigorous evaluation.

This shift towards greater transparency and data-driven discovery requires that students in the social sciences learn how to analyze data, interpret the results, and effectively communicate their empirical findings. Traditionally, introductory statistics courses have focused on teaching students basic statistical concepts by having them conduct straightforward calculations with paper and pencil or, at best, a scientific calculator. Although these concepts are still important and covered in this book, this traditional approach cannot meet the current demands of society. It is simply not sufficient to achieve "statistical literacy" by learning about common statistical concepts and methods. Instead, all students in the social sciences should acquire basic data analysis skills so that they can exploit the ample opportunities to learn from data and make contributions to society through data-driven discovery.

The belief that everyone should be able to analyze data is the main motivation for writing this book. The book introduces the three elements of data analysis required for quantitative social science research: research contexts, programming techniques, and statistical methods. Any of these elements in isolation is insufficient. Without research contexts, we cannot assess the credibility of assumptions required for data analysis and will not be able to understand what the empirical findings imply. Without programming techniques, we will not be able to analyze data and answer research questions. Without the guidance of statistical principles, we cannot distinguish systematic patterns, known as signals, from idiosyncratic ones, known as noise, possibly leading to invalid inference. (Here, inference refers to drawing conclusions about unknown quantities based on observed data.) This book demonstrates the power of data analysis by combining these three elements.

1.1 Overview of the Book

This book is written for anyone who wishes to learn data analysis and statistics for the first time. The target audience includes researchers, undergraduate and graduate students in social science and other fields, as well as practitioners and even ambitious high-school students. The book has no prerequisite other than some elementary algebra. In particular, readers do not have to possess knowledge of calculus or probability. No programming experience is necessary, though it can certainly be helpful. The book is also appropriate for those who have taken a traditional "paper-and-pencil" introductory statistics course where little data analysis is taught. Through this book, students will discover the excitement that data analysis brings. Those who want to learn R programming might also find this book useful, although here the emphasis is on how to use R to answer quantitative social science questions.

As mentioned above, the unique feature of this book is the presentation of programming techniques and statistical concepts simultaneously through analysis of data sets taken directly from published quantitative social science research. The goal is to demonstrate how social scientists use data analysis to answer important questions about societal problems and human behavior. At the same time, users of the book will learn fundamental statistical concepts and basic programming skills. Most importantly, readers will gain experience with data analysis by examining approximately forty data sets.

The book consists of eight chapters. The current introductory chapter explains how to best utilize the book and presents a brief introduction to R, a popular open-source statistical programming environment. R is freely available for download and runs on Macintosh, Windows, and Linux computers. Readers are strongly encouraged to use RStudio, another freely available software package that has numerous features to make data analysis easier. This chapter ends with two exercises that are designed to help readers practice elementary R functionalities using data sets from published social science research. All data sets used in this book are freely available for download via links from `http://press.princeton.edu/qss/`. Links to other useful materials, such as the review exercises for each chapter, can also be found on the website. With the exception of chapter 5, the book focuses on the most basic syntax of R and does not introduce the wide range of additional packages that are available. However, upon completion of this book, readers will have acquired enough R programming skills to be able to utilize these packages.

Chapter 2 introduces *causality*, which plays an essential role in social science research whenever we wish to find out whether a particular policy or program changes an outcome of interest. Causality is notoriously difficult to study because we must infer counterfactual outcomes that are not observable. For example, in order to understand the existence of racial discrimination in the labor market, we need to know whether an African-American candidate who did not receive a job offer would have done so if they were white. We will analyze the data from a well-known experimental study in which researchers sent the résumés of fictitious job applicants to potential employers after randomly choosing applicants' names to sound either African-American or Caucasian. Using this study as an application, the chapter will explain how the randomization of treatment assignment enables researchers to identify the average causal effect of the treatment.

Additionally, readers will learn about causal inference in observational studies where researchers do not have control over treatment assignment. The main application is a classic study whose goal was to figure out the impact of increasing the minimum wage on employment. Many economists argue that a minimum-wage increase can reduce employment because employers must pay higher wages to their workers and are therefore made to hire fewer workers. Unfortunately, the decision to increase the minimum wage is not random, but instead is subject to many factors, like economic growth, that are themselves associated with employment. Since these factors influence which companies find themselves in the treatment group, a simple comparison between those who received treatment and those who did not can lead to biased inference.

We introduce several strategies that attempt to reduce this type of selection bias in observational studies. Despite the risk that we will inaccurately estimate treatment effects in observational studies, the results of such studies are often easier to generalize than those obtained from randomized controlled trials. Other examples in chapter 2 include a field experiment concerning social pressure in get-out-the-vote mobilization. Exercises then include a randomized experiment that investigates the causal effect of small class size in early education as well as a natural experiment about political leader assassination and its effects. In terms of R programming, chapter 2 covers logical statements and subsetting.

Chapter 3 introduces the fundamental concept of *measurement*. Accurate measurement is important for any data-driven discovery because bias in measurement can lead to incorrect conclusions and misguided decisions. We begin by considering how to measure public opinion through sample surveys. We analyze the data from a study in which researchers attempted to measure the degree of support among Afghan citizens for international forces and the Taliban insurgency during the Afghanistan war. The chapter explains the power of randomization in survey sampling. Specifically, random sampling of respondents from a population allows us to obtain a representative sample. As a result, we can infer the opinion of an entire population by analyzing one small representative group. We also discuss the potential biases of survey sampling. Nonresponses can compromise the representativeness of a sample. Misreporting poses a serious threat to inference, especially when respondents are asked sensitive questions, such as whether they support the Taliban insurgency.

The second half of chapter 3 focuses on the measurement of latent or unobservable concepts that play a key role in quantitative social science. Prominent examples of such concepts include ability and ideology. In the chapter, we study political ideology. We first describe a model frequently used to infer the ideological positions of legislators from roll call votes, and examine how the US Congress has polarized over time. We then introduce a basic clustering algorithm, k-means, that makes it possible for us to find groups of similar observations. Applying this algorithm to the data, we find that in recent years, the ideological division within Congress has been mainly characterized by the party line. In contrast, we find some divisions within each party in earlier years. This chapter also introduces various measures of the spread of data, including quantiles, standard deviation, and the Gini coefficient. In terms of R programming, the chapter introduces various ways to visualize univariate and bivariate data. The exercises include the reanalysis of a controversial same-sex marriage experiment, which raises issues of academic integrity while illustrating methods covered in the chapter.

Chapter 4 considers *prediction*. Predicting the occurrence of certain events is an essential component of policy and decision-making processes. For example, the forecasting of economic performance is critical for fiscal planning, and early warnings of civil unrest allow foreign policy makers to act proactively. The main application of this chapter is the prediction of US presidential elections using preelection polls. We show that we can make a remarkably accurate prediction by combining multiple polls in a straightforward manner. In addition, we analyze the data from a psychological experiment in which subjects are shown the facial pictures of unknown political candidates and asked to rate their competence. The analysis yields the surprising result that a quick facial impression can predict election outcomes. Through this example, we introduce linear regression models, which are useful tools to predict the values of one variable based on another variable. We describe the relationship between linear regression and correlation, and examine the phenomenon called "regression towards the mean," which is the origin of the term "regression."

Chapter 4 also discusses when regression models can be used to estimate causal effects rather than simply make predictions. Causal inference differs from standard prediction in requiring the prediction of counterfactual, rather than observed, outcomes using the treatment variable as the predictor. We analyze the data from a randomized natural experiment in India where randomly selected villages reserved some of the seats in their village councils for women. Exploiting this randomization, we investigate whether or not having female politicians affects policy outcomes, especially concerning the policy issues female voters care about. The chapter also introduces the regression discontinuity design for making causal inference in observational studies. We investigate how much of British politicians' accumulated wealth is due to holding political office. We answer this question by comparing those who barely won an election with those who narrowly lost it. The chapter introduces powerful but challenging R programming concepts: loops and conditional statements. The exercises at the end of the chapter include an analysis of whether betting markets can precisely forecast election outcomes.

Chapter 5 is about the *discovery* of patterns from data of various types. When analyzing "big data," we need automated methods and visualization tools to identify consistent patterns in the data. First, we analyze texts as data. Our primary application here is authorship prediction of *The Federalist Papers*, which formed the basis of the US Constitution. Some of the papers have known authors while others do not. We show that by analyzing the frequencies of certain words in the papers with known authorship, we can predict whether Alexander Hamilton or James Madison authored each of the papers with unknown authorship. Second, we show how to analyze network data, focusing on explaining the relationships among units. Within marriage networks in Renaissance Florence, we quantify the key role played by the Medici family. As a more contemporary example, various measures of centrality are introduced and applied to social media data generated by US senators on Twitter.

Finally in chapter 5, we introduce geo-spatial data. We begin by discussing the classic spatial data analysis conducted by John Snow to examine the cause of the 1854 cholera outbreak in London. We then demonstrate how to visualize spatial data through the creation of maps, using US election data as an example. For spatial–temporal data, we create a series of maps as an animation in order to visually characterize changes in spatial patterns over time. Thus, the chapter applies various data visualization techniques using several specialized R packages.

Chapter 6 shifts the focus from data analysis to *probability*, a unified mathematical model of uncertainty. While earlier chapters examine how to estimate parameters and make predictions, they do not discuss the level of uncertainty in empirical findings, a topic that chapter 7 introduces. Probability is important because it lays a foundation for statistical inference, the goal of which is to quantify inferential uncertainty. We begin by discussing the question of how to interpret probability from two dominant perspectives, frequentist and Bayesian. We then provide mathematical definitions of probability and conditional probability, and introduce several fundamental rules of probability. One such rule is called Bayes' rule. We show how to use Bayes' rule and accurately predict individual ethnicity using surname and residence location when no survey data are available.

This chapter also introduces the important concepts of random variables and probability distributions. We use these tools to add a measure of uncertainty to election predictions that we produced in chapter 4 using preelection polls. Another exercise adds uncertainty to the forecasts of election outcomes based on betting market data. The chapter concludes by introducing two fundamental theorems of probability: the law of large numbers and the central limit theorem. These two theorems are widely applicable and help characterize how our estimates behave over repeated sampling as sample size increases. The final set of exercises then addresses two problems: the German cryptography machine from World War II (Enigma), and the detection of election fraud in Russia.

Chapter 7 discusses how to quantify the *uncertainty* of our estimates and predictions. In earlier chapters, we introduced various data analysis methods to find patterns in data. Building on the groundwork laid in chapter 6, chapter 7 thoroughly explains how certain we should be about such patterns. This chapter shows how to distinguish signals from noise through the computation of standard errors and confidence intervals as well as the use of hypothesis testing. In other words, the chapter concerns statistical inference. Our examples come from earlier chapters, and we focus on measuring the uncertainty of these previously computed estimates. They include the analysis of preelection polls, randomized experiments concerning the effects of class size in early education on students' performance, and an observational study assessing the effects of a minimum-wage increase on employment. When discussing statistical hypothesis tests, we also draw attention to the dangers of multiple testing and publication bias. Finally, we discuss how to quantify the level of uncertainty about the estimates derived from a linear regression model. To do this, we revisit the randomized natural experiment of female politicians in India and the regression discontinuity design for estimating the amount of wealth British politicians are able to accumulate by holding political office.

The final chapter concludes by briefly describing the next steps readers might take upon completion of this book. The chapter also discusses the role of data analysis in quantitative social science research.

1.2 How to Use this Book

In this section, we explain how to use this book, which is based on the following principle:

One can learn data analysis only by doing, not by reading.

This book is not just for reading. The emphasis must be placed on gaining experience in analyzing data. This is best accomplished by trying out the code in the book on one's own, playing with it, and working on various exercises that appear at the end of each chapter. All code and data sets used in the book are freely available for download via links from `http://press.princeton.edu/qss/`.

The book is cumulative. Later chapters assume that readers are already familiar with most of the materials covered in earlier parts. Hence, in general, it is not advisable to skip chapters. The exception is chapter 5, "Discovery," the contents of which are not used in subsequent chapters. Nevertheless, this chapter contains some of the most interesting data analysis examples of the book and readers are encouraged to study it.

The book can be used for course instruction in a variety of ways. In a traditional introductory statistics course, one can assign the book, or parts of it, as supplementary reading that provides data analysis exercises. The book is best utilized in a data analysis course where an instructor spends less time on lecturing to students and instead works interactively with students on data analysis exercises in the classroom. In such a course, the relevant portion of the book is assigned prior to each class. In the classroom, the instructor reviews new methodological and programming concepts and then applies them to one of the exercises from the book or any other similar application of their choice. Throughout this process, the instructor can discuss the exercises interactively with students, perhaps using the Socratic method, until the class collectively arrives at a solution. After such a classroom discussion, it would be ideal to follow up with a computer lab session, in which a small number of students, together with an instructor, work on another exercise.

This teaching format is consistent with the "particular general particular" principle.[1] This principle states that an instructor should first introduce a particular example to illustrate a new concept, then provide a general treatment of it, and finally apply it to another particular example. Reading assignments introduce a particular example and a general discussion of new concepts to students. Classroom discussion then allows the instructor to provide another general treatment of these concepts and then, together with students, apply them to another example. This is an effective teaching strategy that engages students with active learning and builds their ability to conduct data analysis in social science research. Finally, the instructor can assign another application as a problem set to assess whether students have mastered the materials. To facilitate this, for each chapter instructors can obtain, upon request, access to a private repository that contains additional exercises and their solutions.

In terms of the materials to cover, an example of the course outline for a 15-week-long semester is given below. We assume that there are approximately two hours of lectures and one hour of computer lab sessions each week. Having hands-on computer lab sessions with a small number of students, in which they learn how to analyze data, is essential.

Chapter title	*Chapter number*	*Weeks*
Introduction	1	1
Causality	2	2–3
Measurement	3	4–5
Prediction	4	6–7
Discovery	5	8–9
Probability	6	10–12
Uncertainty	7	13–15

For a shorter course, there are at least two ways to reduce the material. One option is to focus on aspects of data science and omit statistical inference. Specifically, from the above outline, we can remove chapter 6, "Probability," and chapter 7, "Uncertainty." An alternative approach is to skip chapter 5, "Discovery," which covers the analysis of textual, network, and spatial data, and include the chapters on probability and uncertainty.

Finally, to ensure mastery of the basic methodological and programming concepts introduced in each chapter, we recommend that users first read a chapter, practice all of the code it contains, and upon completion of each chapter, try the online review questions before attempting to solve the associated exercises. These review questions

[1] Frederick Mosteller (1980) "Classroom and platform performance." *American Statistician*, vol. 34, no. 1 (February), pp. 11–17.

Table 1.1. The **swirl** Review Exercises.

Chapter	***swirl** lesson*	*Sections covered*
1: Introduction	INTRO1	1.3
	INTRO2	1.3
2: Causality	CAUSALITY1	2.1–2.4
	CAUSALITY2	2.5–2.6
3: Measurement	MEASUREMENT1	3.1–3.4
	MEASUREMENT2	3.5–3.7
4: Prediction	PREDICTION1	4.1
	PREDICTION2	4.2
	PREDICTION3	4.3
5: Discovery	DISCOVERY1	5.1
	DISCOVERY2	5.2
	DISCOVERY3	5.3
6: Probability	PROBABILITY1	6.1–6.3
	PROBABILITY2	6.4–6.5
7: Uncertainty	UNCERTAINTY1	7.1
	UNCERTAINTY2	7.2
	UNCERTAINTY3	7.3

Note: The table shows the correspondence between the chapters and sections of the book and each set of **swirl** review exercises.

are available as **swirl** lessons via links from `http://press.princeton.edu/qss/`, and can be answered within R. Instructors are strongly encouraged to assign these **swirl** exercises prior to each class so that students learn the basics before moving on to more complicated data analysis exercises. To start the online review questions, users must first install the **swirl** package (see section 1.3.7) and then the lessons for this book using the following three lines of commands within R. Note that this installation needs to be done only once.

```r
install.packages("swirl") # install the package

library(swirl) # load the package

install_course_github("kosukeimai", "qss-swirl") # install the course
```

Table 1.1 lists the available set of **swirl** review exercises along with their corresponding chapters and sections. To start a **swirl** lesson for review questions, we can use the following command.

```r
library(swirl)
swirl()
```

More information about **swirl** is available at `http://swirlstats.com/`.

1.3 Introduction to R

This section provides a brief, self-contained introduction to R that is a prerequisite for the remainder of this book. R is an open-source statistical programming environment, which means that anyone can download it for free, examine source code, and make their own contributions. R is powerful and flexible, enabling us to handle a variety of data sets and create appealing graphics. For this reason, it is widely used in academia and industry. The *New York Times* described R as

> a popular programming language used by a growing number of data analysts inside corporations and academia. It is becoming their lingua franca. . . whether being used to set ad prices, find new drugs more quickly or fine-tune financial models. Companies as diverse as Google, Pfizer, Merck, Bank of America, the InterContinental Hotels Group and Shell use it. . . . "The great beauty of R is that you can modify it to do all sorts of things," said Hal Varian, chief economist at Google. "And you have a lot of prepackaged stuff that's already available, so you're standing on the shoulders of giants."[2]

To obtain R, visit `https://cran.r-project.org/` (The Comprehensive R Archive Network or CRAN), select the link that matches your operating system, and then follow the installation instructions.

While a powerful tool for data analysis, R's main cost from a practical viewpoint is that it must be learned as a programming language. This means that we must master various syntaxes and basic rules of computer programming. Learning computer programming is like becoming proficient in a foreign language. It requires a lot of practice and patience, and the learning process may be frustrating. Through numerous data analysis exercises, this book will teach you the basics of statistical programming, which then will allow you to conduct data analysis on your own. The core principle of the book is that we can learn data analysis only by analyzing data.

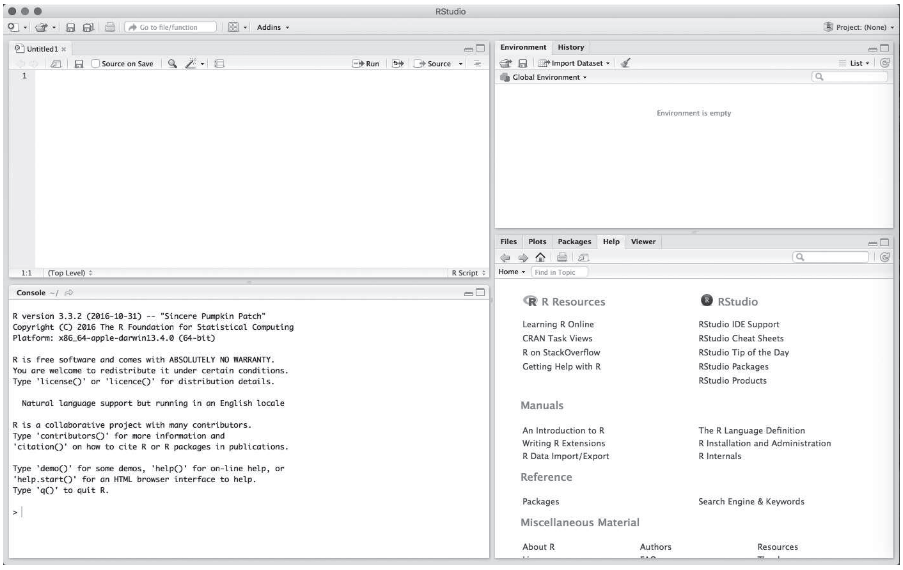
Unless you have prior programming experience (or have a preference for another text editor such as Emacs), we recommend that you use RStudio. RStudio is an open-source and free program that greatly facilitates the use of R. In one window, RStudio gives users a text editor to write programs, a graph viewer that displays the graphics we create, the R console where programs are executed, a help section, and many other features. It may look complicated at first, but RStudio can make learning how to use R much easier. To obtain RStudio, visit `http://www.rstudio.com/` and follow the download and installation instructions. Figure 1.1 shows a screenshot of RStudio.

In the remainder of this section, we cover three topics: (1) using R as a calculator, (2) creating and manipulating various objects in R, and (3) loading data sets into R.

1.3.1 ARITHMETIC OPERATIONS

We begin by using R as a calculator with standard arithmetic operators. In figure 1.1, the left-hand window of RStudio shows the R console where we can directly enter R

[2] Vance, Ashlee. 2009. "Data Analysts Captivated by R's Power." *New York Times*, January 6.



```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 1.1. Screenshot of RStudio (version 1.0.44). The upper-left window displays a script that contains code. The lower-left window shows the console where R commands can be directly entered. The upper-right window lists R objects and a history of executed R commands. Finally, the lower-right window enables us to view plots, data sets, files and subdirectories in the working directory, R packages, and help pages.

commands. In this R console, we can type in, for example, `5 + 3`, then hit Enter on our keyboard.

