CONTENTS

1. INTRODUCTION

This book provides a friendly introduction to data analysis for the social sciences. It covers the fundamental methods of quantitative social science research, using plain language and assuming absolutely no prior knowledge of the subject matter.

Proceeding step by step, we show how to analyze real-world data using the statistical program R for the purpose of answering a wide range of substantive questions. Along the way, we teach the statistical concepts and programming skills needed to conduct and evaluate social scientific studies. We explain not only how to perform the analyses but also how to interpret the results and identify the analyses' strengths and potential limitations.

Through this book, you will learn how to *measure*, *predict*, and *explain* quantities of interest based on data. These are the three fundamental goals of quantitative social science research. (See outline 1.1.)

R symbols, operators, and functions introduced in this chapter: +, −, *, /, <-, ", (), sqrt(), #, setwd(), read.csv(), View(), head(), dim(), $, and mean().

WHY DO WE ANALYZE DATA IN THE SOCIAL SCIENCES?

In the social sciences we analyze data to:

- *measure* a quantity of interest, such as the proportion of eligible voters in favor of a particular policy
- *predict* a quantity of interest, such as the likely winner of an upcoming election
- *explain* a quantity of interest, such as the causal effect of attending a private school on student test scores.

OUTLINE 1.1. The three goals of quantitative social science research.

Figuring out whether you aim to measure, predict, and/or explain a quantity of interest should always precede the analysis and often also precede the data collection. As you will learn, the goals of your research will determine (i) what data you need to collect and how, (ii) the statistical methods you use, and (iii) what you pay attention to in the analysis. As you read this book and learn about each goal in detail, the distinctions will become clearer. Here we provide a brief preview.

To measure a quantity of interest such as a population characteristic, we often use survey data, that is, information collected on a sample of individuals from the target population. To analyze the data, we may compute various descriptive statistics, such as mean and median, and create visualizations like histograms and scatter plots. The validity of our conclusions depends on whether the sample is representative of the target population. To measure the proportion of eligible voters in favor of a particular policy, for example, our conclusions will be valid if the sample of voters surveyed is representative of *all* eligible voters.

To predict a quantity of interest, we typically use a statistical model such as a linear regression model to summarize the relationship between the predictors and the outcome variable of interest. The stronger the association between the predictors and the outcome variable, the better the predictive model will usually be. To predict the likely winner of an upcoming election, for example, if economic conditions are strongly associated with the electoral outcomes of candidates from the incumbent party, we may be able to use the current unemployment rate as our predictor.

To explain a quantity of interest such as the causal effect of a treatment on an outcome, we need to find or create a situation in which the group of individuals who received the treatment is comparable, in the aggregate, to the group of individuals who did not. In other words, we need to eliminate or control for all confounding variables, which are variables that affect both (i) the likelihood of receiving the treatment and (ii) the outcome variable. For example, when estimating the causal effect of attending a private school on student test scores, family wealth is a potential confounding variable. Students from wealthier families are more likely to attend a private school and also more likely to receive after-school tutoring, which might have a positive impact on their test scores. To produce valid estimates of causal effects, we may conduct a randomized experiment, which eliminates all confounding variables by assigning the treatment at random. In the current example, we would achieve this by using a lottery to determine which students attend private schools and which do not. Alternatively, if we cannot conduct a randomized experiment and need to rely on observational data instead, we would need to use statistical methods to control for all confounding variables such as family wealth. Otherwise, we would not know what portion of the difference in average test scores between private and public school students was the result of the type of school attended and what portion was the result of family background.

1.1 BOOK OVERVIEW

The book consists of seven chapters.

Chapter 1 is the introductory chapter, which lays the groundwork for the forthcoming data analyses.

Chapters 2 through 5 each introduce one or two published social scientific studies. In these chapters, we show how to analyze real-world datasets to answer different kinds of substantive questions. Specifically, we teach how to use several quantitative methods to measure, predict, and explain quantities of interest. (See outline 1.2, which indicates how each chapter relates to the three goals of quantitative social science research.)

BOOK OUTLINE

Chapter	Goal
1. Introduction	
2. Estimating Causal Effects with Randomized Experiments	Explain
3. Inferring Population Characteristics via Survey Research	Measure
4. Predicting Outcomes Using Linear Regression	Predict
5. Estimating Causal Effects with Observational Data	Explain
6. Probability	
7. Quantifying Uncertainty	All Three

OUTLINE 1.2. Book outline showing how each chapter relates to the three goals of quantitative social science research.