```
5 + 3

## [1] 8
```

R ignores spaces, and so `5+3` will return the same result. However, we added a space before and after the operator + to make it easier to read. As this example illustrates, this book displays R commands followed by the outputs they would produce if entered in the R console. These outputs begin with `##` to distinguish them from the R commands that produced them, though this mark will not appear in the R console. Finally, in this example, `[1]` indicates that the output is the first element of a *vector* of length 1 (we will discuss vectors in section 1.3.3). It is important for readers to try these examples on their own. Remember that we can learn programming only by doing! Let's try other examples.

```
5 - 3

## [1] 2

5 / 3

## [1] 1.666667

5 ^ 3
```

```
## [1] 125
5 * (10 - 3)
## [1] 35
sqrt(4)
## [1] 2
```

The final expression is an example of a so-called *function*, which takes an input (or multiple inputs) and produces an output. Here, the function `sqrt()` takes a nonnegative number and returns its square root. As discussed in section 1.3.4, R has numerous other functions, and users can even make their own functions.

1.3.2 OBJECTS

R can store information as an *object* with a name of our choice. Once we have created an object, we just refer to it by name. That is, we are using objects as "shortcuts" to some piece of information or data. For this reason, it is important to use an intuitive and informative name. The name of our object must follow certain restrictions. For example, it cannot begin with a number (but it can contain numbers). Object names also should not contain spaces. We must avoid special characters such as `%` and `$`, which have specific meanings in R. In RStudio, in the upper-right window, called `Environment` (see figure 1.1), we will see the objects we created. We use the *assignment operator* `<-` to assign some value to an object.

For example, we can store the result of the above calculation as an object named `result`, and thereafter we can access the value by referring to the object's name. By default, R will print the value of the object to the console if we just enter the object name and hit Enter. Alternatively, we can explicitly print it by using the `print()` function.

```
result <- 5 + 3
result
## [1] 8
print(result)
## [1] 8
```

Note that if we assign a different value to the same object name, then the value of the object will be changed. As a result, we must be careful not to overwrite previously assigned information that we plan to use later.

```
result <- 5 - 3
result
## [1] 2
```

Another thing to be careful about is that object names are case sensitive. For example, `Hello` is not the same as either `hello` or `HELLO`. As a consequence, we receive an error in the R console when we type `Result` rather than `result`, which is defined above.

```
Result

## Error in eval(expr, envir, enclos): object 'Result' not found
```

Encountering programming errors or bugs is part of the learning process. The tricky part is figuring out how to fix them. Here, the error message tells us that the `Result` object does not exist. We can see the list of existing objects in the `Environment` tab in the upper-right window (see figure 1.1), where we will find that the correct object is `result`. It is also possible to obtain the same list by using the `ls()` function.

So far, we have assigned only numbers to an object. But R can represent various other types of values as objects. For example, we can store a string of characters by using quotation marks.

```
kosuke <- "instructor"
kosuke

## [1] "instructor"
```

In character strings, spacing is allowed.

```
kosuke <- "instructor and author"
kosuke

## [1] "instructor and author"
```

Notice that R treats numbers like characters when we tell it to do so.

```
Result <- "5"
Result

## [1] "5"
```

However, arithmetic operations like addition and subtraction cannot be used for character strings. For example, attempting to divide or take a square root of a character string will result in an error.

```
Result / 3

## Error in Result/3: non-numeric argument to binary operator
```

```
sqrt(Result)

## Error in sqrt(Result): non-numeric argument to mathematical function
```

R recognizes different types of objects by assigning each object to a *class*. Separating objects into classes allows R to perform appropriate operations depending on the objects' class. For example, a number is stored as a numeric object whereas a character string is recognized as a character object. In RStudio, the `Environment` window will show the class of an object as well as its name. The function (which by the way is another class) `class()` tells us to which class an object belongs.

```
result

## [1] 2

class(result)

## [1] "numeric"

Result

## [1] "5"

class(Result)

## [1] "character"


class(sqrt)

## [1] "function"
```

There are many other classes in R, some of which will be introduced throughout this book. In fact, it is even possible to create our own object classes.

1.3.3 VECTORS

We present the simplest (but most inefficient) way of entering data into R. Table 1.2 contains estimates of world population (in thousands) over the past several decades. We can enter these data into R as a numeric vector object. A *vector* or a one-dimensional array simply represents a collection of information stored in a specific order. We use the function `c()`, which stands for "concatenate," to enter a data vector containing multiple values with commas separating different elements of the vector we are creating. For example, we can enter the world population estimates as elements of a single vector.

```
world.pop <- c(2525779, 3026003, 3691173, 4449049, 5320817, 6127700,
   6916183)
world.pop

## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

Table 1.2. World Population Estimates.

Year	*World population* (thousands)
1950	2,525,779
1960	3,026,003
1970	3,691,173
1980	4,449,049
1990	5,320,817
2000	6,127,700
2010	6,916,183

Source: United Nations, Department of Economic and Social Affairs, Population Division (2013). *World Population Prospects: The 2012 Revision, DVD Edition.*

We also note that the `c()` function can be used to combine multiple vectors.

```
pop.first <- c(2525779, 3026003, 3691173)
pop.second <- c(4449049, 5320817, 6127700, 6916183)
pop.all <- c(pop.first, pop.second)
pop.all

## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

To access specific elements of a vector, we use square brackets `[ ]`. This is called *indexing*. Multiple elements can be extracted via a vector of indices within square brackets. Also within square brackets the dash, `-`, removes the corresponding element from a vector. Note that none of these operations change the original vector.

```
world.pop[2]

## [1] 3026003

world.pop[c(2, 4)]

## [1] 3026003 4449049

world.pop[c(4, 2)]

## [1] 4449049 3026003

world.pop[-3]

## [1] 2525779 3026003 4449049 5320817 6127700 6916183
```

Since each element of this vector is a numeric value, we can apply arithmetic operations to it. The operations will be repeated for each element of the vector. Let's give the population estimates in millions instead of thousands by dividing each element of the vector by 1000.

```
pop.million <- world.pop / 1000
pop.million

## [1] 2525.779 3026.003 3691.173 4449.049 5320.817 6127.700
## [7] 6916.183
```

We can also express each population estimate as a proportion of the 1950 population estimate. Recall that the 1950 estimate is the first element of the vector `world.pop`.

```
pop.rate <- world.pop / world.pop[1]
pop.rate

## [1] 1.000000 1.198047 1.461400 1.761456 2.106604 2.426063
## [7] 2.738238
```

In addition, arithmetic operations can be done using multiple vectors. For example, we can calculate the percentage increase in population for each decade, defined as the increase over the decade divided by its beginning population. For example, suppose that the population was 100 thousand in one year and increased to 120 thousand in the following year. In this case, we say, "the population increased by 20%." To compute the percentage increase for each decade, we first create two vectors, one without the first decade and the other without the last decade. We then subtract the second vector from the first vector. Each element of the resulting vector equals the population increase. For example, the first element is the difference between the 1960 population estimate and the 1950 estimate.

```
pop.increase <- world.pop[-1] - world.pop[-7]
percent.increase <- (pop.increase / world.pop[-7]) * 100
percent.increase

## [1] 19.80474 21.98180 20.53212 19.59448 15.16464 12.86752
```

Finally, we can also replace the values associated with particular indices by using the usual assignment operator (`<-`). Below, we replace the first two elements of the `percent.increase` vector with their rounded values.

```
percent.increase[c(1, 2)] <- c(20, 22)
percent.increase

## [1] 20.00000 22.00000 20.53212 19.59448 15.16464 12.86752
```

1.3.4 FUNCTIONS

Functions are important objects in R and perform a wide range of tasks. A function often takes multiple input objects and returns an output object. We have already seen several functions: `sqrt()`, `print()`, `class()`, and `c()`. In R, a function generally runs as `funcname(input)` where `funcname` is the function name and `input` is the input object. In programming (and in math), we call these inputs *arguments*. For example, in the syntax `sqrt(4)`, `sqrt` is the function name and `4` is the argument or the input object.

Some basic functions useful for summarizing data include `length()` for the length of a vector or equivalently the number of elements it has, `min()` for the *minimum* value, `max()` for the *maximum* value, `range()` for the *range* of data, `mean()` for the *mean*, and `sum()` for the *sum* of the data. Right now we are inputting only one object into these functions so we will not use argument names.

```
length(world.pop)

## [1] 7

min(world.pop)

## [1] 2525779

max(world.pop)

## [1] 6916183

range(world.pop)

## [1] 2525779 6916183

mean(world.pop)

## [1] 4579529

sum(world.pop) / length(world.pop)

## [1] 4579529
```

The last expression gives another way of calculating the mean as the sum of all the elements divided by the number of elements.

When multiple arguments are given, the syntax looks like `funcname(input1, input2)`. The order of inputs matters. That is, `funcname(input1, input2)` is different from `funcname(input2, input1)`. To avoid confusion and problems stemming from the order in which we list arguments, it is also a good idea to specify the name of the argument that each input corresponds to. This looks like `funcname(arg1 = input1, arg2 = input2)`.

For example, the `seq()` function can generate a vector composed of an increasing or decreasing sequence. The first argument `from` specifies the number to start from; the second argument `to` specifies the number at which to end the sequence; the last argument `by` indicates the interval to increase or decrease by. We can create an object for the `year` variable from table 1.2 using this function.

```
year <- seq(from = 1950, to = 2010, by = 10)
year

## [1] 1950 1960 1970 1980 1990 2000 2010
```

Notice how we can switch the order of the arguments without changing the output because we have named the input objects.

```
seq(to = 2010, by = 10, from = 1950)

## [1] 1950 1960 1970 1980 1990 2000 2010
```

Although not relevant in this particular example, we can also create a decreasing sequence using the `seq()` function. In addition, the colon operator `:` creates a simple sequence, beginning with the first number specified and increasing or decreasing by 1 to the last number specified.

```
seq(from = 2010, to = 1950, by = -10)

## [1] 2010 2000 1990 1980 1970 1960 1950

2008:2012

## [1] 2008 2009 2010 2011 2012

2012:2008

## [1] 2012 2011 2010 2009 2008
```

The `names()` function can access and assign names to elements of a vector. Element names are not part of the data themselves, but are helpful attributes of the R object. Below, we see that the object `world.pop` does not yet have the names attribute, with `names(world.pop)` returning the `NULL` value. However, once we assign the `year` as the labels for the object, each element of `world.pop` is printed with an informative label.

```
names(world.pop)

## NULL

names(world.pop) <- year
names(world.pop)

## [1] "1950" "1960" "1970" "1980" "1990" "2000" "2010"
```

```
world.pop

##    1950    1960    1970    1980    1990    2000    2010
## 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

In many situations, we want to create our own functions and use them repeatedly. This allows us to avoid duplicating identical (or nearly identical) sets of code chunks, making our code more efficient and easily interpretable. The `function()` function can create a new function. The syntax takes the following form.

```
myfunction <- function(input1, input2, ..., inputN) {

    DEFINE "output" USING INPUTS

    return(output)
}
```

In this example code, `myfunction` is the function name, `input1, input2, ..., inputN` are the input arguments, and the commands within the braces `{ }` define the actual function. Finally, the `return()` function returns the output of the function. We begin with a simple example, creating a function to compute a summary of a numeric vector.

```
my.summary <- function(x){ # function takes one input
  s.out <- sum(x)
  l.out <- length(x)
  m.out <- s.out / l.out
  out <- c(s.out, l.out, m.out) # define the output
  names(out) <- c("sum", "length", "mean") # add labels
  return(out) # end function by calling output
}
z <- 1:10
my.summary(z)

##    sum length   mean
##   55.0   10.0    5.5

my.summary(world.pop)

##      sum   length     mean
## 32056704        7  4579529
```

Note that objects (e.g., `x`, `s.out`, `l.out`, `m.out`, and `out` in the above example) can be defined within a function independently of the environment in which the function is being created. This means that we need not worry about using identical names for objects inside a function and those outside it.

1.3.5 DATA FILES

So far, the only data we have used has been manually entered into R. But, most of the time, we will load data from an external file. In this book, we will use the following two data file types:

- *CSV* or comma-separated values files represent tabular data. This is conceptually similar to a spreadsheet of data values like those generated by Microsoft Excel or Google Spreadsheet. Each observation is separated by line breaks and each field within the observation is separated by a comma, a tab, or some other character or string.
- *RData* files represent a collection of R objects including data sets. These can contain multiple R objects of different kinds. They are useful for saving intermediate results from our R code as well as data files.

Before interacting with data files, we must ensure they reside in the *working directory*, which R will by default load data from and save data to. There are different ways to change the working directory. In RStudio, the default working directory is shown in the bottom-right window under the `Files` tab (see figure 1.1). Oftentimes, however, the default directory is not the directory we want to use. To change the working directory, click on `More > Set As Working Directory` after choosing the folder we want to work from. Alternatively, we can use the RStudio pull-down menu `Session > Set Working Directory > Choose Directory...` and pick the folder we want to work from. Then, we will see our files and folders in the bottom-right window.

It is also possible to change the working directory using the `setwd()` function by specifying the full path to the folder of our choice as a character string. To display the current working directory, use the function `getwd()` without providing an input. For example, the following syntax sets the working directory to `qss/INTRO` and confirms the result (we suppress the output here).

```
setwd("qss/INTRO")
getwd()
```

Suppose that the United Nations population data in table 1.2 are saved as a CSV file `UNpop.csv`, which resembles that below:

```
year, world.pop
1950, 2525779
1960, 3026003
1970, 3691173
1980, 4449049
1990, 5320817
2000, 6127700
2010, 6916183
```

In RStudio, we can read in or load CSV files by going to the drop-down menu in the upper-right window (see figure 1.1) and clicking `Import Dataset > From Text`

`File ...`. Alternatively, we can use the `read.csv()` function. The following syntax loads the data as a data frame object (more on this object below).

```
UNpop <- read.csv("UNpop.csv")
class(UNpop)

## [1] "data.frame"
```

On the other hand, if the same data set is saved as an object in an RData file named `UNpop.RData`, then we can use the `load()` function, which will load all the R objects saved in `UNpop.RData` into our R session. We do not need to use the assignment operator with the `load()` function when reading in an RData file because the R objects stored in the file already have object names.

```
load("UNpop.RData")
```

Note that R can access any file on our computer if the full location is specified. For example, we can use syntax such as `read.csv("Documents/qss/INTRO/UNpop.csv")` if the data file `UNpop.csv` is stored in the directory `Documents/qss/INTRO/`. However, setting the working directory as shown above allows us to avoid tedious typing.

A data frame object is a collection of vectors, but we can think of it like a spreadsheet. It is often useful to visually inspect data. We can view a spreadsheet-like representation of data frame objects in RStudio by double-clicking on the object name in the `Environment` tab in the upper-right window (see figure 1.1). This will open a new tab displaying the data. Alternatively, we can use the `View()` function, which as its main argument takes the name of a data frame to be examined. Useful functions for this object include `names()` to return a vector of variable names, `nrow()` to return the number of rows, `ncol()` to return the number of columns, `dim()` to combine the outputs of `ncol()` and `nrow()` into a vector, and `summary()` to produce a summary.

```
names(UNpop)

## [1] "year"      "world.pop"

nrow(UNpop)

## [1] 7

ncol(UNpop)

## [1] 2

dim(UNpop)

## [1] 7 2
```

```
summary(UNpop)

##       year          world.pop
##  Min.   :1950   Min.   :2525779
##  1st Qu.:1965   1st Qu.:3358588
##  Median :1980   Median :4449049
##  Mean   :1980   Mean   :4579529
##  3rd Qu.:1995   3rd Qu.:5724258
##  Max.   :2010   Max.   :6916183
```

Notice that the `summary()` function yields, for each variable in the data frame object, the minimum value, the first *quartile* (or 25th *percentile*), the *median* (or 50th percentile), the third quartile (or 75th percentile), and the maximum value. See section 2.6 for more discussion.

The `$` operator is one way to access an individual variable from within a data frame object. It returns a vector containing the specified variable.

```
UNpop$world.pop

## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

Another way of retrieving individual variables is to use indexing inside square brackets `[ ]`, as done for a vector. Since a data frame object is a two-dimensional array, we need two indexes, one for rows and the other for columns. Using brackets with a comma `[rows, columns]` allows users to call specific rows and columns by either row/column numbers or row/column names. If we use row/column numbers, sequencing functions covered above, i.e., `:` and `c()`, will be useful. If we do not specify a row (column) index, then the syntax will return all rows (columns). Below are some examples, demonstrating the syntax of indexing.

```
UNpop[, "world.pop"] # extract the column called "world.pop"

## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183

UNpop[c(1, 2, 3),]   # extract the first three rows (and all columns)

##   year world.pop
## 1 1950   2525779
## 2 1960   3026003
## 3 1970   3691173

UNpop[1:3, "year"]   # extract the first three rows of the "year" column

## [1] 1950 1960 1970
```

When extracting specific observations from a variable in a data frame object, we provide only one index since the variable is a vector.

```
## take elements 1, 3, 5, ... of the "world.pop" variable
UNpop$world.pop[seq(from = 1, to = nrow(UNpop), by = 2)]

## [1] 2525779 3691173 5320817 6916183
```

In R, missing values are represented by `NA`. When applied to an object with missing values, functions may or may not automatically remove those values before performing operations. We will discuss the details of handling missing values in section 3.2. Here, we note that for many functions, like `mean()`, the argument `na.rm = TRUE` will remove missing data before operations occur. In the example below, the eighth element of the vector is missing, and one cannot calculate the mean until R has been instructed to remove the missing data.

```
world.pop <- c(UNpop$world.pop, NA)
world.pop

## [1] 2525779 3026003 3691173 4449049 5320817 6127700  6916183
## [8]      NA

mean(world.pop)

## [1] NA

mean(world.pop, na.rm = TRUE)

## [1] 4579529
```

1.3.6 SAVING OBJECTS

The objects we create in an R session will be temporarily saved in the *workspace*, which is the current working environment. As mentioned earlier, the `ls()` function displays the names of all objects currently stored in the workspace. In RStudio, all objects in the workspace appear in the `Environment` tab in the upper-right corner. However, these objects will be lost once we terminate the current session. This can be avoided if we save the workspace at the end of each session as an RData file.

When we quit R, we will be asked whether we would like to save the workspace. We should answer no to this so that we get into the habit of explicitly saving only what we need. If we answer yes, then R will save the entire workspace as `.RData` in the working directory without an explicit file name and automatically load it next time we launch R. This is not recommended practice, because the `.RData` file is invisible to users of many operating systems and R will not tell us what objects are loaded unless we explicitly issue the `ls()` function.

In RStudio, we can save the workspace by clicking the `Save` icon in the upper-right `Environment` window (see figure 1.1). Alternatively, from the navigation bar, click on `Session > Save Workspace As...`, and then pick a location to save the file. Be sure to use the file extension `.RData`. To load the same workspace the next time we start RStudio, click the `Open File` icon in the upper-right `Environment` window, select `Session > Load Workspace...`, or use the `load()` function as before.

It is also possible to save the workspace using the `save.image()` function. The file extension `.RData` should always be used at the end of the file name. Unless the full path is specified, objects will be saved to the working directory. For example, the following syntax saves the workspace as `Chapter1.RData` in the `qss/INTRO` directory provided that this directory already exists.

```
save.image("qss/INTRO/Chapter1.RData")
```

Sometimes, we wish to save only a specific object (e.g., a data frame object) rather than the entire workspace. This can be done with the `save()` function as in `save(xxx, file = "yyy.RData")`, where `xxx` is the object name and `yyy.RData` is the file name. Multiple objects can be listed, and they will be stored as a single RData file. Here are some examples of syntax, in which we again assume the existence of the `qss/INTRO` directory.

```
save(UNpop, file = "Chapter1.RData")
save(world.pop, year, file = "qss/INTRO/Chapter1.RData")
```

In other cases, we may want to save a data frame object as a CSV file rather than an RData file. We can use the `write.csv()` function by specifying the object name and the file name, as the following example illustrates.

```
write.csv(UNpop, file = "UNpop.csv")
```

Finally, to access objects saved in the RData file, simply use the `load()` function as before.