As you can see, chapters 2 and 5 are both about explanation, also known as causal inference. They teach how to estimate causal effects using different types of data. Since the methods differ, they are presented in separate chapters.

The book progresses from simple to more complex methods. Chapter 2 shows how to estimate causal effects using data from a randomized experiment. Chapter 3 is about measurement and teaches how to infer the characteristics of an entire population from a sample of survey respondents. Chapter 4 is about prediction and demonstrates how to use simple linear regression. Chapter 5 shows how to estimate causal effects with observational data and teaches multiple linear regression, the most complicated method we see in the book.

In chapter 6, we cover basic probability, and in chapter 7 we complete some of the analyses from chapters 2 through 5 by quantifying the uncertainty of our empirical findings. A more detailed description of each chapter is below.

1.2 CHAPTER SUMMARIES

In the current introductory chapter, we discuss why data analysis is a required skill among social scientists. We also explain how to get our computers ready, and we familiarize ourselves with RStudio and R, the two programs we will use. Then, we learn to load and make sense of data and practice computing and interpreting means.

In chapter 2, we define and learn how to estimate causal effects using data from a randomized experiment. As the working example, we analyze data from one of the largest experiments in U.S. education policy research, Project STAR, to determine whether attending a small class improves student performance.

In chapter 3, we use survey research to measure population characteristics. In addition, we learn how to visualize and summarize the distribution of single variables as well as the relationship between two variables. To illustrate these concepts, we analyze data related to the 2016 British referendum on withdrawing from the European Union, a decision popularly known as Brexit.

In chapter 4, we learn how to predict outcomes using simple linear regression models. For practice, we analyze data from 170 countries in order to predict growth in gross domestic product (GDP) using night-time light emissions as measured from space.

In chapter 5, we return to estimating causal effects, but this time using observational data. We define confounding variables, examine how their presence complicates the estimation of causal effects, and learn how to use multiple linear regression models to help mitigate the potential bias these variables introduce. To illustrate how this works step by step, we estimate the effects of Russian TV reception on the 2014 Ukrainian parliamentary elections. In this context, we introduce the concepts of internal and external validity. We then discuss the pros and cons of randomized experiments and of observational studies.

In chapter 6, we shift our focus away from data analysis to cover basic probability. We learn about random variables and their distributions as well as the distinction between population parameters and sample statistics. We then discuss the two large sample theorems that enable us to measure statistical uncertainty.

In chapter 7, we use everything we have learned in the preceding chapters and show how to quantify the uncertainty in our empirical findings in order to draw conclusions at the population level. In particular, we show how to quantify the uncertainty in (i) population inferences, (ii) predictions, and (iii) causal effect estimates. As illustrations, we complete some of the analyses we started in chapters 2 through 5.

1.3 HOW TO USE THIS BOOK

This is no ordinary textbook on data analysis. It is intentionally designed to accommodate readers with a variety of math and programming backgrounds.

The book uses a two-column layout: a main column and a side column or margin.

The main column contains the essential material and code, which are intended for all readers, except for the sections labeled FORMULA IN DETAIL. These contain more advanced material and are clearly identified so that you can easily skip them if you so choose.

In the margin are various types of notes and figures, each with a different purpose:

- At the beginning of each chapter, we list the R functions, symbols, and operators that will be introduced. You can look through the list to get a sense of what will be covered. (See, for example, the list for this chapter shown on the first page, and note that we always display code in cyan.)
- TIPs include supplemental material, such as additional explanations, answers to common questions, notes on best practices, and recommendations.
- RECALLs remind you of relevant information mentioned earlier in the book. These reminders are particularly helpful when the book is read only a few pages at a time, such as over the course of a semester.
- To help you review the core concepts, which are shown in **bold red** in the main text, we repeat their definitions in the margin. These notes are displayed in red.
- To help you with R functions, symbols, and operators, the first time these are introduced, we include in the margin an explanation of how they work and provide an example. These explanations are displayed in a cyan-colored frame.

At the end of each chapter, in place of the usual list of supplementary exercises, we include CHEATSHEETS to help you review the core concepts as well as the R functions, symbols, and operators covered.

Supplementary chapter-specific exercises, categorized by degree of difficulty, are available at http://press.princeton.edu/dss.

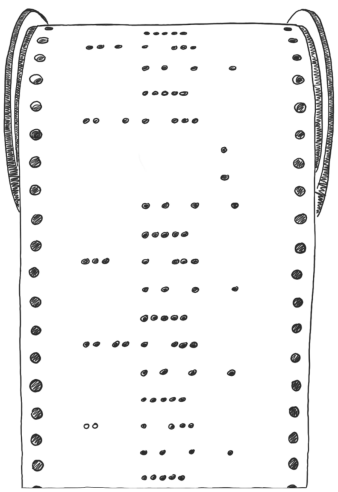
Finally, at the end of the book, we include three separate indexes for concepts, mathematical notation, and R-related topics.

1.4 WHY LEARN TO ANALYZE DATA?

As a social scientist, sooner or later you will need to rely on data to (i) measure the characteristics of a certain population of interest, (ii) make predictions, and/or (iii) make or evaluate decisions involving cause-and-effect relationships. What proportion of a population is in favor of a particular policy? Who is the candidate most likely to win an upcoming election? Shall we implement a particular policy to boost economic growth? You will want to be able to answer these types of questions either by analyzing data yourself or by understanding and assessing someone else's data analysis.

Even if you are not planning to become a social scientist, it is useful for you to know how to analyze data and/or how to distinguish a good quantitative study from a poorly conducted one. These are highly marketable skills. Recent advancements in computing power and the proliferation of data have increased the demand for data analysts who can inform decision makers in the public and private sectors alike.

The analytical skills you will learn by making your way through this book can also be used to improve everyday decisions, from choosing a candidate to vote for to determining the best way to increase your productivity. Perhaps most importantly, by learning the strengths and limitations of different quantitative methods, you will become less vulnerable to arguments based on faulty inferences from data. In the era of big data, we all stand to benefit from becoming savvy consumers of quantitative research, even if we do not all become skilled researchers ourselves.



1.4.1 LEARNING TO CODE

For the purpose of analyzing data, we write and run code. Code contains instructions that a computer can implement. These instructions consist of sequences of clearly defined steps written in a particular programming language. In this book, we code in R, which is a programming language used by many data analysts.

Don't worry if you have never done any coding before. Learning to code is not as difficult as one might think. You may even find it fun. Back in 1944, when the first programmable computer in the United States was built, only highly trained mathematicians were able to code. At that time, coding required punching paper tape in specific sequences that the machine could read. (See a rendition of what this tape looked like in the margin.) Today, anyone with access to a computer, some spare time, and a little patience can learn how to code.

1.5 GETTING READY

To perform the analyses in this book, we first need to download and install the necessary files and programs. We should also familiarize ourselves with RStudio, which is the interface we use throughout.

❶ DOWNLOAD AND SAVE FILES

All the files we will use are in a folder named DSS, which is available at http://press.princeton.edu/dss. For easy access, we recommend saving the folder on your Desktop. This is where the code used throughout the book assumes the DSS folder is located. In case you choose to save the folder elsewhere, we also provide instructions for making the necessary changes to the code.

TIP: By default, your computer will likely save the DSS folder to your Downloads. To move it, you can copy and paste it or drag it to the new location.

❷ DOWNLOAD AND INSTALL R AND RSTUDIO

We will use two programs: R and RStudio. R is the statistical program, the engine if you will, that will perform the calculations and create the graphics for us. RStudio is the user-friendly interface we will use to communicate with R. While we could use R directly, going through RStudio makes writing and running code much easier.

Why do we use R as our statistical program? Because it is free, open-source (anyone can see the underlying code and improve it), powerful, and flexible. It is also widely used. Indeed, many jobs these days require knowledge of R.

Unfortunately, these programs are compatible only with Linux, Mac, and Windows operating systems. They cannot be used on tablets or phones. We provide instructions for using these two programs on a Mac or a Windows computer.

To download and install R, go to http://cran.r-project.org, select the link that matches your operating system, and follow the instructions.

To download and install RStudio, go to http://rstudio.com, select the link that matches your operating system, and follow the instructions.