```
load("Chapter1.RData")
```

1.3.7 PACKAGES

One of R's strengths is the existence of a large community of R users who contribute various functionalities as R packages. These packages are available through the Comprehensive R Archive Network (CRAN; `http://cran.r-project.org`). Throughout the book, we will employ various packages. For the purpose of illustration, suppose that we wish to load a data file produced by another statistical software package such as Stata or SPSS. The **foreign** package is useful when dealing with files from other statistical software.

To use the package, we must load it into the workspace using the `library()` function. In some cases, a package needs to be installed before being loaded. In RStudio, we can do this by clicking on `Packages > Install` in the bottom-right window (see figure 1.1), where all currently installed packages are listed, after choosing the desired packages to be installed. Alternatively, we can install from the R console using the `install.packages()` function (the output is suppressed below). Package installation needs only to occur once, though we can update the package later upon the release of a new version (by clicking `Update` or reinstalling it via the `install.packages()` function).

```
install.packages("foreign") # install package
library("foreign") # load package
```

Once the package is loaded, we can use the appropriate functions to load the data file. For example, the `read.dta()` and `read.spss()` functions can read Stata and SPSS data files, respectively (the following syntax assumes the existence of the `UNpop.dta` and `UNpop.sav` files in the working directory).

```
read.dta("UNpop.dta")
read.spss("UNpop.sav")
```

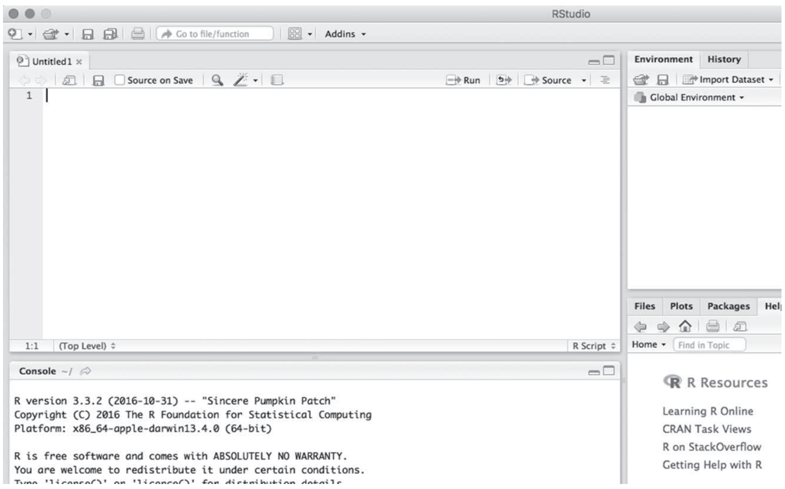
As before, it is also possible to save a data frame object as a data file that can be directly loaded into another statistical software package. For example, the `write.dta()` function will save a data frame object as a Stata data file.

```
write.dta(UNpop, file = "UNpop.dta")
```

1.3.8 PROGRAMMING AND LEARNING TIPS

We conclude this brief introduction to R by providing several practical tips for learning how to program in the R language. First, we should use a text editor like the one that comes with RStudio to write our program rather than directly typing it into the R console. If we just want to see what a command does, or quickly calculate some quantity, we can go ahead and enter it directly into the R console. However, for more involved programming, it is always better to use the text editor and save our code as a text file with the `.R` file extension. This way, we can keep a record of our program and run it again whenever necessary.

In RStudio, use the pull-down menu `File > New File > R Script` or click the `New File` icon (a white square with a green circle enclosing a white plus sign) and choose `R Script`. Either approach will open a blank document for text editing in the upper-left window where we can start writing our code (see figure 1.2). To run our code from the RStudio text editor, simply highlight the code and press the `Run` icon. Alternatively, in Windows, Ctrl+Enter works as a shortcut. The equivalent shortcut for Mac is Command+Enter. Finally, we can also run the entire code in the background



```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
```

Figure 1.2. Screenshot of the RStudio Text Editor. Once we open an R script file in RStudio, the text editor will appear as one of the windows. It can then be used to write our code.

(so, the code will not appear in the console) by clicking the `Source` icon or using the `source()` function with the code file name (including a full path if it is not placed in the working directory) as the input.

```
source("UNpop.R")
```

Second, we can annotate our R code so that it can be easily understandable to ourselves and others. This is especially important as our code gets more complex. To do this, we use the comment character #, which tells R to ignore everything that follows it. It is customary to use a double comment character ## if a comment occupies an entire line and use a single comment character # if a comment is made within a line after an R command. An example is given here.

```
##
## File: UNpop.R
## Author: Kosuke Imai
## The code loads the UN population data and saves it as a Stata file
##

library(foreign)
UNpop <- read.csv("UNpop.csv")
UNpop$world.pop <- UNpop$world.pop / 1000  # population in millions
write.dta(UNpop, file = "UNpop.dta")
```

Third, for further clarity it is important to follow a certain set of coding rules. For example, we should use informative names for files, variables, and functions. Systematic spacing and indentation are essential too. In the above examples, we place spaces around all binary operators such as <-, =, +, and -, and always add a space after a comma. While comprehensive coverage of coding style is beyond the scope of this book, we encourage you to follow a useful R style guide published by Google at `https://google.github.io/styleguide/Rguide.xml`. In addition, it is possible to check our R code for potential errors and incorrect syntax. In computer science, this process is called *linting*. The `lintr()` function in the **lintr** package enables the linting of R code. The following syntax implements the linting of the `UNpop.R` file shown above, where we replace the assignment operator <- in line 8 with the equality sign = for the sake of illustration.

```
library(lintr)
lint("UNpop.R")

## UNpop.R:8:7: style: Use <-, not =, for assignment.
## UNpop = read.csv("UNpop.csv")
##       ^
```

Finally, R Markdown via the **rmarkdown** package is useful for quickly writing documents using R. R Markdown enables us to easily embed R code and its output within a document using straightforward syntax in a plain-text format. The resulting documents can be produced in the form of HTML, PDF, or even Microsoft Word. Because R Markdown embeds R code as well as its output, the results of data analysis presented in documents are reproducible. R Markdown is also integrated into RStudio, making it possible to produce documents with a single click. For a quick start, see `http://rmarkdown.rstudio.com/`.

1.4 Summary

This chapter began with a discussion of the important role that quantitative social science research can play in today's data-rich society. To make contributions to this society through data-driven discovery, we must learn how to analyze data, interpret the results, and communicate our findings to others. To start our journey, we presented a brief introduction to R, which is a powerful programming language for data analysis. The remaining pages of this chapter are dedicated to exercises, designed to ensure that you have mastered the contents of this section. Start with the **swirl** review questions that are available via links from `http://press.princeton.edu/qss/`. If you answer these questions incorrectly, be sure to go back to the relevant sections and review the materials before moving on to the exercises.

Table 1.3. US Election Turnout Data.

Variable	*Description*
`year`	election year
`ANES`	ANES estimated turnout rate
`VEP`	voting eligible population (in thousands)
`VAP`	voting age population (in thousands)
`total`	total ballots cast for highest office (in thousands)
`felons`	total ineligible felons (in thousands)
`noncitizens`	total noncitizens (in thousands)
`overseas`	total eligible overseas voters (in thousands)
`osvoters`	total ballots counted by overseas voters (in thousands)

1.5 Exercises

1.5.1 BIAS IN SELF-REPORTED TURNOUT

Surveys are frequently used to measure political behavior such as voter turnout, but some researchers are concerned about the accuracy of self-reports. In particular, they worry about possible *social desirability bias* where, in postelection surveys, respondents who did not vote in an election lie about not having voted because they may feel that they should have voted. Is such a bias present in the American National Election Studies (ANES)? ANES is a nationwide survey that has been conducted for every election since 1948. ANES is based on face-to-face interviews with a nationally representative sample of adults. Table 1.3 displays the names and descriptions of variables in the `turnout.csv` data file.

1. Load the data into R and check the dimensions of the data. Also, obtain a summary of the data. How many observations are there? What is the range of years covered in this data set?

2. Calculate the turnout rate based on the voting age population or VAP. Note that for this data set, we must add the total number of eligible overseas voters since the `VAP` variable does not include these individuals in the count. Next, calculate the turnout rate using the voting eligible population or VEP. What difference do you observe?

3. Compute the differences between the VAP and ANES estimates of turnout rate. How big is the difference on average? What is the range of the differences? Conduct the same comparison for the VEP and ANES estimates of voter turnout. Briefly comment on the results.

4. Compare the VEP turnout rate with the ANES turnout rate separately for presidential elections and midterm elections. Note that the data set excludes the year 2006. Does the bias of the ANES estimates vary across election types?

Table 1.4. Fertility and Mortality Estimate Data.

Variable	*Description*
`country`	abbreviated country name
`period`	period during which data are collected
`age`	age group
`births`	number of births (in thousands), i.e., the number of children born to women of the age group
`deaths`	number of deaths (in thousands)
`py.men`	person-years for men (in thousands)
`py.women`	person-years for women (in thousands)

Source: United Nations, Department of Economic and Social Affairs, Population Division (2013). *World Population Prospects: The 2012 Revision, DVD Edition.*

5. Divide the data into half by election years such that you subset the data into two periods. Calculate the difference between the VEP turnout rate and the ANES turnout rate separately for each year within each period. Has the bias of ANES increased over time?

6. ANES does not interview prisoners and overseas voters. Calculate an adjustment to the 2008 VAP turnout rate. Begin by subtracting the total number of ineligible felons and noncitizens from the VAP to calculate an adjusted VAP. Next, calculate an adjusted VAP turnout rate, taking care to subtract the number of overseas ballots counted from the total ballots in 2008. Compare the adjusted VAP turnout with the unadjusted VAP, VEP, and the ANES turnout rate. Briefly discuss the results.

1.5.2 UNDERSTANDING WORLD POPULATION DYNAMICS

Understanding population dynamics is important for many areas of social science. We will calculate some basic demographic quantities of births and deaths for the world's population from two time periods: 1950 to 1955 and 2005 to 2010. We will analyze the following CSV data files: `Kenya.csv`, `Sweden.csv`, and `World.csv`. The files contain population data for Kenya, Sweden, and the world, respectively. Table 1.4 presents the names and descriptions of the variables in each data set. The data are collected for a period of 5 years where *person-year* is a measure of the time contribution of each person during the period. For example, a person who lives through the entire 5-year period contributes 5 person-years, whereas someone who lives only through the first half of the period contributes 2.5 person-years. Before you begin this exercise, it would be a good idea to directly inspect each data set. In R, this can be done with the `View()` function, which takes as its argument the name of the data frame to be examined. Alternatively, in RStudio, double-clicking a data frame in the `Environment` tab will enable you to view the data in a spreadsheet-like form.

1. We begin by computing *crude birth rate* (CBR) for a given period. The CBR is defined as

$$\text{CBR} = \frac{\text{number of births}}{\text{number of person-years lived}}.$$

   Compute the CBR for each period, separately for Kenya, Sweden, and the world. Start by computing the total person-years, recorded as a new variable within each existing data frame via the $ operator, by summing the person-years for men and women. Then, store the results as a vector of length 2 (CBRs for two periods) for each region with appropriate labels. You may wish to create your own function for the purpose of efficient programming. Briefly describe patterns you observe in the resulting CBRs.

2. The CBR is easy to understand but contains both men and women of all ages in the denominator. We next calculate the *total fertility rate* (TFR). Unlike the CBR, the TFR adjusts for age compositions in the female population. To do this, we need to first calculate the *age-specific fertility rate* (ASFR), which represents the fertility rate for women of the reproductive age range [15, 50). The ASFR for the age range $[x, x + \delta)$, where x is the starting age and δ is the width of the age range (measured in years), is defined as

$$\text{ASFR}_{[x,\ x+\delta)} = \frac{\text{number of births to women of age } [x,\ x+\delta)}{\text{number of person-years lived by women of age } [x,\ x+\delta)}.$$

   Note that square brackets, [and], include the limit whereas parentheses, (and), exclude it. For example, [20, 25) represents the age range that is greater than or equal to 20 years old and less than 25 years old. In typical demographic data, the age range δ is set to 5 years. Compute the ASFR for Sweden and Kenya as well as the entire world for each of the two periods. Store the resulting ASFRs separately for each region. What does the pattern of these ASFRs say about reproduction among women in Sweden and Kenya?

3. Using the ASFR, we can define the TFR as the average number of children that women give birth to if they live through their entire reproductive age:

$$\text{TFR} = \text{ASFR}_{[15,\ 20)} \times 5 + \text{ASFR}_{[20,\ 25)} \times 5 + \cdots + \text{ASFR}_{[45,\ 50)} \times 5.$$

   We multiply each age-specific fertility rate by 5 because the age range is 5 years. Compute the TFR for Sweden and Kenya as well as the entire world for each of the two periods. As in the previous question, continue to assume that the reproductive age range of women is [15, 50). Store the resulting two TFRs for each country or the world as vectors of length 2. In general, how has the number of women changed in the world from 1950 to 2000? What about the total number of births in the world?

4. Next, we will examine another important demographic process: death. Compute the *crude death rate* (CDR), which is a concept analogous to the CBR, for each

period and separately for each region. Store the resulting CDRs for each country and the world as vectors of length 2. The CDR is defined as

$$\text{CDR} = \frac{\text{number of deaths}}{\text{number of person-years lived}}.$$

Briefly describe the patterns you observe in the resulting CDRs.

5. One puzzling finding from the previous question is that the CDR for Kenya during the period 2005–2010 is about the same level as that for Sweden. We would expect people in developed countries like Sweden to have a lower death rate than those in developing countries like Kenya. While it is simple and easy to understand, the CDR does not take into account the age composition of a population. We therefore compute the *age-specific death rate* (ASDR). The ASDR for age range $[x, x + \delta)$ is defined as

$$\text{ASDR}_{[x,\ x+\delta)} = \frac{\text{number of deaths for people of age } [x,\ x+\delta)}{\text{number of person-years of people of age } [x,\ x+\delta)}.$$

   Calculate the ASDR for each age group, separately for Kenya and Sweden, during the period 2005–2010. Briefly describe the pattern you observe.

6. One way to understand the difference in the CDR between Kenya and Sweden is to compute the counterfactual CDR for Kenya using Sweden's population distribution (or vice versa). This can be done by applying the following alternative formula for the CDR:

$$\text{CDR} = \text{ASDR}_{[0,5)} \times P_{[0,5)} + \text{ASDR}_{[5,10)} \times P_{[5,10)} + \cdots ,$$

   where $P_{[x,x+\delta)}$ is the proportion of the population in the age range $[x, x + \delta)$. We compute this as the ratio of person-years in that age range relative to the total person-years across all age ranges. To conduct this counterfactual analysis, we use $\text{ASDR}_{[x,x+\delta)}$ from Kenya and $P_{[x,x+\delta)}$ from Sweden during the period 2005–2010. That is, first calculate the age-specific population proportions for Sweden and then use them to compute the counterfactual CDR for Kenya. How does this counterfactual CDR compare with the original CDR of Kenya? Briefly interpret the result.

Chapter 3

Measurement

Not everything that can be counted counts, and not
everything that counts can be counted.
—William Bruce Cameron, *Informal Sociology*

Measurement plays a central role in social science research. In this chapter, we first discuss survey methodology, which is perhaps the most common mode of data collection. For example, the minimum-wage study discussed in chapter 2 used a survey to measure information about employment at each fast-food restaurant. Surveys are also effective tools for making inferences about a large target population of interest from a relatively small sample of randomly selected units. In addition to surveys, we also discuss the use of latent concepts, such as ideology, that are essential for social science research. These concepts are fundamentally unobservable and must be measured using a theoretical model. Thus, issues of measurement often occupy the intersection of theoretical and empirical analyses in the study of human behavior. Finally, we introduce a basic clustering method, which enables researchers to conduct an exploratory analysis of data by discovering interesting patterns. We also learn how to plot data in various ways and compute relevant descriptive statistics in R.

3.1 Measuring Civilian Victimization during Wartime

After the September 11 attacks, the United States and its allies invaded Afghanistan with the goal of dismantling al-Qaeda, which had been operating there under the protection of the Taliban government. In 2003, the North Atlantic Treaty Organization (NATO) became involved in the conflict, sending in a coalition of international troops organized under the name of the International Security Assistance Force (ISAF). To wage this war against the Taliban insurgency, the ISAF engaged in a "hearts and minds" campaign, combining economic assistance, service delivery, and protection in order to win the support of civilians. To evaluate the success of such a campaign, it is essential to measure and understand civilians' experiences and sentiments during the war. However, measuring the experiences and opinions of civilians during wartime is a challenging task because of harsh security conditions, posing potential threats to

Table 3.1. Afghanistan Survey Data.

Variable	*Description*
`province`	province where the respondent lives
`district`	district where the respondent lives
`village.id`	ID of the village where the respondent lives
`age`	age of the respondent
`educ.years`	years of education of the respondent
`employed`	whether the respondent is employed
`income`	monthly income of the respondent (five levels)
`violent.exp.ISAF`	whether the respondent experienced violence by ISAF
`violent.exp.taliban`	whether the respondent experienced violence by the Taliban
`list.group`	randomly assigned group for the list experiment (`control`, `ISAF`, `taliban`)
`list.response`	response to the list experiment question (`0`–`4`)

interviewers and respondents. This means that respondents may inaccurately answer survey questions in order to avoid giving socially undesirable responses.

A group of social scientists conducted a public opinion *survey* in southern Afghanistan, the heartland of the insurgency.[1] The survey was administered to a sample of 2754 respondents between January and February 2011. The researchers note that the participation rate was 89%. That is, they originally contacted 3097 males and 343 of them refused to take the survey. Because local culture prohibited interviewers from talking to female citizens, the respondents were all males.

We begin by summarizing the characteristics of respondents in terms of age, years of education, employment, and monthly income in Afghani (the local currency). The CSV file `afghan.csv` contains the survey data and can be loaded via the `read.csv()` function. The names and descriptions of the variables are given in table 3.1. We use the `summary()` function to provide numerical summaries of several variables.

```
## load data
afghan <- read.csv("afghan.csv")
## summarize variables of interest
summary(afghan$age)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   15.00   22.00   30.00   32.39   40.00   80.00
```

[1] This section is based on the following two articles: Jason Lyall, Graeme Blair, and Kosuke Imai (2013) "Explaining support for combatants during wartime: A survey experiment in Afghanistan." *American Political Science Review*, vol. 107, no. 4 (November), pp. 679–705 and Graeme Blair, Kosuke Imai, and Jason Lyall (2014) "Comparing and combining list and endorsement experiments: Evidence from Afghanistan." *American Journal of Political Science*, vol. 58, no. 4 (October), pp. 1043–1063.

```
summary(afghan$educ.years)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   1.000   4.002   8.000  18.000

summary(afghan$employed)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  1.0000  0.5828  1.0000  1.0000

summary(afghan$income)

##   10,001-20,000    2,001-10,000   20,001-30,000
##             616            1420              93
## less than 2,000     over 30,000            NA's
##             457              14             154
```

We observe that the average age of the respondents is 32, a large fraction of them have very little education, and approximately 60% of the respondents are employed. Most respondents have a monthly income of less than 10,000 Afghani, which is about 200 dollars.

While civilians are often victimized during war, it is difficult to systematically measure the extent to which attacks against civilians occur. A survey measure, though it is based on self-reporting, is one possible way to quantify civilian victimization. In this survey, the interviewers asked the following question: “Over the past year, have you or anyone in your family suffered harm due to the actions of the Foreign Forces / the Taliban?” They explained to the respondents that the phrase “harm” refers to physical injury, as well as property damage. We analyze the `violent.exp.ISAF` and `violent.exp.taliban` variables, which represent whether the respondents were harmed by the ISAF and the Taliban, respectively.

```
prop.table(table(ISAF = afghan$violent.exp.ISAF,
                 Taliban = afghan$violent.exp.taliban))

##     Taliban
## ISAF         0         1
##    0 0.4953445 0.1318436
##    1 0.1769088 0.1959032
```

Using the `table()` and `prop.table()` functions, which were introduced in chapter 2, the analysis shows that over the past year, 37%(= 17.7% + 19.6%) and 33% (= 13.2% + 19.6%) of the respondents were victimized by the ISAF (second row) and the Taliban (second column), respectively. Approximately 20% of the respondents suffered from physical or property damage caused by both parties. This finding suggests that Afghan civilians were victimized (or at least they perceived that they were being victimized) by both the ISAF and the Taliban to a similar extent, rather than one warring party disproportionately harming civilians.

3.2 Handling Missing Data in R

In many surveys, researchers may encounter nonresponse because either respondents refuse to answer some questions or they simply do not know the answer. Such missing values are also common in other types of data. For example, many developing countries lack certain official statistics such as the gross domestic product (GDP) or unemployment rate. In R, missing data are coded as `NA`. For example, in the Afghanistan survey, we saw in the above analysis that 154 respondents did not provide their income. Since `NA` is a special value reserved for missing data, we can count the number of missing observations using the `is.na()` function. This function returns a logical value of `TRUE` if its argument is `NA` and yields `FALSE` otherwise.