❸ BECOME FAMILIAR WITH RSTUDIO

To analyze data, we always operate R through RStudio. Let's take a moment to become acquainted with RStudio's layout.

After installing both programs, go ahead and start RStudio. Then, from within RStudio, open a new R script, which is the type of file we use to store the code we write to analyze data. Instructions are shown in the margin.

TIP: How do we open a new R script? In the RStudio dropdown menu, click on File > New File > R Script. A new "Untitled" file will open. The extension of this type of file is ".R", which is why R scripts are also called R files.

After opening a new R script, RStudio's interface should look like figure 1.1.

- The upper-left window is the *R script*, which is where we write and run code, giving R commands to execute.
- The lower-left window is the *R console*, where R provides either the results of successfully executed code (known as outputs) or any error messages.
- The upper-right window is the *environment*, which is the storage room of the current R session. It lists all the objects we have created. (We will soon explain what objects are and provide examples showing how the environment works.)
- The lower-right window is where we find the *help* and *plots tabs*, which we will learn how to use later on.

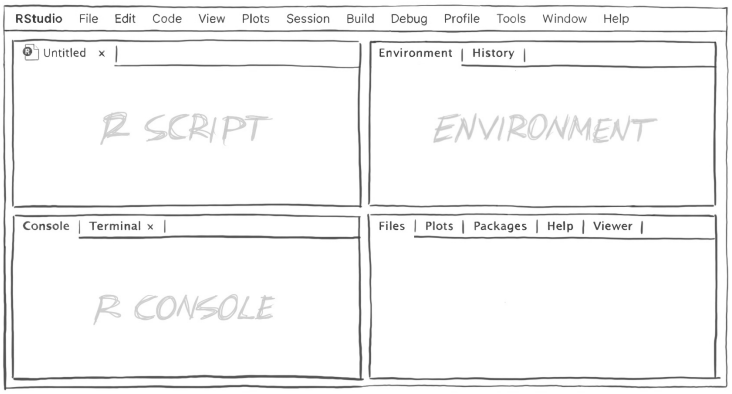


FIGURE 1.1. Layout of RStudio after opening a new R script. The upper-left window is the R script. The lower-left window is the R console. The upper-right window is the environment of the R session. The plots and help tabs appear in the lower-right window.

1.6 INTRODUCTION TO R

To use R, we need to learn the R programming language. (R is the name of both the statistical program and the programming language.) Learning a programming language is like learning a foreign language. It is not easy, and it takes a lot of practice and patience. The exercises in this book will help you learn to code in R, so be sure to follow along. Practice is everything!

Let's begin. R can be used to do many things. In our case, we will use R (i) as a calculator; (ii) to create objects, which is how R stores data; and (iii) to interact with data using functions.

WE WILL USE THE STATISTICAL PROGRAM R TO:

(i) do calculations
(ii) create objects
(iii) use functions.

1.6.1 DOING CALCULATIONS IN R

We can use R as a calculator. R can do summation (+), subtraction (−), multiplication (*), and division (/), as well as other more complicated mathematical operations. For example, the code to ask R to calculate 1 plus 3 is:

+, −, *, and / are some of the arithmetic operators recognized by R. Example: (4 − 1 + 3) * (2 / 3)

```
1 + 3
```

To run this or any other code, we first type it in the R script (the upper-left window of RStudio). Then, we highlight as much of it as we want to run and either (a) manually hit the run icon (shown in the margin) or (b) use the shortcut *command+enter* in Mac or *ctrl+enter* in Windows. The result, or output, of the executed code will show up in the R console (the lower-left window of RStudio). (Instead, we could type the code directly in the R console and hit enter, but we should avoid doing it that way. It is best to run code through an R script so that you can save it, re-run it, tweak it, expand it, and share it.)

After running the code above, we should see the following in the R console: first, the executed code shown in blue, indicating that R was able to run it without problems, and then the output shown in black. In this case, the output is:

```
4
```

Indeed, one plus three equals four.

Congratulations! You just wrote and ran your first line of code in R. Notice that now that you have written some code in the R script, RStudio shows the name of the file in red. This is to remind you that you have some unsaved changes. Once you save the file, the file name will return to black.

TIP: To save any changes you make to the R script, either (a) use the shortcut *command+S* in Mac or *ctrl+S* in Windows or (b) click on File > Save or Save As...

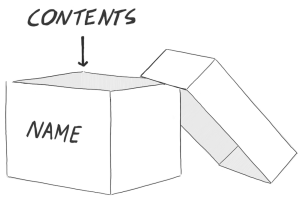
Throughout the book, we show the output that you should see in the R console right after the code that produces it. To distinguish the output from the code, we display the output with the symbol ## at the beginning of the line. For example, we display the code and output above as follows:

```
1 + 3
## [1] 4
```

TIP: Adding spaces around operators makes the code easier to read. R ignores these spaces. Example: 1+3 produces the same output as 1 + 3.

The first line, shown in cyan, is the code to be typed and run in the R script. The second line, which begins with ## and is shown in gray, is what should appear in the R console after running the code.

What does the number in brackets before the 4 mean? It indicates the position of the output immediately to its right. In this instance, [1] indicates that 4 is the first output of the code we ran. Later in the chapter, we will see examples of code that produce multiple outputs, which will clarify how this works.



1.6.2 CREATING OBJECTS IN R

In order to manipulate and analyze data, we need to load and store datasets. R stores information in what are known as objects, and so we need to learn how to create objects in R.

Think of an object as a box that can contain anything. All we need to do is give it a name, so that we know how to refer to it, and specify its contents.

To create an object in R, we use the assignment operator <-:

- To its left, we specify the name we want to give the object. This name can be anything as long as it does not begin with a number or contain spaces or special symbols like $ or % that are reserved for other purposes. Underscores _ are permitted and are good substitutes for spaces.
- To its right, we specify the contents of the object, that is, the data we want to store.

<- is the assignment operator. It creates new objects in R (unless one with the same name already exists, in which case R overwrites its contents). To its left, we specify the name of the object (without quotes). To its right, we specify the contents of the object. Example: four <- 4.

CREATING OBJECTS: To store data as an object in R, we run code using this format:

object_name <- *object_contents*

where:

- *object_name* is the name we want to give the object
- <- is the assignment operator, which creates an object by assigning contents to a name
- *object_contents* is the data we want to store in the object.

TIP: We would accomplish the same thing by running: four <- 4.

For example, if we want to create an object called *four* containing the output of the calculation 1+3, we run:

```
four <- 1 + 3
```

TIP: RStudio continues to work in the same R session until you quit the program. At that time, R will ask whether you want to save the workspace image, which contains all the objects you created during the R session. We recommend that you do not save it. If you need to continue to work with those objects, you can always re-create them by re-running your code.

Notice that after running the code above, the object will show up in the environment (the upper-right window in RStudio). As mentioned earlier, the environment is the storage room of the current R session. It shows the objects that we have created and that are available for us to use.

If we want to know the contents of the object *four*, we can type and run the name of the object in the R script. Its contents will appear in the R console. This is equivalent to asking R, what is inside the object named *four*?

```
four
## [1] 4
```

Not surprisingly, the object *four* contains the number 4.

Objects can contain text as well as numbers. For example, to create an object called *hello* containing the text "hi" we run:

```
hello <- "hi"
```

After running the code above, the environment should contain two objects: *four* and *hello*.

Let's stop here to learn something important about R. Look at the code above. Why did we use quotation marks around the content of the object "hi" but not around the name of the object hello? In other words, when do we use quotes " when coding in R? Here is the rule: When writing code, the names of objects, names of functions, and names of arguments as well as special values such as TRUE, FALSE, NA, and NULL should *not* be in quotes; all other text should be in quotes. (In the next subsection, we will see what we mean by functions and arguments. We will learn the meaning and usage of TRUE and FALSE in chapter 2 and of NA and NULL in chapter 3.)

" when writing code, the names of objects, names of functions, and names of arguments as well as special values such as TRUE, FALSE, NA, and NULL should not be in quotes; all other text should be in quotes. Examples: "this is just text", object_name. Never use quotes around a number unless you want R to treat it as text, in which case you will not be able to use it to perform arithmetic operations.

What would have happened had we tried to run the code above without quotes around *hi*? Go ahead and try it:

```
hello <- hi
## Error: object 'hi' not found
```

In the R console, you will see an error message (in red) that reads, "Error: object 'hi' not found". Indeed, by typing *hi* without quotes, you are telling R that *hi* is the name of an object. Because there is no object named *hi* in the environment, R gives you an error message. Encountering programming errors is part of the coding process. Try not to be discouraged by them.