```
## print income data for first 10 respondents
head(afghan$income, n = 10)

##  [1] 2,001-10,000  2,001-10,000  2,001-10,000  2,001-10,000
##  [5] 2,001-10,000  <NA>          10,001-20,000 2,001-10,000
##  [9] 2,001-10,000  <NA>
## 5 Levels: 10,001-20,000 2,001-10,000 ... over 30,000

## indicate whether respondents' income is missing
head(is.na(afghan$income), n = 10)

##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [10]  TRUE
```

Here, we see that the sixth and tenth respondents are not reporting their monthly income and hence are coded as `NA`. The syntax `is.na(afghan$income)` returns a vector of logical values, each indicating whether the corresponding respondent provided an answer to the income question. Thus, the sixth and tenth elements of the output from this syntax are `TRUE`. Given this function, it is now straightforward to count the total number and proportion of missing data for this variable.

```
sum(is.na(afghan$income)) # count of missing values

## [1] 154

mean(is.na(afghan$income)) # proportion missing

## [1] 0.05591866
```

Some R functions treat missing data differently from other data. For example, the `mean()` function returns `NA` when a variable contains at least one missing value. Fortunately, the `mean()` function takes an additional argument `na.rm`, which can be set to `TRUE` so that missing data are removed before the function is applied. Many other functions, including `max()`, `min()`, and `median()`, take this argument as well.

```
x <- c(1, 2, 3, NA)
mean(x)

## [1] NA

mean(x, na.rm = TRUE)

## [1] 2
```

In our data, the application of the `table()` function above ignored missing data, as if observations with missing values were not part of the data set. We can tell these functions to explicitly account for missing data. This can be done by setting the additional argument `exclude` to `NULL` so that no data including a missing value is excluded.

```
prop.table(table(ISAF = afghan$violent.exp.ISAF,
                 Taliban = afghan$violent.exp.taliban, exclude = NULL))

##       Taliban
## ISAF              0           1        <NA>
##   0     0.482933914 0.128540305 0.007988381
##   1     0.172476398 0.190994916 0.007988381
##   <NA> 0.002541757 0.002904866 0.003631082
```

We find that almost all respondents answered the victimization questions. Indeed, the nonresponse rates for these questions are less than 2%. The nonresponse rates for the Taliban and ISAF victimization questions can be obtained by adding the entries of the final column and those of the final row of the above generated table, respectively. It appears that the Afghan civilians are willing to answer questions about their experiences of violence.

Finally, the `na.omit()` function provides a straightforward way to remove all observations with at least one missing value from a data frame. The function then returns another data frame without these observations. However, we should note that this operation will result in *listwise deletion*, which eliminates an entire observation if at least one of its variables has a missing value. For example, if a respondent answers every question asked of him except for the question about income, listwise deletion would completely remove all of his information from the data, including the responses to the questions that he did answer. In our Afghanistan survey data, other variables that we have not yet discussed also have missing data. As a result, applying the `na.omit()` function to the `afghan` data frame returns a subset of the data with far fewer observations than applying the same function to the `income` variable alone.

```
afghan.sub <- na.omit(afghan) # listwise deletion
nrow(afghan.sub)

## [1] 2554
```

```
length(na.omit(afghan$income))

## [1] 2600
```

We find that the procedure of listwise deletion yields a data set of 2554 observations, whereas a total of 2600 respondents answered the income question. The difference represents the number of respondents who did answer the income question but refused to answer at least one other question in the survey.

3.3 Visualizing the Univariate Distribution

Up until now, we have been summarizing the distribution of each variable in a data set using descriptive statistics such as the mean, median, and quantiles. However, it is often helpful to visualize the distribution itself. In this section, we introduce several ways to visualize the distribution of a single variable in R. When making a figure in RStudio, you may occasionally encounter the error message "figure margins too large." We can solve this problem by increasing the size of the plots pane.

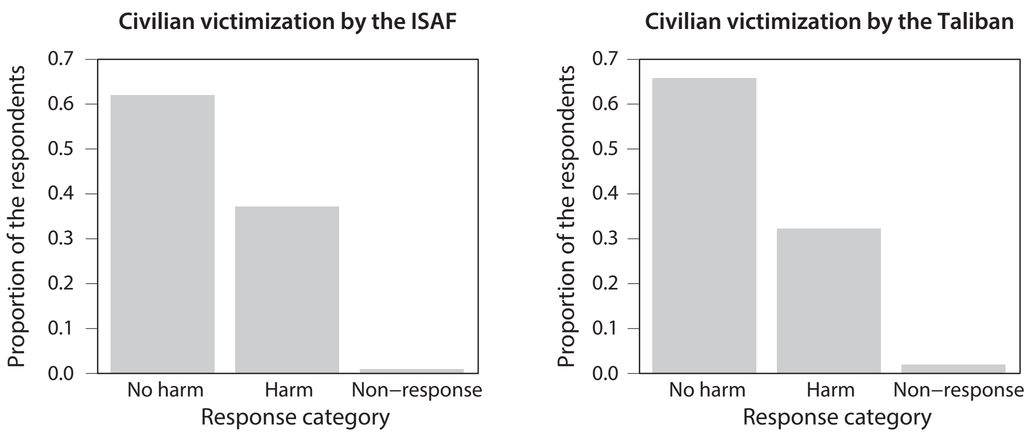
3.3.1 BAR PLOT

To summarize the distribution of a *factor variable* or *factorial variable* with several categories (see section 2.2.5), a simple table with counts or proportions, as produced above using the `table()` and `prop.table()` functions, is often sufficient. However, it is also possible to use a *bar plot* to visualize the distribution. In R, the `barplot()` function takes a vector of height and displays a bar plot in a separate graphical window. In this example, the vector of height represents the proportion of respondents in each response category.

```
## a vector of proportions to plot
ISAF.ptable <- prop.table(table(ISAF = afghan$violent.exp.ISAF,
                                exclude = NULL))
ISAF.ptable

## ISAF
##           0           1        <NA>
## 0.619462600 0.371459695 0.009077705

## make bar plots by specifying a certain range for y-axis
barplot(ISAF.ptable,
        names.arg = c("No harm", "Harm", "Nonresponse"),
        main = "Civilian victimization by the ISAF",
        xlab = "Response category",
        ylab = "Proportion of the respondents", ylim = c(0, 0.7))
## repeat the same for victimization by the Taliban
Taliban.ptable <- prop.table(table(Taliban = afghan$violent.exp.taliban,
                                   exclude = NULL))
```

```
barplot(Taliban.ptable,
        names.arg = c("No harm", "Harm", "Nonresponse"),
        main = "Civilian victimization by the Taliban",
        xlab = "Response category",
        ylab = "Proportion of the respondents", ylim = c(0, 0.7))
```

The plots in this book, including these below, may appear different from those produced by running the corresponding code in R.



We immediately see that the distributions for civilian victimization by the ISAF and the Taliban are quite similar. In addition, the nonresponse rate is equally low for both variables. Note that `names.arg` is an optional argument unique to the `barplot()` function and takes a vector of characters specifying the label for each bar. The above syntax also illustrates the use of several arguments that are common to other plot functions and are summarized here:

- `main`: a character string, i.e., a series of characters in double quotes, for the main title of the plot
- `ylab`, `xlab`: character strings for labeling the vertical axis (i.e., y-axis) and the horizontal axis (i.e., x-axis), respectively (R will automatically set these arguments to the default labels if left unspecified)
- `ylim`, `xlim`: numeric vectors of length 2 specifying the interval for the y-axis and x-axis, respectively (R will automatically set these arguments if left unspecified)

3.3.2 HISTOGRAM

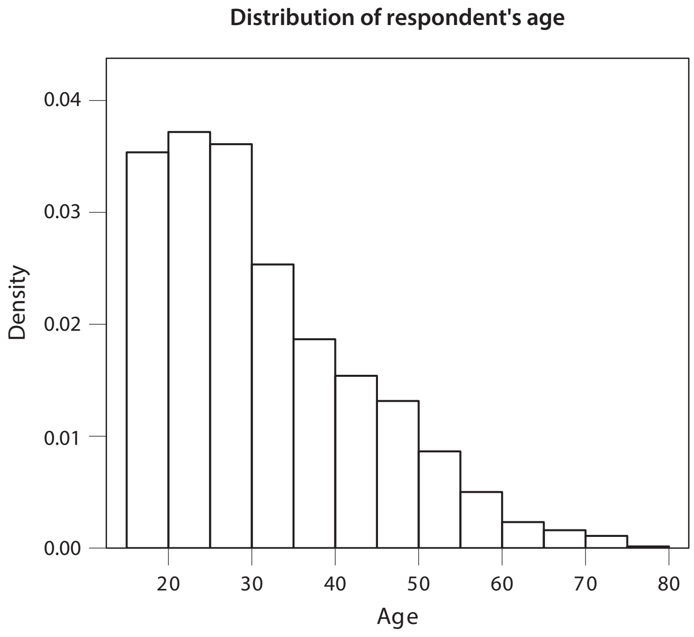
The *histogram* is a common method for visualizing the distribution of a *numeric variable* rather than a factor variable. Suppose that we would like to plot the histogram for the `age` variable in our Afghanistan survey data. To do this, we first discretize the variable by creating *bins* or intervals along the variable of interest. For example, we may use 5 years as the size of each bin for the `age` variable, which results in the intervals [15, 20), [20, 25), [25, 30), and so on. Recall from an exercise of chapter 1 (see section 1.5.2) that in mathematics square brackets, [and], include the limit, whereas parentheses, (and), exclude it. For example, [20, 25) represents the age range that is greater than or equal to 20 years old and less than 25 years old. We then count the number of observations that fall within each bin. Finally, we compute the *density* for each bin, which is the height of the bin and is defined as

$$\text{density} = \frac{\text{proportion of observations in the bin}}{\text{width of the bin}}.$$

We often care about not the exact value of each density, but rather the variable's distribution as shown by the relationship of the different bins' densities to one another within a histogram. We can therefore think of histograms as rectangular approximations of the distribution.

To create histograms in R, we use the `hist()` function and set the argument `freq` to `FALSE`. The default for this argument is `TRUE`, which plots the frequency, i.e., counts, instead of using density as the height of each bin. Using density rather than frequency is useful for comparing two distributions, because the density scale is comparable across distributions even when the number of observations is different. Below, we create histograms for the `age` variable from the Afghanistan survey data.

```
hist(afghan$age, freq = FALSE, ylim = c(0, 0.04), xlab = "Age",
     main = "Distribution of respondent's age")
```



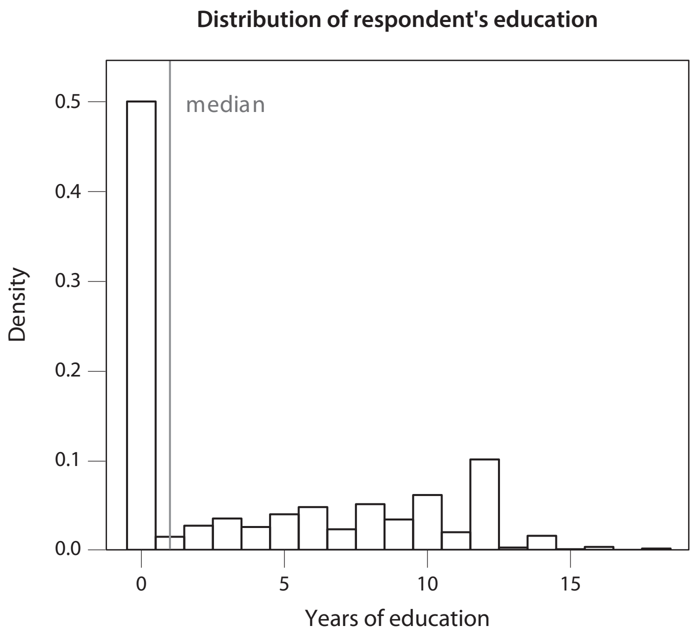
Importantly, the area of each bin in a histogram equals the proportion of observations that fall in that bin. Therefore, in general, we interpret the density scale, the unit of the vertical axis, as percentage per horizontal unit. In the age example, the density is measured as percentage per year. This implies that density is not a proportion and hence the height of each bin can exceed 1. On the other hand, the area of each bin represents the percentage of observations it contains, so the areas of all bins sum to 1. In this way, histograms visualize how observations are distributed across the different values of the variable of interest. The age distribution for the survey respondents is skewed towards the left, suggesting that a larger number of young males were interviewed.

A histogram divides the data into bins where the area of each bin represents the proportion of observations that fall within the bin. The height of each bin represents **density**, which is equal to the proportion of observations within each bin divided by the width of the bin. A histogram approximates the distribution of a variable.

Our next histogram features the years of education variable, `educ.years`. Instead of letting R automatically choose the width of bins, as we did for the age variable, we now specify exactly how the bins are created using $[-0.5, 0.5), [0.5, 1.5), [1.5, 2.5), \ldots,$ to center each bin around each of the integer values, i.e., $0, 1, 2, \ldots,$ corresponding to the observed values. The height of each bin then represents the proportion of observations that received the corresponding number of years of education. We implement this by specifying a vector of the breakpoints between histogram bins with the `breaks` argument. In this case, the default specification, which we will get by leaving the argument unspecified, is $[0, 1), [1, 2), [2, 3), \ldots$ where it is centered around $0.5, 1.5, 2.5, \ldots,$ failing to correspond to the observed values. Note that the `breaks` argument can take other forms of input to manipulate the histogram. For example, it also accepts a single integer specifying the number of bins for the histogram.

```
## histogram of education.  use "breaks" to choose bins
hist(afghan$educ.years, freq = FALSE,
     breaks = seq(from = -0.5, to = 18.5, by = 1),
     xlab = "Years of education",
     main = "Distribution of respondent's education")
## add a text label at (x, y) = (3, 0.5)
text(x = 3, y = 0.5, "median")
## add a vertical line representing median
abline(v = median(afghan$educ.years))
```

The histogram for the years of education variable clearly shows that the education level of these respondents is extremely low. Indeed, almost half of them have never attended school. We also add a vertical line and a text label indicating the *median* value, using the `abline()` and `text()` functions, respectively. Both of these functions add a layer to any existing plot, and this is why they are used after the `hist()` function in the above example. The `text(x,y,z)` function adds character text `z` centered at



the points specified by the coordinate vectors, (`x`, `y`). The `abline()` function can add a straight line to an existing plot in the following three ways:

- `abline(h = x)` to place a horizontal line at height `x`
- `abline(v = x)` to place a vertical line at point `x`
- `abline(a = y, b = s)` to place a line with intercept `y` and slope `s`

A more general function to plot a line is `lines()`. This function takes two arguments, `x` and `y`. These two arguments must be vectors with the same number of x-coordinates and y-coordinates respectively. The function will then draw line segments connecting the point denoted by the first coordinate in argument `x` and the first coordinate in argument `y`, to the point denoted by the second coordinates in each argument, to the point denoted by the third coordinates in each argument, and so on. For example, we can draw the median line as done above using this function instead.

```
## adding a vertical line representing the median
lines(x = rep(median(afghan$educ.years), 2), y = c(0,0.5))
```

In this example, we want to create a vertical line at the x value for the median of `afghan$educ.years`. We use y values, 0 and 0.5, so that the line will extend between the bottom and top limits of the histogram respectively. We then need x-coordinates equal to the median of `afghan$educ.years` to correspond with each y-coordinate. To do this easily, we can use the `rep()` function, whose first argument takes the value we want to repeat and whose second argument takes the number of repetitions, which is the length of the resulting vector. The above `rep()` function creates a vector of length 2 with the median of `afghan$educ.years` as each element in that vector. Thus, a line goes from point $(x, y) = (1, 0)$ to point $(x, y) = (1, 0.5)$, since the median year of education is 1.

It is also possible to add points to any existing plot using the `points()` function. Specifically, in `points(x, y)`, two vectors—`x` and `y`—specify the coordinates of points to be plotted. Finally, R has various functionalities that enable users to choose different colors, line types, and other aesthetic choices. Some commonly used arguments are given below, but the details about each function can be obtained on their manual pages:

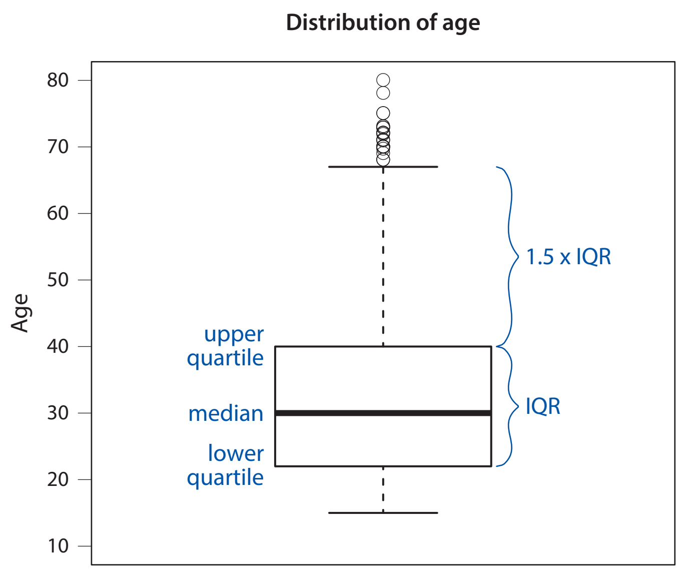
- `col` specifies the color to use, such as `"blue"` and `"red"`. This argument can be used in many functions including `text()`, `abline()`, `lines()`, and `points()`. Type `colors()` to see all the built-in color names R has (see section 5.3.3 for more details).
- `lty` specifies the type of line to be drawn, using either a character or a numeric value, including `"solid"` or `1` (default) for solid lines, `"dashed"` or `2` for dashed lines, `"dotted"` or `3` for dotted lines, `"dotdash"` or `4` for dotted and dashed lines, and `"longdash"` or `5` for long dashed lines. This argument can be used in many functions that produce lines, including `abline()` and `lines()`.
- `lwd` specifies the thickness of lines where `lwd = 1` is the default value. This argument can be used in many functions that produce lines, including `abline()` and `lines()`.

3.3.3 BOX PLOT

The *box plot* represents another way to visualize the distributions of a numeric variable. It is particularly useful when comparing the distribution of several variables by placing them side by side. A box plot visualizes the median, the quartiles, and the IQR all together as a single object. To make box plots in R, we use the `boxplot()` function by simply giving a variable of interest as an input. Again, we use the `age` variable as an example.

```
## commands for plotting curly braces and text in blue are omitted
boxplot(afghan$age, main = "Distribution of age", ylab = "Age",
        ylim = c(10, 80))
```

As illustrated below, the box contains 50% of the data ranging from the lower quartile (25th percentile) to the upper quartile (75th percentile) with the solid horizontal line indicating the median value (50th percentile). Then, dotted vertical lines, each of which has its end indicated by a short horizontal line called a “whisker,” extend below and above the box. These two dotted lines represent the data that are contained within 1.5 IQR below the lower quartile and above the upper quartile, respectively. Furthermore, the observations that fall outside 1.5 IQR from the upper

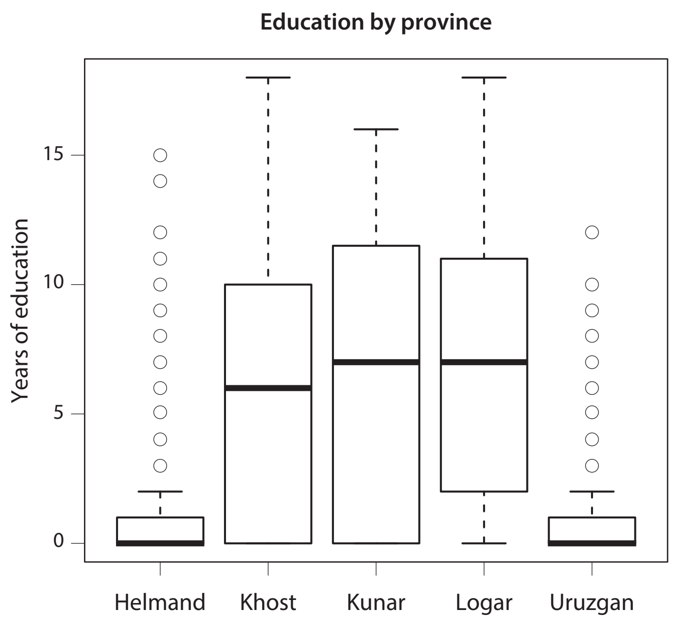


and lower quartiles are indicated by open circles. In this plot, the section of the dotted line extending from the top of the box to the horizontal line represents 1.5 IQR. If the minimum (maximum) value is contained within 1.5 IQR below the lower quartile (above the upper quartile), the dotted line will end at the minimum (maximum) value. The absence of open circles below the horizontal line implies that the minimum value of this variable is indeed within the 1.5 IQR of the lower quantile.

If we wish to visualize the distribution of a single variable, then a histogram is often more informative than a box plot because the former shows the full shape of the distribution. One of the main advantages of a box plot is that it allows us to compare multiple distributions in a more compact manner than histograms, as the next example shows. Using the `boxplot()` function, we can create a box plot for a different group of observations where the groups are defined by a factor variable. This is done by using the formula in R, which takes the form `y ~ x`. In the current context, `boxplot(y ~ x, data = d)` creates box plots for variable `y` for different groups defined by a factor `x` where the variables, `x` and `y`, are taken from the data frame `d`. As an illustration, we plot the distribution of the years of education variable by province.