TIP: If you have problems figuring out what a particular error means, Google it. Lots of data analysts participate in Q&A sites, such as Stack Overflow, which can be very helpful for this sort of thing.

A word of caution: R overwrites (replaces) old objects if we use the same name when creating a new object. For example, go ahead and run the following:

```
hello <- "hi, nice to meet you"
```

You should see that you still have only two objects in the environment: *four* and *hello*, but now *hello* contains the text "hi, nice to meet you" instead of simply "hi". To confirm this, we run:

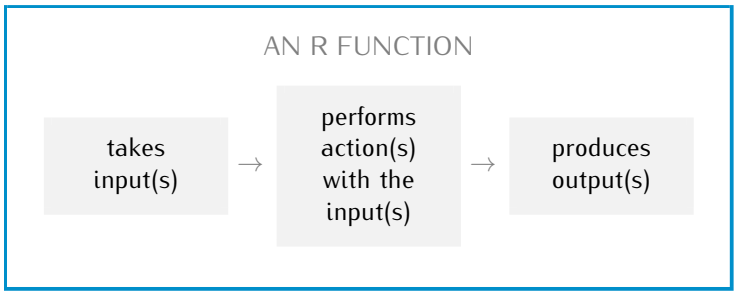
```
hello
## [1] "hi, nice to meet you"
```

Note also that R is case-sensitive. It will treat *Hello* as a completely different object name than *hello*. If we run the name *Hello* by mistake, R will not be able to find the object because there is no object in the environment called *Hello* with an uppercase H at the beginning. To avoid this problem, we recommend using all lowercase letters when naming objects.

1.6.3 USING FUNCTIONS IN R

Finally, we use R to interact with data, which requires using functions.

Think of a function as an action that you request R to perform with a particular piece of data, such as calculating the square root of four. A function takes one or more inputs, such as the number four; performs one or more actions with the inputs, such as calculating the square root of the inputs; and produces one or more outputs, such as the number two, which is the result of taking the square root of four.



R functions in this book: sqrt(), setwd(), read.csv(), View(), head(), dim(), mean(), ifelse(), table(), prop.table(), na.omit(), hist(), median(), sd(), var(), plot(), abline(), cor(), lm(), log(), c(), sample(), rnorm(), pnorm(), print(), nrow(), predict(), abs(), and summary().

() the names of functions are always followed by parentheses. Inside the parentheses, we write the argument(s) of the function, separated by commas if there is more than one argument. Example: *function_name(arg1, arg2)*.

TIP: There are two types of arguments: required and optional. Required arguments are the inputs that we must specify in order to use a particular function. Optional arguments are the inputs that we could specify if we wanted to modify the function's default settings.

Throughout the book, we will learn how to use the functions listed in the margin, which come automatically loaded with R. In time, we will learn their names, the actions they perform, the inputs they require, and the outputs they produce. Meanwhile, here are some important things to know about functions:

- The name of a function (without quotes) is always followed by parentheses: *function_name()*
- Inside the parentheses, we specify the inputs to be used by the function, which we refer to as arguments: *function_name(arguments)*
- Most functions require that we specify at least one argument but can take many optional arguments. When multiple arguments are specified inside the parentheses, they are separated by commas: *function_name(argument1, argument2)*
- To identify the type of argument that we are specifying, we either enter the arguments in a particular order or include their names (without quotes) in our specification: *function_name(argument1, argument2)* or
  function_name(argument1_name = argument1,
  argument2_name = argument2)
- In this book, we follow the most common practices. We always specify required arguments first. If there is more than one required argument, we enter them in the order expected by R. We specify any optional arguments we want next and include their names so that R knows how to interpret them: *function_name(required_argument,*
  optional_argument_name = optional_argument)

USING R FUNCTIONS: To use a function in R, we typically write code in one of these two formats:

(a) *function_name*(*required_argument*)

(b) *function_name*(*required_argument*, *optional_argument_name* = *optional_argument*)

where:

- *function_name* is the name of the function; for example, "mean" is the name of the function that computes the mean of a set of values
- *required_argument* is the argument the function requires, such as the values we want to calculate the mean of; we typically do not include the names of required arguments; we enter the required arguments first, and if there is more than one, we enter them in the order expected by R
- , is a comma, which we use to separate different arguments
- *optional_argument_name* is the name of the optional argument we want to use, such as the argument that enables us to eliminate missing values before calculating a mean
- *optional_argument* is what we set the optional argument to be.