```
boxplot(educ.years ~ province, data = afghan,
        main = "Education by province", ylab = "Years of education")
```

We find that the education level in Helmand and Uruzgan provinces is much lower than that of the other three provinces. It also turns out that civilians in these two provinces report harm inflicted by both parties more than those who live in the other provinces. This is shown below by computing the proportion of affirmative answers to the corresponding question, for each province.



```
tapply(afghan$violent.exp.taliban, afghan$province, mean, na.rm = TRUE)

##    Helmand      Khost      Kunar      Logar    Uruzgan
## 0.50422195 0.23322684 0.30303030 0.08024691 0.45454545

tapply(afghan$violent.exp.ISAF, afghan$province, mean, na.rm = TRUE)

##   Helmand     Khost     Kunar     Logar   Uruzgan
## 0.5410226 0.2424242 0.3989899 0.1440329 0.4960422
```

Note that the syntax, `na.rm = TRUE`, is passed to the `mean()` function within the `tapply()` function so that missing observations are deleted when computing the mean for each province (see section 3.2).

A **box plot** visualizes the distribution of a variable by indicating its median, lower and upper quartiles, and the points outside the 1.5 interquartile range from the lower and upper quartiles. It enables the comparison of distributions across multiple variables in a compact manner.

3.3.4 PRINTING AND SAVING GRAPHS

There are a few ways to print and save the graphs you create in R. The easiest way is to use the menus in RStudio. In RStudio, each time you create a graph using any of the R plotting functions, a new tab will open in the bottom-right window. To save an image of the plot, click `Export` and then either `Save Plot as Image` or `Save Plot as PDF`.

You can also save or print a graph with a command by using the `pdf()` function to open the PDF device before your plotting commands and then the `dev.off()` function afterwards to close the device. For example, the following syntax saves the box plots we just created above as a PDF file `educ.pdf` in the working directory. The `pdf()` function can specify the height and width of the graphics region in inches.

```
pdf(file = "educ.pdf", height = 5, width = 5)
boxplot(educ.years ~ province, data = afghan,
        main = "Education by province", ylab = "Years of education")
dev.off()
```

In many cases, we want to compare multiple plots by printing them next to each other in a single figure file. To do this, we use the function `par()` as `par(mfrow = c(X, Y))` before we start making plots. This will create an `X` by `Y` grid of "subplots" (`mfrow` stands for multiple figures in rows). Our multiple plots will fill in this grid, row by row. To fill the grid column by column, you can, instead, use the syntax `par(mfcol = c(X, Y))`. Note that the `par()` function also takes many other arguments that allow users to control graphics in R. For example, the `cex` argument changes the size of a character or symbol, with `cex = 1` as the default value. We can set the `cex` argument to a value greater than 1 (e.g., `par(cex = 1.2)`) in order to enlarge the fonts in displayed graphics. Note that it is also possible to separately specify the size for different parts of a plot using `cex.main` (main plot title), `cex.lab` (axis title labels), and `cex.axis` (axis value labels). Executing the following code chunk all at once creates the two histograms we made earlier in this chapter and saves them side by side in a single PDF file.

```
pdf(file = "hist.pdf", height = 4, width = 8)
## one row with 2 plots with font size 0.8
par(mfrow = c(1, 2), cex = 0.8)
## for simplicity omit the text and lines from the earlier example
hist(afghan$age, freq = FALSE,
     xlab = "Age", ylim = c(0, 0.04),
     main = "Distribution of respondent's age")
hist(afghan$educ.years, freq = FALSE,
     breaks = seq(from = -0.5, to = 18.5, by = 1),
     xlab = "Years of education", xlim = c(0, 20),
     main = "Distribution of respondent's education")
dev.off()
```

3.4 Survey Sampling

Survey sampling is one of the main data collection methods in quantitative social science research. It is often used to study public opinion and behavior when such information is not available from other sources such as administrative records. Survey sampling is a process in which researchers select a subset of the population, called a sample, to understand the features of a target population. It should be distinguished from a *census*, for which the goal is to enumerate all members of the population.

What makes survey sampling remarkable is that one can learn about a fairly large population by interviewing a small fraction of it. In the Afghanistan data, a sample of 2754 respondents was used to infer the experiences and attitudes of approximately 15 million civilians. In the United States, a sample of just about 1000 respondents is typically used to infer the public opinion of more than 200 million adult citizens. In this section, we explain what makes this seemingly impossible task possible and discuss important methodological issues when collecting and analyzing survey data.

3.4.1 THE ROLE OF RANDOMIZATION

As in the randomized control trials (RCTs) discussed in chapter 2, randomization plays an essential role in survey sampling. We focus on a class of sampling procedures called *probability sampling* in which every unit of a target population has a known nonzero probability of being selected. Consider the most basic probability sampling procedure, called *simple random sampling* (SRS), which selects the predetermined number of respondents to be interviewed from a target population, with each potential respondent having an equal chance of being selected. The sampling is done *without replacement* rather than *with replacement* so that once individuals are selected for interview they are taken out of the *sampling frame*, which represents the complete list of potential respondents. Therefore, sampling without replacement assigns at most one interview per individual.

SRS produces a sample of respondents that are *representative* of the population. By "representative," we mean that if we repeat the procedure many times, the features of each resulting sample would not be exactly the same as those of the population, but on average (across all the samples) would be identical. For example, while one may happen to obtain, due to random chance, a sample of individuals who are slightly older than those of the population, the age distribution over repeated samples would resemble that of the population. Moreover, as in RCTs, probability sampling guarantees that the characteristics of the sample, whether observed or unobserved, are on average identical to the corresponding characteristics of the population. For this reason, we can infer population characteristics using those of a representative sample obtained through probability sampling procedures (see chapter 7 for more details).

Before probability sampling was invented, researchers often used a procedure called *quota sampling*. Under this alternative sampling strategy, we specify fixed quotas of certain respondents to be interviewed such that the resulting sample characteristics resemble those of the population. For example, if 20% of the population has a college degree, then researchers will set the maximum number of college graduates who will be selected for interview to be 20% of the sample size. They will stop interviewing those with college degrees once they reach that quota. The quota can be defined using multiple variables. Often, the basic demographics such as age, gender, education, and race are used to construct the categories for which the quota is specified. For example, we may interview black females with a college degree and between 30 and 40 years old, up to 5% of the sample size.

The problem of quota sampling is similar to that of the observational studies discussed in chapter 2. Even if a sample is representative of the population in terms of some observed characteristics, which are used to define quotas, its unobserved features may be quite different from those of the population. Just as individuals may self-select to receive a treatment in an observational study, researchers may inadvertently interview individuals who have characteristics systematically different from those who are not interviewed. Probability sampling eliminates this potential *sample selection bias* by making sure that the resulting sample is representative of the target population.

Simple random sampling (SRS) is the most basic form of **probability sampling**, which avoids **sample selection bias** by randomly choosing units from a population. Under SRS, the predetermined number of units is randomly selected from a target population without replacement, where each unit has an equal probability of being selected. The resulting sample is representative of the population in terms of any observed and unobserved characteristics.

Quota sampling is believed to have caused one of the most well-known errors in the history of newspapers. In the 1948 US presidential election, most major preelection polls, including those conducted by Gallup and Roper, used quota sampling and predicted that Thomas Dewey, then the governor of New York, would decisively defeat Harry Truman, the incumbent, on Election Day. On election night, the *Chicago Tribune* went ahead and sent the next morning's newspaper to press, with the erroneous headline "Dewey defeats Truman," even before many East Coast states reported their polling results. The election result, however, was the exact opposite. Truman won by a margin of 5 percentage points in the national vote. Figure 3.1 shows a well-known picture of Truman happily holding a copy of the *Chicago Tribune* with the erroneous headline.

In order to apply SRS, we need a list of all individuals in the population to sample from. As noted earlier, such a list is called a *sampling frame*. In practice, given a target population, obtaining a sampling frame that enumerates all members of the population is not necessarily straightforward. Lists of phone numbers, residential addresses, and email addresses are often incomplete, missing a certain subset of the population who have different characteristics. *Random digit dialing* is a popular technique for phone surveys. However, the procedure may suffer from sample selection bias since some people may not have a phone number and others may have multiple phone numbers.

Most in-person surveys employ a complex sampling procedure due to logistical challenges. While an in-depth study of various survey sampling strategies is beyond the scope of this book, we briefly discuss how the Afghanistan survey was conducted in order to illustrate how survey sampling is done in practice. For the Afghanistan survey, the researchers used a *multistage cluster sampling* procedure. In countries like Afghanistan, it is difficult to obtain a sampling frame that contains most, let alone all, of their citizens. However, comprehensive lists of administrative units such as districts and villages are often readily available. In addition, since sending interviewers across a large number of distant areas may be too costly, it is often necessary to sample respondents within a reasonable number of subregions.

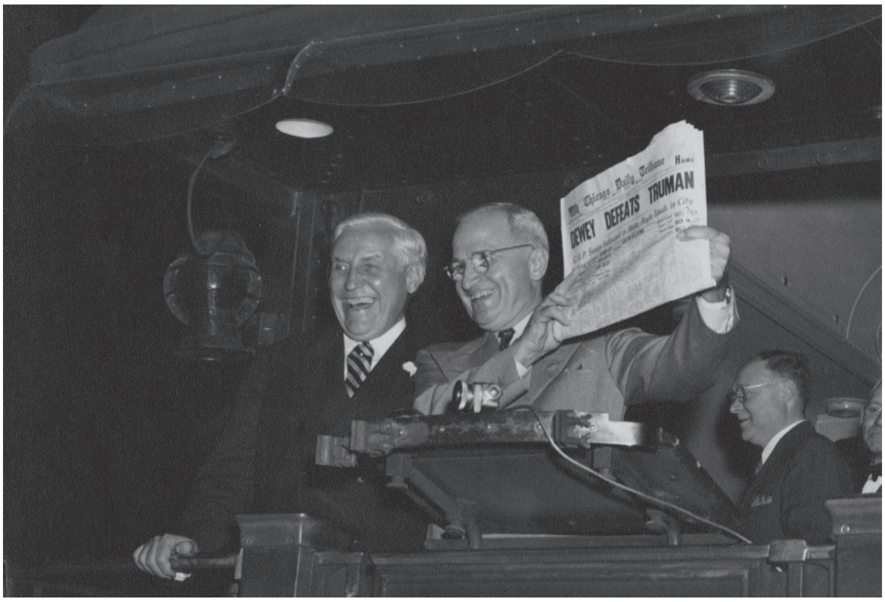


Figure 3.1. Harry Truman, the Winner of the 1948 US Presidential Election, Holding a Copy of the *Chicago Tribune* with the Erroneous Headline. Source: Copyright unknown, Courtesy of Harry S. Truman Library.

Table 3.2. Afghanistan Village Data.

Variable	*Description*
`village.surveyed`	whether a village is sampled for survey
`altitude`	altitude of the village
`population`	population of the village

The multistage cluster sampling method proceeds in multiple stages by sampling larger units first and then randomly selecting smaller units within each of the selected larger units. In the Afghanistan survey, within each of the five provinces of interest, the researchers sampled districts and then villages within each selected district. Within each sampled village, interviewers selected a household in an approximately random manner based on their location within the village, and finally administered a survey to a male respondent aged 16 years or older, who was sampled using the *Kish grid* method. While the probability of selecting each individual in the population is known only approximately, the method in theory should provide a roughly representative sample of the target population.

We examine the representativeness of the randomly sampled villages in the Afghanistan data. The data file `afghan-village.csv` contains the altitude and population of each village (see table 3.2 for the names and descriptions of the variables). For the population variable, it is customary to take the *logarithmic transformation* so that the distribution does not look too skewed with a small number of extremely large or small values. The logarithm of a positive number x is defined as the exponent of a base value b, i.e., $y = \log_b x \iff x = b^y$. For example, if the base value is 10, then the

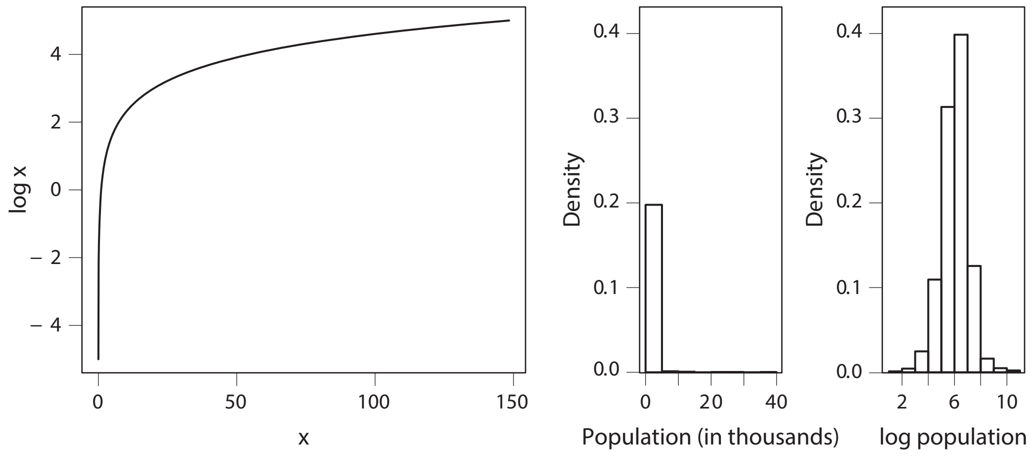


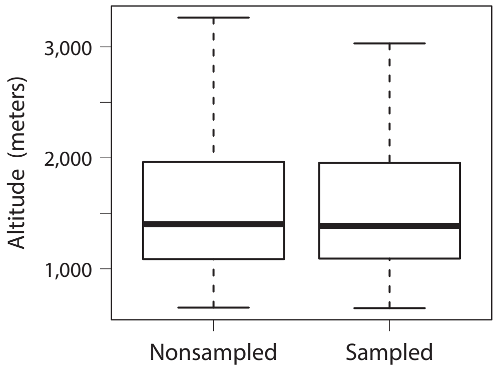
Figure 3.2. The Natural Logarithm. The left plot shows the natural logarithm $\log_e x$ where x is a positive number and e = 2.7182... is Euler's number. The remaining plots display the histograms for the population of Afghan villages on the original scale (in thousands) and the natural logarithmic scale. The population distribution is skewed without the logarithmic transformation.

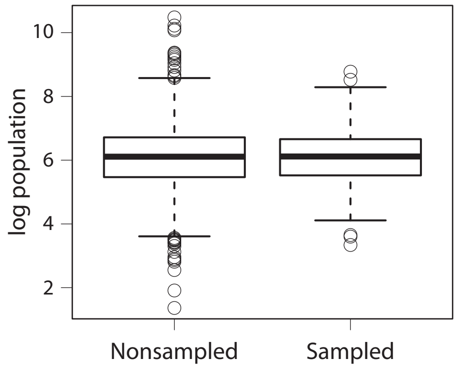
logarithm of 1000 is $3 = \log_{10} 1000$. Similarly, the logarithm of 0.01 is $-2 = \log_{10} 0.01$. The *natural logarithm* uses as its base value an important mathematical constant $e = 2.7182\ldots$, which is defined as the limit of $(1 + 1/n)^n$ as n approaches infinity and is sometimes called Euler's number, so that $y = \log_e x \iff x = e^y$. The left-hand plot of figure 3.2 depicts the natural logarithm function graphically. The figure also shows that in the Afghanistan data, without the logarithmic transformation, the distribution of the population is quite skewed because there exist a large number of small villages and a small number of large villages.

The **natural logarithmic transformation** is often used to correct the skewness of variables such as income and population that have a small number of observations with extremely large or small positive values. The natural logarithm is the logarithm with base e, which is a mathematical constant approximately equal to 2.7182, and defined as $y = \log_e x$. It is the inverse function of the exponential function, so $x = e^y$.

We use box plots to compare the distribution of these variables across sampled and nonsampled villages. The variable `village.surveyed` indicates whether each village in the data is (randomly) sampled and surveyed; `1` indicates yes and `0` no. As explained above, we take the natural logarithmic transformation for the population variable using the `log()` function. By default R uses e as its base, though it is possible to specify a different base using the `base` argument in this function. Note that the exponential function in R is given by `exp()`. In the `boxplot()` function, we can use the `names` argument to specify a character vector of labels for each group.

```
## load village data
afghan.village <- read.csv("afghan-village.csv")
## box plots for altitude
boxplot(altitude ~ village.surveyed, data = afghan.village,
        ylab = "Altitude (meters)",  names = c("Nonsampled", "Sampled"))
## box plots for log population
boxplot(log(population) ~ village.surveyed, data = afghan.village,
        ylab = "log population", names = c("Nonsampled", "Sampled"))
```





The result shows that although there are some outliers, the distribution of these two variables is largely similar between the sampled and nonsampled villages. So, at least for these variables the sample appears to be representative of the population.

3.4.2 NONRESPONSE AND OTHER SOURCES OF BIAS

While probability sampling has attractive theoretical properties, in practice conducting a survey faces many obstacles. As mentioned earlier, a *sampling frame*, which enumerates all members of a target population, is difficult to obtain. In many cases, we end up sampling from a list that may systematically diverge from the target population in terms of some important characteristics. Even if a representative sampling frame is available, interviewing randomly selected individuals may not be straightforward. Failure to reach selected units is called *unit nonresponse*. For example, many individuals refuse to participate in phone surveys. In the Afghanistan survey, the authors report that 2754 out of 3097 potential respondents agreed to participate in the survey, resulting in an 11% refusal rate. If those to whom researchers fail to administer the survey are systematically different from those who participate in the survey, then bias due to unit nonresponse arises.

In addition to unit nonresponse, most surveys also encounter the *item nonresponse* problem when respondents refuse to answer certain survey questions. For example, we saw in section 3.2 that in the Afghanistan survey, the income variable had a nonresponse rate of approximately 5%. If those who refuse to answer are systematically different from those who answer, then the resulting inference based only on the observed responses may be biased. In the Afghanistan data, for example, the item nonresponse rates for the questions about civilian victimization by the Taliban and the ISAF appear to vary across provinces.

```
tapply(is.na(afghan$violent.exp.taliban), afghan$province, mean)

##     Helmand       Khost       Kunar       Logar     Uruzgan
## 0.030409357 0.006349206 0.000000000 0.000000000 0.062015504

tapply(is.na(afghan$violent.exp.ISAF), afghan$province, mean)

##     Helmand       Khost       Kunar       Logar     Uruzgan
## 0.016374269 0.004761905 0.000000000 0.000000000 0.020671835
```

We observe that in Helmand and Uruzgan, which are known to be the most violent provinces (see section 3.3.3), the item nonresponse rates are the highest. These differences are especially large for the question about civilian victimization by the Taliban. The evidence presented here suggests that although the item nonresponse rate in this survey is relatively low, certain systematic factors appear to affect its magnitude. While they are beyond the scope of this book, there exist many statistical methods of reducing the bias due to unit and item nonresponse.

There are two types of nonresponse in survey research. **Unit nonresponse** refers to a case in which a potential respondent refuses to participate in a survey. **Item nonresponse** occurs when a respondent who agreed to participate refuses to answer a particular question. Both nonresponses can result in biased inferences if those who respond to a question are systematically different from those who do not.

Beyond item and unit nonresponse, another potential source of bias is *misreporting*. Respondents may simply lie because they may not want interviewers to find out their true answers. In particular, *social desirability bias* refers to the problem where respondents choose an answer that is seen as socially desirable regardless of what their truthful answer is. For example, it is well known that in advanced democracies voters tend to report they participated in an election even when they actually did not, because abstention is socially undesirable. Similarly, social desirability bias makes it difficult to accurately measure sensitive behavior and opinions such as corruption, illegal behavior, racial prejudice, and sexual activity. For this reason, some scholars remain skeptical of self-reports as measurement for social science research.

One main goal of the Afghanistan study was to measure the extent to which Afghan citizens support foreign forces. To defeat local insurgent forces and win the wars in Afghanistan and Iraq, many Western policy makers believed that “winning the hearts and minds” of a civilian population was essential. Unfortunately, directly asking whether citizens are supportive of foreign forces and insurgents in rural Afghan villages can put interviewers and respondents at risk because interviews are often conducted in public. The *Institutional Review Board*, which evaluates the ethical issues and potential risks of research projects involving human subjects, may not approve direct questioning of sensitive questions in a civil war setting. Even if possible, direct questioning may lead to nonresponse and misreporting.

To address this problem, the authors of the original study implemented a survey methodology called *item count technique* or *list experiment*. The idea is to use aggregation to provide a certain level of anonymity to respondents. The method first randomly divides the sample into two comparable groups. In the "control" group, the following question was asked.

> I'm going to read you a list with the names of different groups and individuals on it. After I read the entire list, I'd like you to tell me how many of these groups and individuals you broadly support, meaning that you generally agree with the goals and policies of the group or individual. Please don't tell me which ones you generally agree with; only tell me how many groups or individuals you broadly support.
>
> Karzai Government; National Solidarity Program; Local Farmers

The "treatment" group received the same question except with an additional sensitive item:

> Karzai Government; National Solidarity Program; Local Farmers; Foreign Forces

Here, the last item, Foreign Forces, which refers to the ISAF, is the sensitive item. The item count technique does not require respondents to answer each item separately. Instead, they give an aggregate count of items. Since the two conditions are comparable apart from the sensitive item, the difference in the average number of items a respondent reports will be an estimate of the proportion of those who support the ISAF. The `list.group` variable indicates which group each respondent is randomly assigned to, where for the two relevant groups the variable equals `ISAF` and `control`. The outcome variable is `list.response`, which represents the item count reported by each respondent.

```
mean(afghan$list.response[afghan$list.group == "ISAF"])-
    mean(afghan$list.response[afghan$list.group == "control"])
## [1] 0.04901961
```

The item count technique estimates that approximately 5% of Afghan citizens support the ISAF, implying that the ISAF is unpopular among Afghans.

The weakness of the item count technique, however, is that in the "treatment" group, answering either "0" or "4" in this case reveals one's honest answer. These potential problems are called *floor effects* and *ceiling effects*, respectively. In the Afghan data, we see clear evidence of this problem when the Taliban, instead of the ISAF, is added to the list as the sensitive item.

```
table(response = afghan$list.response, group = afghan$list.group)

##          group
## response control ISAF taliban
##        0     188  174       0
##        1     265  278     433
##        2     265  260     287
##        3     200  182     198
##        4       0   24       0
```

Remarkably, no respondents in the `taliban` group answered either “0” or “4,” perhaps because they do not want to be identified as either supportive or critical of the Taliban.

As we can see, measuring the truthful responses to sensitive questions is a challenging task. In addition to the item count technique, social scientists have used a variety of survey methodologies in an effort to overcome this problem. Another popular methodology is called the *randomized response technique* in which researchers use randomization to provide anonymity to respondents. For example, respondents are asked to roll a six-sided die in private without revealing the outcome. They are then asked to answer yes if the outcome of rolling the die was 1, no if 6, and give an honest answer if the outcome was between 2 and 5. Therefore, unlike the item count technique, the secrecy of individual responses is completely protected. Since the probability of each outcome is known, the researchers can estimate the aggregate proportion of honest responses out of those who responded with a yes answer even though they have no way of knowing the truthfulness of individual answers with certainty.

3.5 Measuring Political Polarization

Social scientists often devise *measurement models* to summarize and understand the behaviors, attitudes, and unobservable characteristics of human beings. A prominent example is the question of how to quantitatively characterize the *ideology* of political actors such as legislators and judges from their behavior. Of course, we do not directly observe the extent to which an individual is liberal or conservative. While ideology is perhaps a purely artificial concept, it is nonetheless a useful way to describe the political orientation of various individuals. Over the past several decades, social scientists have attempted to infer the ideology of politicians from their roll call votes. In each year, for example, legislators in the US Congress vote on hundreds of bills. Using this voting record, which is publicly available, researchers have tried to characterize the political ideology of each member of Congress and how the overall ideological orientation in the US Congress has changed over time.[2]

A simple measurement model of *spatial voting* can relate a legislator’s ideology to their votes. Figure 3.3 illustrates this model, which characterizes the ideology

[2] This section is based on Nolan McCarty, Keith T. Poole, and Howard Rosenthal (2006) *Polarized America: The Dance of Ideology and Unequal Riches*. MIT Press.

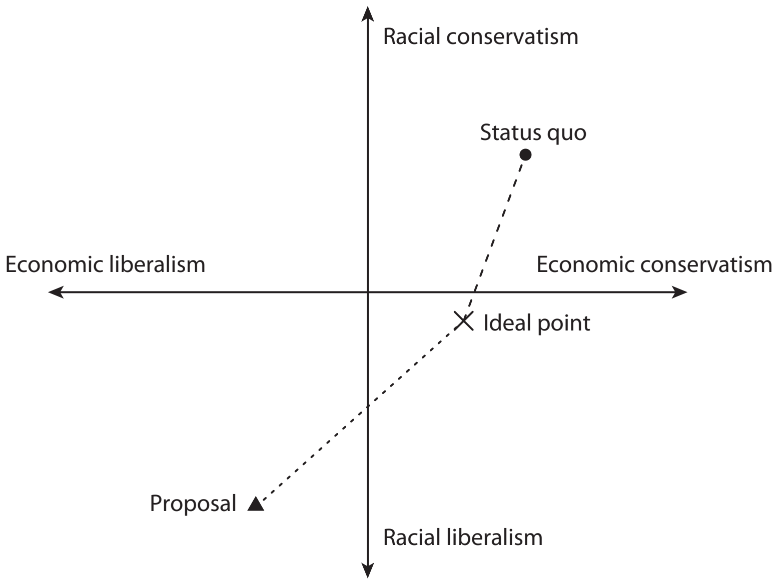


Figure 3.3. An Illustration for the Spatial Voting Model of Legislative Ideology.

or “ideal point” of legislators by two dimensions—economic and racial liberalism/conservatism—identified by researchers as the main ideological characteristics of postwar congressional politics. Researchers have found that much of congressional roll call voting can be explained by the economic liberalism/conservatism dimension while the racial liberalism/conservatism dimension is less pronounced. Under this model, the legislator, whose ideal point is indicated by a cross mark in the figure, is more likely to vote against the proposal (solid triangle) whenever their ideal point is closer to the status quo (solid circle) than to the proposal location. The outcomes of congressional votes on controversial proposals reveal much about legislators’ ideologies. On the other hand, a unanimously accepted or rejected proposal provides no information about legislators’ ideological orientations.

A similar model is used in educational testing literature. Scholars have developed a class of statistical methods called *item response theory* for standardized tests such as the SAT and Graduate Record Examination (GRE). In this context, legislators and legislative proposals are replaced with student examinees and exam questions. Instead of ideal points, the goal is to measure students’ abilities. The model also estimates the difficulty of each question. This helps the researchers choose good exam questions, which are neither too difficult nor too easy, so that only competent students will be able to provide a correct answer. These examples illustrate the importance of latent (i.e., unobserved) measurements in social science research.

3.6 Summarizing Bivariate Relationships

In this section, we introduce several ways to summarize the relationship between two variables. We analyze the estimates of legislators’ ideal points, known as *DW-NOMINATE scores*, where more negative (positive) scores are increasingly liberal

Table 3.3. Legislative Ideal Points Data.

Variable	*Description*
`name`	name of the congressional representative
`state`	state of the congressional representative
`district`	district number of the congressional representative
`party`	party of the congressional representative
`congress`	congressional session number
`dwnom1`	DW-NOMINATE score (first dimension)
`dwnom2`	DW-NOMINATE score (second dimension)

(conservative). The CSV file `congress.csv` contains the estimated ideal points of all legislators who served in the House of Representatives from the 80th (1947–1948) to the 112th (2011–2012) Congresses. Table 3.3 presents the names and descriptions of the variables in the data set.

3.6.1 SCATTER PLOT

Using the `plot()` function, we create a *scatter plot*, which plots one variable against another in order to visualize their relationship. The syntax for this function is `plot(x, y)`, where `x` and `y` are vectors of horizontal and vertical coordinates, respectively. Here, we plot the DW-NOMINATE first dimension score (`dwnom1` variable) on the horizontal axis, which represents economic liberalism/conservatism, against its second dimension score on the vertical axis (`dwnom2` variable), which represents racial liberalism/conservatism. We will start by creating scatter plots for the 80th and 112th Congresses. We begin by subsetting the relevant part of the data.

```
congress <- read.csv("congress.csv")
## subset the data by party
rep <- subset(congress, subset = (party == "Republican"))
dem <- congress[congress$party == "Democrat", ] # another way to subset
## 80th and 112th Congress
rep80 <- subset(rep, subset = (congress == 80))
dem80 <- subset(dem, subset = (congress == 80))
rep112 <- subset(rep, subset = (congress == 112))
dem112 <- subset(dem, subset = (congress == 112))
```

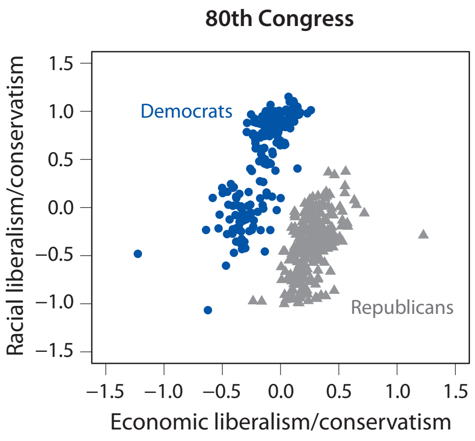
We will be creating multiple scatter plots with the same set of axis labels and axis limits. To avoid repetition, we store them as objects for later use.

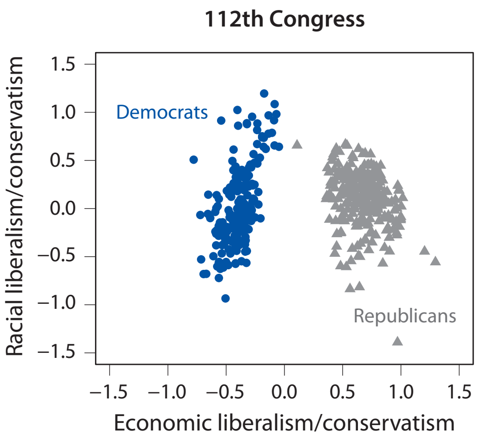
```
## preparing the labels and axis limits to avoid repetition
xlab <- "Economic liberalism/conservatism"
ylab <- "Racial liberalism/conservatism"
lim <- c(-1.5, 1.5)
```

Finally, using this axis information, we create scatter plots of ideal points for the 80th and 112th Congresses. Note that the `pch` argument in the `plot()` and `points()` functions can be used to specify different plotting symbols for the two parties. In the current example, `pch = 16` graphs solid triangles for Republicans while `pch = 17` graphs solid circles for Democrats. More options are available and can be viewed by typing `example(points)` into R console.

```
## scatter plot for the 80th Congress
plot(dem80$dwnom1, dem80$dwnom2, pch = 16, col = "blue",
     xlim = lim, ylim = lim, xlab = xlab, ylab = ylab,
     main = "80th Congress") # Democrats
points(rep80$dwnom1, rep80$dwnom2, pch = 17, col = "red") # Republicans
text(-0.75, 1, "Democrats")
text(1, -1, "Republicans")
## scatter plot for the 112th Congress
plot(dem112$dwnom1, dem112$dwnom2, pch = 16, col = "blue",
     xlim = lim, ylim = lim, xlab = xlab, ylab = ylab,
     main = "112th Congress")
points(rep112$dwnom1, rep112$dwnom2, pch = 17, col="red")
```

The plots below use solid gray triangles instead of red triangles for Republicans. See page C1 for the full-color version.





The plots show that in the 112th Congress (as opposed to the 80th Congress), the racial liberalism/conservatism dimension is no longer important in explaining the ideological difference between Democrats and Republicans. Instead, the economic dimension appears to be a dominant explanation for the partisan difference, and the difference between Democrats and Republicans in the racial dimension is much less pronounced.

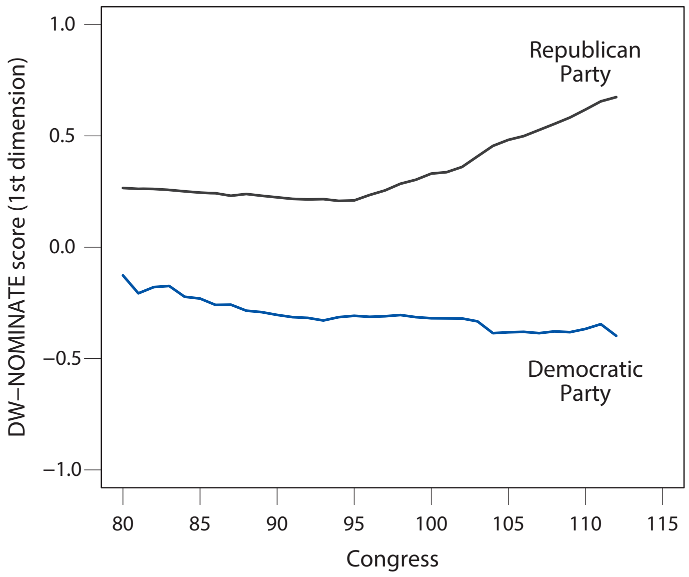
Next, we compute the median legislator, based on the DW-NOMINATE first dimension score, separately for the Democratic and Republican Parties and for each Congress. These party median ideal points represent the center of each party in the economic liberalism/conservatism dimension. We can do this easily by using the `tapply()` function.

```r
## party median for each congress
dem.median <- tapply(dem$dwnom1, dem$congress, median)
rep.median <- tapply(rep$dwnom1, rep$congress, median)
```

Finally, using the `plot()` function, we create a *time-series plot* where each party median is displayed for each Congress. We set the `type` argument to `"l"` in order to draw a line connecting the median points over time. This plot enables us to visualize how the party medians have changed over time. We will use the term of Congress as the horizontal axis. This information is available as the `name` of the `dem.median` vector.

```r
## Democrats
plot(names(dem.median), dem.median, col = "blue", type = "l",
     xlim = c(80, 115), ylim = c(-1, 1), xlab = "Congress",
     ylab = "DW-NOMINATE score (first dimension)")
## add Republicans
lines(names(rep.median), rep.median, col = "red")
text(110, -0.6, "Democratic\n Party")
text(110, 0.85, "Republican\n Party")
```

The plot below uses a gray line instead of a red line for Republicans. See page C1 for the full-color version.



Note that the syntax `\n` used in the `text()` function indicates a change to a new line. The plot clearly shows that the ideological centers of the two parties diverge over

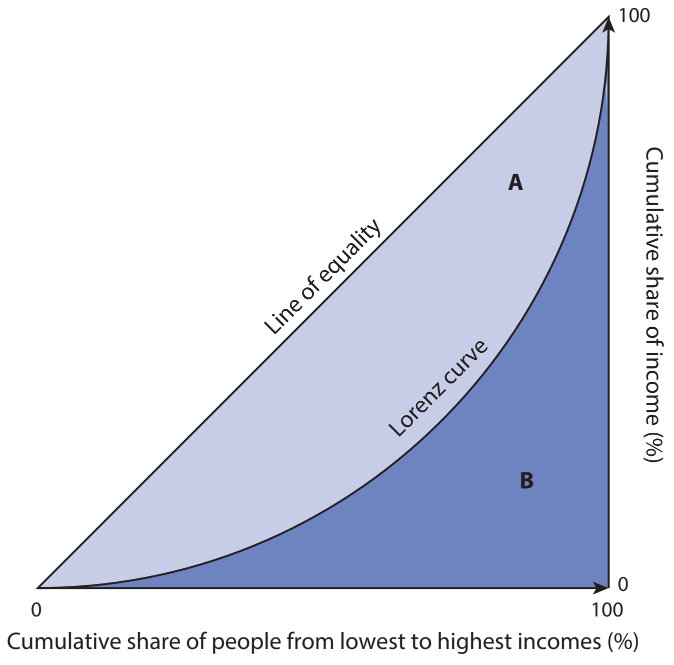

100
A
Line of equality
Lorenz curve
B
0
0
100
Cumulative share of income (%)
Cumulative share of people from lowest to highest incomes (%)

Figure 3.4. Gini Coefficient and Lorenz Curve.

time. The Democratic Party has become more liberal while the Republican Party has increasingly moved in a conservative direction in recent years. Many scholars refer to this phenomenon as *political polarization*.

A **scatter plot** graphically compares two variables measured on the same set of units by plotting the value of one variable against that of the other for each unit.

3.6.2 CORRELATION

What is the cause of political polarization? This is a difficult question to answer, and is the subject of much scholarly debate. However, it has been pointed out that rising income inequality may be responsible for the widening partisan gap. To measure income inequality, we use the *Gini coefficient* (*Gini index*), which is best understood graphically. Figure 3.4 illustrates the idea. The horizontal axis represents the cumulative share of people sorted from the lowest to highest income. The vertical axis, on the other hand, plots the cumulative share of income held by those whose income is equal to or less than that of a person at a given income percentile. The *Lorenz curve* connects these two statistics. If everyone earns exactly the same income, then the Lorenz curve will be the same as the 45-degree line because x% of the population will hold exactly x% of national income regardless of the value of x. Let's call this the line of equality. However, if low income people earn a lot less than high income people, the Lorenz curve will become flatter at the beginning and then sharply increase at the end.

Table 3.4. US Gini Coefficient Data.

Variable	*Description*
`year`	year
`gini`	US Gini coefficient

Now, we can define the Gini coefficient as the area between the line of equality and the Lorenz curve divided by the area under the line of equality. In terms of figure 3.4,

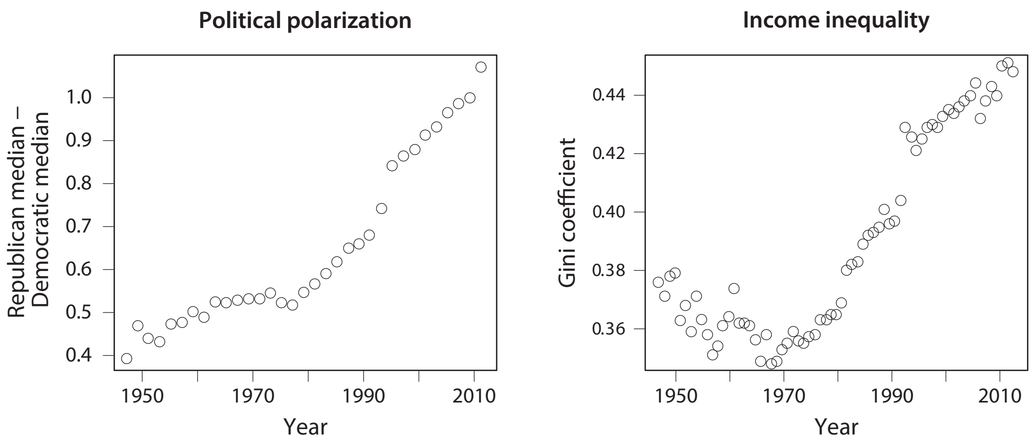
$$\begin{aligned}\text{Gini coefficient} &= \frac{\text{area between the line of equality and the Lorenz curve}}{\text{area under the line of equality}}\\ &= \frac{\text{area A in figure 3.4}}{\text{area A + area B in figure 3.4}}.\end{aligned}$$

The formula implies that the larger (smaller) area A is, the higher (lower) the Gini coefficient, meaning more (less) inequality. In a perfectly equal society, the Gini coefficient is 0. In contrast, a society where one person possesses all the wealth has a Gini coefficient of 1.

The **Gini coefficient** (Gini index) measures the degree of income equality and inequality in a given society. It ranges from 0 (everyone has the same amount of wealth) to 1 (one person possesses all the wealth).

To examine the relationship between political polarization and income inequality, we create two time-series plots side by side. The first plot shows the partisan gap, i.e., the difference between the two party medians, over time. The second time-series plot displays the Gini coefficient during the same time period. The CSV data file, `USGini.csv`, contains the Gini coefficient from 1947 to 2013 (see table 3.4). We notice that both political polarization and income inequality have been steadily increasing in the United States.

```r
## Gini coefficient data
gini <- read.csv("USGini.csv")
## time-series plot for partisan difference
plot(seq(from = 1947.5, to = 2011.5, by = 2),
     rep.median - dem.median, xlab = "Year",
     ylab = "Republican median -\n Democratic median",
     main = "Political polarization")
## time-series plot for Gini coefficient
plot(gini$year, gini$gini, ylim = c(0.35, 0.45), xlab = "Year",
     ylab = "Gini coefficient", main = "Income inequality")
```



However, in chapter 2, we learned that *association does not necessarily imply causation* and hence we should not necessarily interpret this upwards trend as evidence for income inequality causing polarization. For example, life expectancy has also constantly increased during this time period, and yet this does not imply that longer life expectancy caused political polarization or vice versa.

Correlation (also referred to as a *correlation coefficient*) is one of the most frequently used statistics to summarize bivariate relationships. The measure represents how, on average, two variables move together relative to their respective means. Before defining correlation, we need to introduce the *z-score*, which represents the number of standard deviations an observation is above or below the mean. Specifically, the z-score of the ith observation of variable x is defined as

$$z\text{-score of } x_i = \frac{x_i - \text{mean of } x}{\text{standard deviation of } x}. \tag{3.1}$$

For example, if the z-score of a particular observation equals 1.5, the observation is 1.5 standard deviations above the mean. The z-score standardizes a variable so its unit of measurement no longer matters. More formally, the z-score of $ax_i + b$, where a and b are constants (a is non-zero), is identical to the z-score of x_i. Simple algebra can show this property:

$$\begin{aligned} z\text{-score of } (ax_i + b) &= \frac{(ax_i + b) - \text{mean of } (ax + b)}{\text{standard deviation of } (ax + b)} \\ &= \frac{a \times (x_i - \text{mean of } x)}{a \times \text{standard deviation of } x} \\ &= z\text{-score of } x_i, \end{aligned}$$

where the first equality follows from the definition of z-score in equation (3.1) and the second equality is based on the definitions of mean and standard deviation (see equation (2.4)). The constant b can be dropped in the above equations because its mean equals b itself.

The z-**score** of the ith observation of a variable x measures the number of standard deviations an observation is above or below the mean. It is defined as

$$z\text{-score of } x_i = \frac{x_i - \bar{x}}{S_x},$$

where $\bar{x}$ and S_x are the mean and standard deviation of x, respectively. The z-score, as a measure of deviation from the mean, is not sensitive to how the variable is scaled and/or shifted.

Now, we can define the correlation between two variables x and y, measured for the same set of n observations, as the average products of z-scores for the two variables:

$$\text{correlation}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(z\text{-score of } x_i \times z\text{-score of } y_i). \tag{3.2}$$

As in the case of standard deviation (see section 2.6.2), the denominator of the correlation is often $n - 1$ rather than n. However, this difference should not affect one's conclusion so long as the sample size is sufficiently large. Within the summation, each z-score measures the deviation of the corresponding observation from its mean in terms of standard deviation. Suppose that when one variable is above its mean, the other variable is also likely to be greater than its own mean. Then, the correlation is likely to be positive because the signs of the standardized units tend to agree with each other. On the other hand, suppose that when one variable is above its mean, the other variable is likely to be less than its own mean. Then, the correlation is likely to be negative. In the current example, a positive correlation means that in years when income inequality is above its over-time mean, political polarization is also likely to be higher than its over-time mean.

Recall that z-scores are not sensitive to what units are used to measure a variable. Because it is based on z-scores, correlation also remains identical even if different units are used for measurement. For example, the correlation does not change even if one measures income in thousands of dollars instead of dollars. Indeed, one can even use a different currency. This is convenient because, for example, the relationship between income and education should not change depending on what scales we use to measure income. As another consequence of standardization, correlation varies only between -1 and 1. This allows us to compare the strengths and weaknesses of association between different pairs of variables.

Correlation (correlation coefficient) measures the degree to which two variables are associated with each other. It is defined as

$$\text{correlation of } x \text{ and } y = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i-\bar{x}}{S_x}\times\frac{y_i-\bar{y}}{S_y}\right)$$

$$\text{or} \quad \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i-\bar{x}}{S_x}\times\frac{y_i-\bar{y}}{S_y}\right),$$

where $\bar{x}$ and $\bar{y}$ are the means and S_x and S_y are the standard deviations for variables x and y, respectively. Correlation ranges from -1 to 1 and is not sensitive to how a variable is scaled and/or shifted.

In R, the correlation can be calculated using the `cor()` function. For example, we can now calculate the correlation between the Gini coefficient and the measure of political polarization. To do this, since each US congressional session lasts two years, we take the Gini coefficient for the second year of each session.

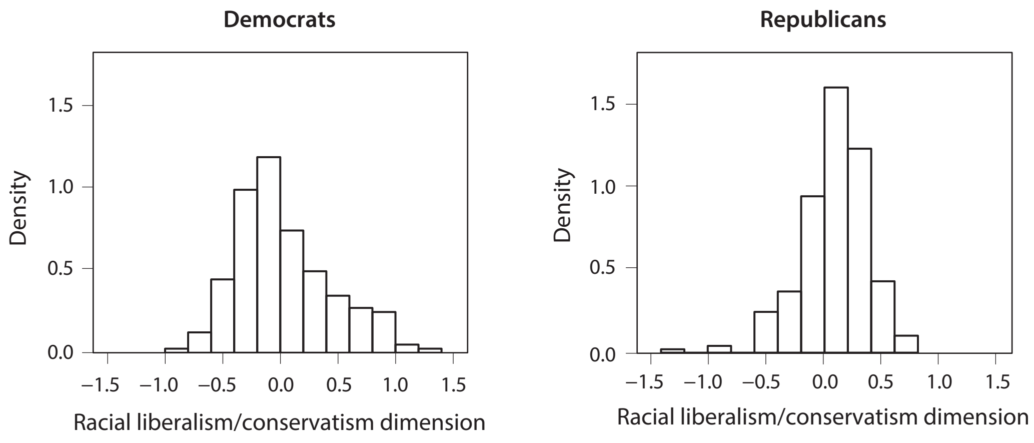
```
cor(gini$gini[seq(from = 2, to = nrow(gini), by = 2)],
    rep.median - dem.median)

## [1] 0.9418128
```

We find that the correlation is positive and quite high, indicating that political polarization and income inequality move in a similar direction. As we have already emphasized, this correlation alone does not imply causality. Many variables have an upwards trend during this time, leading to a high positive correlation among them.

3.6.3 QUANTILE–QUANTILE PLOT

Finally, in some cases, we are interested in comparing the entire distributions of two variables rather than just the mean or median. One way to conduct such a comparison is to simply plot two histograms side-by-side. As an example, we compare the distribution of ideal points on the racial liberalism/conservatism dimension in the 112th Congress. When comparing across multiple plots, it is important to use the same scales for the horizontal and vertical axes for all plots to facilitate the comparison.

```
hist(dem112$dwnom2, freq = FALSE, main = "Democrats",
     xlim = c(-1.5, 1.5), ylim = c(0, 1.75),
     xlab = "Racial liberalism/conservatism dimension")
hist(rep112$dwnom2, freq = FALSE, main = "Republicans",
     xlim = c(-1.5, 1.5), ylim = c(0, 1.75),
     xlab = "Racial liberalism/conservatism dimension")
```

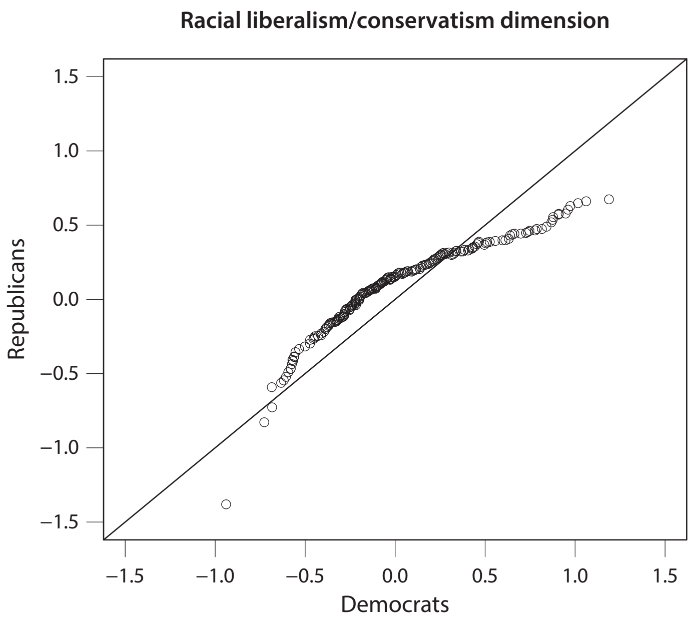


We observe that the two distributions are similar, though the distribution for Democrats appears to have a longer upper tail (i.e., the distribution extends further to the right) than that for Republicans. In addition, the Republicans' ideological positions seem to have a greater concentration towards the center than those of the Democrats.

A more direct way of comparing two distributions is a *quantile–quantile plot* or *Q–Q plot*. The Q–Q plot is based on *quantiles*, defined in section 2.6.1. It is a scatter plot of quantiles where each point represents the same quantile. For example, the median, upper quartile, and lower quartile of one sample will be plotted against the corresponding quantiles of the other sample. If two distributions are identical, then all quantiles have the same values. In this case, the Q–Q plot will result in the 45-degree line. Points above the 45-degree line indicate that a variable plotted on the vertical axis has a greater value at the corresponding quantile than a variable on the horizontal axis. In contrast, points below a 45-degree line imply the opposite relationship. This implies, for example, that if all points are above the 45-degree line, the variable on the vertical axis takes a greater value in every quantile than the variable on the horizontal axis.

Another useful feature of the Q–Q plot is that we can check the relative dispersion of two distributions. If the points in a Q–Q plot form a flatter line than the 45-degree line, they indicate that the distribution plotted on the horizontal axis is more dispersed than that on the vertical axis. In contrast, if the line has a steeper slope than 45 degrees, then the distribution plotted on the vertical line has a greater spread. The `qqplot()` function generates this plot by specifying the arguments `x` and `y`.

```
qqplot(dem112$dwnom2, rep112$dwnom2, xlab = "Democrats",
       ylab = "Republicans", xlim = c(-1.5, 1.5), ylim = c(-1.5, 1.5),
       main = "Racial liberalism/conservatism dimension")
abline(0, 1) # 45-degree line
```



In this Q–Q plot, the horizontal and vertical axes represent the racial dimension for Democrats and Republicans, respectively. The fact that the points representing lower quantiles appear above the 45-degree line indicate that liberal Republicans are more conservative than liberal Democrats. This is because these quantiles have greater values (i.e., more conservative) for Republicans than the corresponding quantiles for Democrats. In contrast, the points representing upper quantiles are located below the 45-degree line. That is, at the highest quantiles, i.e., the conservative ones, the Democrats score higher and so more conservatively than the Republicans. Thus, conservative Democrats are more conservative than conservative Republicans. Conservative Republicans would be more conservative than conservative Democrats if all the points for the upper quantiles were above the 45-degree line. Finally, the line connecting the points is flatter than the 45-degree line, indicating that the distribution of ideological positions is more dispersed for Democrats than for Republicans.

The **quantile–quantile plot** or Q–Q plot is a scatter plot of quantiles. It plots the value of each quantile for one variable against the value of the corresponding quantile for another variable. If the distributions of the two variables are identical, all points of the Q–Q plot lie on the 45-degree line. If the points form a line whose slope is steeper than 45 degrees, the distribution plotted on the vertical axis is more dispersed than the distribution on the horizontal axis. If the slope is less than 45 degrees, then the distribution on the vertical axis has less dispersion.

3.7 Clustering

In the previous analysis, the scatter plot made it visually clear that the 112th Congress had two ideologically distinct groups, Democrats and Republicans. But, are there any *clusters* of ideologically similar legislators within each party? Is there a well-defined procedure that can uncover groups of similar observations? We consider one of the most basic *clustering algorithms*, called *k-means*. Before we describe the k-means algorithm, we briefly introduce two new important R objects: *matrix* and *list*. These objects will be used when we implement the k-means algorithm in R.

3.7.1 MATRIX IN R

Although both the matrix and data frame objects are rectangular arrays and have many similarities, there are critical differences. Most importantly, a data frame can take different types of variables (e.g., numeric, factor, character) whereas a matrix in principle takes only numeric values (though it also can accommodate logical and other special values under certain circumstances). While one can extract variables from a data frame object using the `$` operator, in general the entries of a matrix need to be extracted by using square brackets `[,]` whose first and second elements, separated by a comma, indicate the rows and columns of interest, respectively. Although we do not exploit it in this book, a matrix is useful for *linear algebra* operations and is generally more computationally efficient than a data frame.

To create a matrix object, we can use the `matrix()` function by specifying the size of the matrix via the `nrow` (number of rows) and `ncol` (number of columns) arguments and indicating whether the matrix should be filled with the input data by row (`byrow = TRUE`) or column (`byrow = FALSE`). Moreover, adding labels to rows and columns can be done by the `rownames()` and `colnames()` functions.

```
## 3x4 matrix filled by row; first argument takes actual entries
x <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
rownames(x) <- c("a", "b", "c")
colnames(x) <- c("d", "e", "f", "g")
dim(x) # dimension

## [1] 3 4

x

##   d  e  f  g
## a 1  2  3  4
## b 5  6  7  8
## c 9 10 11 12
```

If one coerces a data frame object into a matrix using the `as.matrix()` function, some features of the data frame object, such as variable types, will get lost. In the following example, we illustrate the fact that a data frame can take different data types such as character and numeric, but a matrix cannot accommodate them. Instead, the `as.matrix()` function converts variables of different types to a single type, `character` in this case.

```
## data frame can take different data types
y <- data.frame(y1 = as.factor(c("a", "b", "c")), y2 = c(0.1, 0.2, 0.3))
class(y$y1)

## [1] "factor"

class(y$y2)

## [1] "numeric"

## as.matrix() converts both variables to character
z <- as.matrix(y)
z

##      y1  y2
## [1,] "a" "0.1"
## [2,] "b" "0.2"
## [3,] "c" "0.3"
```

Finally, some useful operations on a matrix include `colSums()` and `colMeans()`, which calculate the column sums and means, respectively. The same operations can be applied to rows via the `rowSums()` and `rowMeans()` functions.

```
## column sums
colSums(x)

##  d  e  f  g
## 15 18 21 24

## row means
rowMeans(x)

##    a    b    c
##  2.5  6.5 10.5
```

More generally, we can use the `apply()` function to apply any function to a margin, meaning a row or a column, of a matrix. This function takes three main arguments: the first or `X` argument is a matrix, the second or `MARGIN` argument specifies a dimension over which we wish to apply a function (`1` represents rows while `2` represents columns), and the third or `FUN` argument names a function. We provide three examples. The first two examples are equivalent to the `colSums()` and `rowMeans()` shown above. The last example computes the standard deviation of each row.

```
## column sums
apply(x, 2, sum)

##  d  e  f  g
## 15 18 21 24

## row means
apply(x, 1, mean)

##    a    b    c
##  2.5  6.5 10.5

## standard deviation for each row
apply(x, 1, sd)

##        a        b        c
## 1.290994 1.290994 1.290994
```

3.7.2 LIST IN R

We now turn to another important object class in R, called a list. The list object is useful because it can store different types of objects as its elements. For example, a list can take numeric and character vectors of different lengths. In contrast, a data frame assumes those vectors to be of the same length. In fact, a list can even contain multiple data frames of different sizes as its elements. Therefore, a list is a very general class of objects.

Each element of a list comes with a name and can be extracted using the `$` operator (just like a variable in a data frame). It is also possible to extract an element using double square brackets, `[[ ]]`, with an integer or its element name indicating the element to be extracted. Below is a simple illustrative example of a list, which contains an integer vector of length 10 (`y1`), a character vector of length 3 (`y2`), and a data frame with two variables and three observations (`y3`). To create a list, we use the `list()` function and specify its elements by using their names as arguments.

```
## create a list
x <- list(y1 = 1:10, y2 = c("hi", "hello", "hey"),
          y3 = data.frame(z1 = 1:3, z2 = c("good", "bad", "ugly")))
## three ways of extracting elements from a list
x$y1 # first element

##  [1]  1  2  3  4  5  6  7  8  9 10

x[[2]] # second element

## [1] "hi"    "hello" "hey"
```

```
x[["y3"]] # third element

##   z1   z2
## 1  1 good
## 2  2  bad
## 3  3 ugly
```

Some of the functions we introduced can be applied to the list object. They include the `names()` (to extract the names of elements) and `length()` (to obtain the number of elements) functions.

```
names(x)  # names of all elements

## [1] "y1" "y2" "y3"

length(x) # number of elements

## [1] 3
```

3.7.3 THE *k*-MEANS ALGORITHM

Now that we are familiar with matrices and lists, we can use them to apply the k-means algorithm. The k-means algorithm is an *iterative algorithm* in which a set of operations are repeatedly performed until a noticeable difference in results is no longer produced. The goal of the algorithm is to split the data into k similar groups where each group is associated with its *centroid*, which is equal to the within-group mean. This is done by first assigning each observation to its closest cluster and then computing the centroid of each cluster based on this new cluster assignment. These two steps are iterated until the cluster assignment no longer changes. The algorithm is defined as follows.

The k-**means algorithm** produces the prespecified number of clusters k and consists of the following steps:

Step 1: Choose the initial centroids of k clusters.

Step 2: Given the centroids, assign each observation to a cluster whose centroid is the closest (in terms of Euclidean distance) to that observation.

Step 3: Choose the new centroid of each cluster whose coordinate equals the within-cluster mean of the corresponding variable.

Step 4: Repeat Step 2 and 3 until cluster assignments no longer change.

Note that the researchers must choose the number of clusters k and the initial centroid of each cluster. In R, the initial locations of centroids are randomly selected, unless otherwise specified.

It is typically a good idea to *standardize* the inputs before applying the k-means algorithm. Doing so brings all variables to the same scale so that the clustering result does not depend on how each variable is measured. This is done by computing the *z-score* introduced earlier (see equation (3.1)). Recall that we compute the z-score of a variable by subtracting the mean from it (called *centering*) and then dividing it by the standard deviation (called *scaling*). In R, we can standardize a variable or a set of variables using the `scale()` function, which takes either a vector of a single variable or a matrix of multiple variables.

Going back to our study of partisanship, we apply the k-means clustering algorithm separately to the DW-NOMINATE scores for the 80th and 112th Congresses. We choose $k=2$ and $k=4$, producing 2 and 4 clusters, respectively. The function `kmeans()` implements the k-means algorithm in R. The function has various arguments, but the first argument `x` takes a *matrix* of observations to which one applies the k-means algorithm. For our application, this matrix has two columns, representing the first and second dimensions of DW-NOMINATE scores, and the number of rows equals the number of legislators in each Congress. We use the `cbind()` (or "column bind") function to combine two variables by columns in order to create this matrix. As a side note, the `rbind()` (or "row bind") function allows one to bind two vectors or matrices by rows. We do not standardize the input variables in this application since the DW-NOMINATE scores are already scaled in a substantively meaningful manner.

```r
dwnom80 <- cbind(congress$dwnom1[congress$congress == 80],
                 congress$dwnom2[congress$congress == 80])
dwnom112 <- cbind(congress$dwnom1[congress$congress == 112],
                  congress$dwnom2[congress$congress == 112])
```

The main arguments of the `kmeans()` function include `centers` (the number of clusters), `iter.max` (the maximum number of iterations), and `nstart` (the number of randomly chosen initial centroids). It is recommended that the `nstart` argument is specified so that the algorithm is run several times with different starting values (the `kmeans()` function reports the best results). We begin by fitting the k-means algorithm with two clusters and five randomly selected starting values.

```r
## k-means with 2 clusters
k80two.out <- kmeans(dwnom80, centers = 2, nstart = 5)
k112two.out <- kmeans(dwnom112, centers = 2, nstart = 5)
```

The output objects, `k80two.out` and `k112two.out`, are *list*s, which contain various elements regarding the results of the application of the k-means algorithm. They include `iter` (an integer representing the number of iterations until convergence, which is achieved when the cluster assignments no longer change), `cluster` (a vector of the resulting cluster membership), and `centers` (a matrix of cluster centroids).

```r
## elements of a list
names(k80two.out)

## [1] "cluster"      "centers"      "totss"
## [4] "withinss"     "tot.withinss" "betweenss"
## [7] "size"         "iter"         "ifault"
```

As explained in section 3.7.2, the elements within each list can be accessed using `$` like we access a variable in a data frame object. In both cases, the algorithm converged in just 1 iteration, which can be checked by examining the `iter` element of the output list object. The default maximum number of iterations is 10. If convergence is not achieved, the `iter.max` argument needs to be specified as a number greater than 10.

We now examine the final centroids of the resulting clusters using a 2-cluster model. Each output row shows a cluster with the horizontal and vertical coordinates of its centroid in the first and second columns, respectively.

```
## final centroids
k80two.out$centers

##             [,1]       [,2]
## 1  0.14681029 -0.3389293
## 2 -0.04843704  0.7827259

k112two.out$centers

##            [,1]       [,2]
## 1 -0.3912687 0.03260696
## 2  0.6776736 0.09061157
```

We next compute the numbers of Democratic and Republican legislators who belong to each cluster by creating a cross tabulation of party and cluster label variables.

```
## number of observations for each cluster by party
table(party = congress$party[congress$congress == 80],
      cluster = k80two.out$cluster)

##               cluster
## party            1   2
##   Democrat      62 132
##   Other          2   0
##   Republican 247   3

table(party = congress$party[congress$congress == 112],
      cluster = k112two.out$cluster)

##               cluster
## party            1   2
##   Democrat     200   0
##   Other          0   0
##   Republican     1 242
```

We find that for the 112th Congress, the k-means algorithm with 2 clusters produces 1 cluster containing only Democrats and the other consisting only of Republicans. While we chose the number of clusters to be 2 in this case, the algorithm discovers that these 2 clusters perfectly align on partisanship. In contrast, for the 80th Congress, one of the clusters contains a significant number of Democrats as well as Republicans. This is consistent with the fact that political polarization has worsened over time.

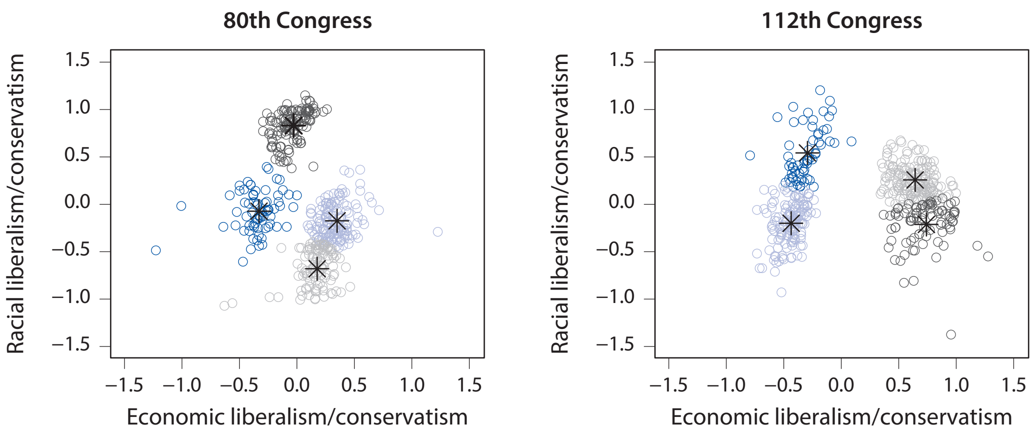
Next, we apply the k-means algorithm with 4 clusters and visualize the results. We begin by fitting the 4-cluster model to the 80th and 112th Congresses.

```
## k-means with 4 clusters
k80four.out <- kmeans(dwnom80, centers = 4, nstart = 5)
k112four.out <- kmeans(dwnom112, centers = 4, nstart = 5)
```

To visualize the results, we use the `plot()` function to create a scatter plot. The following syntax assigns different colors to observations that belong to different clusters. The centroid of each cluster is indicated by an asterisk.

```
## plotting the results using the labels and limits defined earlier
plot(dwnom80, col = k80four.out$cluster + 1, xlab = xlab, ylab = ylab,
     xlim = lim, ylim = lim, main = "80th Congress")
## plotting the centroids
points(k80four.out$centers, pch = 8, cex = 2)
## 112th Congress
plot(dwnom112, col = k112four.out$cluster + 1, xlab = xlab, ylab = ylab,
     xlim = lim, ylim = lim, main = "112th Congress")
points(k112four.out$centers, pch = 8, cex = 2)
```

For the full-color version of the plots, see page C2.



The cex argument given in the points() function controls the font size so that the centroid of each cluster is clearly visible. In addition, the pch argument specifies a certain symbol for plotting. Finally, we specify a vector of integer values, rather than actual color names, for the col argument so that each integer value is used for the corresponding cluster. We add 1 to the cluster labels so that we do not use black, the color of the cluster centroids, for the observations belonging to one of the clusters. The palette() function displays the exact correspondence between the color names and integer values (see section 5.3.3 for more details on the use of color in R).

```
palette()

## [1] "black"   "red"     "green3"  "blue"    "cyan"
## [6] "magenta" "yellow"  "gray"
```

The results show that the 4-cluster model splits the Democrats into 2 clusters and the Republicans into 2 clusters. Within each party, the division between the 2 clusters is clearest among the Democrats in the 80th Congress. For both parties, the within-party division is along the racial dimension. In contrast, the economic dimension dominates the difference between the two parties.

Clustering algorithms such as the k-means algorithm represent examples of *unsupervised learning* methods. Unlike in *supervised learning*, there is no outcome variable. Instead, the goal of unsupervised learning is to discover the hidden structures in data. The difficulty of unsupervised learning is that there is no clear measure of success and failure. In the absence of outcome data, it is difficult to know whether these clustering algorithms are producing the "correct" results. For this reason, human judgment is often required to make sure that the findings produced by clustering algorithms are reasonable.

3.8 Summary

This chapter focused on the issue of measurement. We discussed **survey sampling** as a principled and efficient way to infer the characteristics of a potentially large population from a small number of randomly sampled units without enumerating all units in the population. In chapter 2, we learned about the randomization of treatment assignment, which ensures that the treatment and control groups are equal on average in all aspects but the receipt of treatment. In survey sampling, we used the random sampling of units to make the sample representative of a target population. This allows researchers to infer population characteristics from the sample obtained from random sampling.

While random sampling is an effective technique, there are several complications in practice. First, while random sampling requires a complete list of potential units to be sampled, it is often difficult to obtain such a sampling frame. Second, due to cost and logistical constraints, researchers are forced to use complex random sampling techniques. Third, surveys typically lead to both **unit and item nonresponses**, which, if occurring nonrandomly, threaten the validity of inference. In recent years, the nonresponse rate of phone surveys has dramatically increased. As a result, many polling firms are starting to use cheap Internet surveys through platforms like Qualtrics, even though many such surveys are not based on probability sampling. Beyond nonresponse problems, sensitive questions in surveys often result in **social desirability bias** in which respondents may falsify their answers and provide socially acceptable answers.

Furthermore, social scientists often face the question of how to measure latent concepts such as ideology and ability. We discussed an application of item response theory to political polarization in the US Congress. The idea is to infer legislators' ideological positions from their roll call votes. The same method was also applied to measure students' abilities from standardized tests. Using the estimated ideal points as an example, we also learned how to apply a basic **clustering algorithm** called the k-means algorithm in order to discover latent groups of observations with similar characteristics in data.

In addition to these concepts and methods, the chapter also introduced various numerical and visual summaries of data. While a **bar plot** summarizes the distribution of a factor variable, **box plots** and **histograms** are useful tools for depicting the distribution of continuous variables. The **correlation coefficient** numerically characterizes the association between two variables, whereas a **scatter plot** plots one variable against the other. Finally, unlike scatter plots, **quantile–quantile plots** (Q–Q plots) enable comparison of the distributions of two variables even when they are not measured in the same units.

3.9 Exercises

3.9.1 CHANGING MINDS ON GAY MARRIAGE: REVISITED

In this exercise, we revisit the gay marriage study we analyzed in section 2.8.2. It is important to work on that exercise before answering the following questions. In May 2015, three scholars reported several irregularities in the data set used to produce the results in the study.[3] They found that the gay marriage experimental data were statistically indistinguishable from data in the Cooperative Campaign Analysis Project (CCAP), which interviewed voters throughout the 2012 US presidential campaign. The scholars suggested that the CCAP survey data—and not the original data alleged to have been collected in the experiment—were used to produce the results reported in the gay marriage study. The release of a report on these irregularities ultimately led to the retraction of the original article. In this exercise, we will use several measurement strategies to reproduce the irregularities observed in the gay marriage data set.

To do so, we will use two CSV data files: a reshaped version of the original data set in which every observation corresponds to a unique respondent, `gayreshaped.csv` (see table 3.5), and the 2012 CCAP data set alleged to have been used as the basis for the gay marriage study results, `ccap2012.csv` (see table 3.6). Note that the feeling

[3] This exercise is based on the unpublished report "Irregularities in LaCour (2014)" by David Broockman, Joshua Kalla, and Peter Aronow.

Table 3.5. Gay Marriage Reshaped Data.

Variable	*Description*
`study`	which study the data set is from (`1` = study 1, `2` = study 2)
`treatment`	five possible treatment assignment options
`therm1`	survey thermometer rating of feeling towards gay couples in wave 1 (`0`–`100`)
`therm2`	survey thermometer rating of feeling towards gay couples in wave 2 (`0`–`100`)
`therm3`	survey thermometer rating of feeling towards gay couples in wave 3 (`0`–`100`)
`therm4`	survey thermometer rating of feeling towards gay couples in wave 4 (`0`–`100`)

Note: See table 2.7 for the original data.

Table 3.6. 2012 Cooperative Campaign Analysis Project (CCAP) Survey Data.

Variable	*Description*
`caseid`	unique respondent ID
`gaytherm`	survey thermometer rating of feeling towards gay couples (`0`–`100`)

thermometer measures how warmly respondents feel towards gay couples on a 0–100 scale.

1. In the gay marriage study, researchers used seven waves of a survey to assess how lasting the persuasion effects were over time. One irregularity the scholars found is that responses across survey waves in the control group (where no canvassing occurred) had unusually high correlation over time. What is the correlation between respondents' feeling thermometer ratings in waves 1 and 2 for the control group in study 1? To handle missing data, we should set the `use` argument of the `cor()` function to `"complete.obs"` so that the correlation is computed using only observations that have no missing data. Provide a brief substantive interpretation of the results.

2. Repeat the previous question using study 2 and comparing all waves within the control group. Note that the `cor()` function can take a single data frame with multiple variables. To handle missing data in this case, we can set the `use` argument to `"pairwise.complete.obs"`. This means that the `cor()` function uses all observations that have no missing values for a given pair of waves even if some of them have missing values in other waves. Briefly interpret the results.

3. Most surveys find at least some *outliers* or individuals whose responses are substantially different from the rest of the data. In addition, some respondents may change their responses erratically over time. Create a scatter plot to visualize the relationships between wave 1 and each of the subsequent waves in study 2. Use only the control group. Interpret the results.

4. The researchers found that the data of the gay marriage study appeared unusually similar to the 2012 CCAP data set even though they were supposed to be samples of completely different respondents. We use the data contained in `ccap2012.csv` and `gayreshaped.csv` to compare the two samples. Create a histogram of the 2012 CCAP feeling thermometer, the wave-1 feeling thermometer from study 1, and the wave-1 feeling thermometer from study 2. There are a large number of missing values in the CCAP data. Consider how the missing data might have been recoded in the gay marriage study. To facilitate the comparison across histograms, use the `breaks` argument in the `hist()` function to keep the bin sizes equal across histograms. Briefly comment on the results.

5. A more direct way to compare the distributions of two samples is through a *quantile–quantile plot*. Use this visualization method to conduct the same comparison as in the previous question. Briefly interpret the plots.

3.9.2 POLITICAL EFFICACY IN CHINA AND MEXICO

In 2002, the World Health Organization conducted a survey of two provinces in China and three provinces in Mexico.[4] One issue of interest, which we analyze in this exercise, concerns political efficacy. First, the following self-assessment question was asked.

> How much say do you have in getting the government to address issues that interest you?
>
> (5) Unlimited say, (4) A lot of say, (3) Some say, (2) Little say, (1) No say at all.

After the self-assessment question, three vignette questions were asked.

> [Alison] lacks clean drinking water. She and her neighbors are supporting an opposition candidate in the forthcoming elections that has promised to address the issue. It appears that so many people in her area feel the same way that the opposition candidate will defeat the incumbent representative.
>
> [Jane] lacks clean drinking water because the government is pursuing an industrial development plan. In the campaign for an upcoming election, an opposition party has promised to address the issue, but she feels it would be futile to vote for the opposition since the government is certain to win.
>
> [Moses] lacks clean drinking water. He would like to change this, but he can't vote, and feels that no one in the government cares about this issue. So he suffers in silence, hoping something will be done in the future.

The respondent was asked to assess each vignette in the same manner as the self-assessment question.

[4] This exercise is based on Gary King, Christopher J.L. Murray, Joshua A. Salomon, and Ajay Tandon (2004) "Enhancing the validity and cross-cultural comparability of measurement in survey research." *American Political Science Review*, vol. 98, no. 1 (February), pp. 191–207.

Table 3.7. Vignette Survey Data.

Variable	*Description*
`self`	self-assessment response
`alison`	response to the Alison vignette
`jane`	response to the Jane vignette
`moses`	response to the Moses vignette
`china`	`1` for China and `0` for Mexico
`age`	age of respondent in years

> How much say does [“name”] have in getting the government to address issues that interest [him/her]?
>
> (5) Unlimited say, (4) A lot of say, (3) Some say, (2) Little say, (1) No say at all.

[“name”] is replaced by either Alison, Jane, or Moses.

The data set we analyze `vignettes.csv` contains the variables whose names and descriptions are given in table 3.7. In the analysis that follows, we assume that these survey responses can be treated as numerical values. For example, “Unlimited say” = 5, and “Little say” = 2. This approach is not appropriate if, for example, the difference between “Unlimited say” and “A lot of say” is not the same as the difference between “Little say” and “No say at all.” However, relaxing this assumption is beyond the scope of this chapter.

1. We begin by analyzing the self-assessment question. Plot the distribution of responses separately for China and Mexico using bar plots, where the vertical axis is the proportion of respondents. In addition, compute the mean response for each country. According to this analysis, which country appears to have a higher degree of political efficacy? How does this evidence match with the fact that in the 2000 election, Mexican citizens voted out of office the ruling Institutional Revolutionary Party (PRI) who had governed the country for more than 80 years, while Chinese citizens have not been able to vote in a fair election to date?

2. We examine the possibility that any difference in the levels of efficacy between Mexican and Chinese respondents is due to the difference in their age distributions. Create histograms for the age variable separately for Mexican and Chinese respondents. Add a vertical line representing the median age of the respondents for each country. In addition, use a quantile–quantile plot to compare the two age distributions. What differences in age distribution do you observe between the two countries? Answer this question by interpreting each plot.

3. One problem with the self-assessment question is that survey respondents may interpret the question differently. For example, two respondents who choose the same answer may be facing quite different political situations and hence may interpret “A lot of say” differently. To address this problem, we rank a respondent’s answer to the self-assessment question relative to the same respondent’s answer

to a vignette question. Compute the proportion of respondents, again separately for China and Mexico, who rank themselves (according to the self-assessment question) as having less say in the government's decisions than Moses (the last vignette). How does the result of this analysis differ from that of the previous analysis? Give a brief interpretation of the result.

4. We focus on survey respondents who ranked these three vignettes in the expected order (i.e., Alison $\geq$ Jane $\geq$ Moses). Create a variable that represents how respondents rank themselves relative to these vignettes. This variable should be equal to `1` if respondents rank themselves less than Moses, `2` if ranked the same as Moses or between Moses and Jane, `3` if ranked the same as Jane or between Jane and Alison, and `4` if ranked the same as Alison or higher. Create the bar plots of this new variable as done in question 1. The vertical axis should represent the proportion of respondents for each response category. Also, compute the mean value of this new variable separately for China and Mexico. Give a brief interpretation of the result by comparing these results with those obtained in question 1.

5. Is the problem identified above more or less severe among older respondents when compared to younger ones? Answer the previous question separately for those who are 40 years or older and those who are younger than 40 years. Does your conclusion for the previous question differ between these two groups of respondents? Relate your discussion to your finding for question 2.

3.9.3 VOTING IN THE UNITED NATIONS GENERAL ASSEMBLY

Like legislators in the US Congress, the member states of the United Nations (UN) are politically divided on many issues such as trade, nuclear disarmament, and human rights. During the Cold War, countries in the UN General Assembly tended to split into two factions: one led by the capitalist United States and the other by the communist Soviet Union. In this exercise, we will analyze how states' ideological positions, as captured by their votes on UN resolutions, have changed since the fall of communism.[5] Table 3.8 presents the names and descriptions of the variables in the data set contained in the CSV file `unvoting.csv`.

In the analysis that follows, we measure state preferences in two ways. First, we can use the proportion of votes by each country that coincide with votes on the same issue cast by the two major Cold War powers: the United States and the Soviet Union. For example, if a country voted for 10 resolutions in 1992, and if its vote matched the United States's vote on exactly 6 of these resolutions, the variable `PctAgreeUS` in 1992 would equal 60 for this country. Second, we can also measure state preferences in terms of numerical ideal points as explained in section 3.5. These ideal points capture what international relations scholars have called countries' *liberalism* on issues such as political freedom, democratization, and financial liberalization. The two measures

[5] This exercise is based on Michael A. Bailey, Anton Strezhnev, and Erik Voeten (2015) "Estimating dynamic state preferences from United Nations voting data." *Journal of Conflict Resolution*, doi = 10.1177/0022002715595700.

Table 3.8. United Nations Ideal Points Data.

Variable	*Description*
`CountryName`	name of the country
`CountryAbb`	abbreviated name of the country
`idealpoint`	its estimated ideal point
`Year`	year for which the ideal point is estimated
`PctAgreeUS`	proportion of votes that match with votes cast by the United States on the same issue
`PctAgreeRUSSIA`	proportion of votes that match with votes cast by Russia/the Soviet Union on the same issue

are highly correlated, with larger (more liberal) ideal points corresponding to a higher proportion of votes that agree with the United States.

1. We begin by examining how the distribution of state ideal points has changed since the end of communism. Plot the distribution of ideal points separately for 1980 and 2000—about 10 years before and 10 years after the fall of the Berlin Wall, respectively. Add the median to each plot as a vertical line. How do the two distributions differ? Pay attention to the degree of polarization and give a brief substantive interpretation of the results. Use the `quantile()` function to quantify the patterns you identified.

2. Next, examine how the number of countries voting with the United States has changed over time. Plot the average percentage agreement with the United States across all countries over time. Also, add the average percentage agreement with Russia as another line for comparison. Using the `tapply()` function may help with this analysis. Does the United States appear to be getting more or less isolated over time, as compared to Russia? Identify some countries that are consistently pro-US. What are the most pro-Russian countries? Give a brief substantive interpretation of the results.

3. One problem with using the proportion of votes that agree with the United States or Russia as a measure of state preferences is that the ideological positions, and consequently the voting patterns, of the two countries might themselves have changed over time. This makes it difficult to know which countries' ideological positions have changed. Investigate this issue by plotting the evolution of the two countries' ideal points over time. Add the yearly median ideal point of all countries. How might the results of this analysis modify (or not) your interpretation of the previous analysis?

4. Let's examine how countries that were formerly part of the Soviet Union differ in terms of their ideology and UN voting compared to countries that were not part of the Soviet Union. The former Soviet Union countries are Estonia, Latvia, Lithuania, Belarus, Moldova, Ukraine, Armenia, Azerbaijan, Georgia,

Kazakhstan, Kyrgyzstan, Tajikistan, Turkmenistan, Uzbekistan, and Russia. The `%in%` operator, which is used as `x %in% y`, may be useful. This operator returns a logical vector whose elements are `TRUE` if the corresponding element of vector `x` is equal to a value contained in vector `y` and otherwise `FALSE`. Focus on the most recently available UN data from 2012 and plot each post-Soviet Union state's ideal point against the proportion of its votes that agree with the United States. Compare the post-Soviet Union states, within the same plot, against the other countries. Briefly comment on what you observe.

5. We have just seen that while some post-Soviet countries have retained nonliberal ideologies, other post-Soviet countries were much more liberal in 2012. Let's examine how the median ideal points of Soviet/post-Soviet countries and all other countries have varied over all the years in the data. Plot these median ideal points by year. Be sure to indicate 1989, the year of the fall of the Berlin Wall, on the graph. Briefly comment on what you observe.

6. Following the end of communism, countries that were formerly part of the Soviet Union have become much more ideologically diverse. Is this also true of the world as a whole? In other words, do countries still divide into two ideological factions? Let's assess this question by applying the *k*-means clustering algorithm to ideal points and the proportion of votes agreeing with the United States. Initiate the algorithm with just two centroids and visualize the results separately for 1989 and 2012. Briefly comment on the results.