We will see some complex R functions in the next section. For now, let's look at a simple one. The function sqrt(), which stands for "square root," calculates the square root of the argument specified inside the parentheses. For example, to calculate the square root of 4, we run:

```
sqrt(4)
## [1] 2
```

sqrt() calculates the square root of the argument specified inside the parentheses. Example: sqrt(4).

TIP: Here, the name of the function is sqrt, which, as with all function names, is followed by parentheses (). Inside the parentheses, we need to specify the required argument, which is 4, in this case. The output of the executed code is 2. Indeed, the square root of four is two.

The output here is the number 2. Alternatively, given that the object *four* currently contains the number 4, we can run:

```
sqrt(four)
## [1] 2
```

Note that R will be able to execute the code above only after we have created the object *four*. If we start a new R session and attempt to run this code without having first created the object *four*, R will not be able to find the object in the environment and will give us an error message. This is just to say that one must run code in order. When returning to work on an R script, it is a good idea to run all the code from the beginning of the file up to the line that we are working on.

One of the major advantages of writing code using an R script (instead of writing it directly into the R console) is that we can always replicate our results by re-running the code we have written previously. Using an R script, we are able to work on complex problems that might require running hundreds or thousands of lines of code. As long as we save the code in an R script, we can keep tweaking and expanding it. Writing code in R scripts also means we can share our work and collaborate with others. Anyone with access to our R script will be able to replicate our analyses, which leads us to our next topic: the importance of annotating, or commenting, code.

is the character used to comment code. R ignores everything that follows this character until the end of the line. Example: # this is a comment.

It is good practice to comment code, that is, to include short notes to ourselves or to our collaborators explaining what the code does. This will make reading and understanding our code easier. To write comments in the R script, we use #. R ignores everything that follows this character until the end of the line and will not execute it. For example, running the following code produces exactly the same output as the code above:

```
sqrt(four) # calculates square root of four
## [1] 2
```

After seeing the # character, R stopped reading until the end of the line. Had we inserted the comment at the beginning of the line, before the code, R would not have executed the function sqrt() at all. Go ahead, run:

```
# calculates square root of four sqrt(four)
```

R will not produce an output. R thinks the whole line is a comment because it starts with #.

RStudio helps us write and read code by color coding it in the R script. For example, comments are shown in light green, while executable code is shown in black, gray, blue, and dark green. Becoming familiar with this color scheme will help you detect errors in your code. (In this book, we use only two colors when displaying code: cyan for executable code and gray for comments.)

1.7 LOADING AND MAKING SENSE OF DATA

TIP: We recommend that when you start a new study, you either (a) start a new R session (Session > New Session) or (b) remove all objects from the environment (Session > Clear Workspace > Yes) to avoid operating with objects from previous studies by mistake.

Before starting any analysis, we must load the dataset. Then, we must understand what the observations represent and what each of the variables means. In this section, we show how to do all of this for the data from the Project Student-Teacher Achievement Ratio (Project STAR) in preparation for the analysis in chapter 2. The goal of Project STAR was to examine the causal effects of small classes on student performance. While exploring the

data from Project STAR, we learn what variables are and how to distinguish between different types of variables based on their contents.

To follow along, you can create a new R script (as shown in the previous section) and practice typing the code yourself. Alternatively, you can open the "Introduction.R" file, which contains the code used in the remainder of this chapter.

TIP: How do we open an existing R script? In the RStudio dropdown menu, click on File > Open File... and then click on the ".R" file you want to open.

Here are the steps we recommend you follow before starting a data analysis:

❶ SET THE WORKING DIRECTORY

Before we can load a dataset, we need to direct R to the working directory, that is, the name and location of the folder containing the data. If you followed the advice from the earlier section, all the files necessary for the exercises in this book will be in the DSS folder on your Desktop.

The easiest way to set the working directory is to first save the R script to the folder that contains the dataset, the DSS folder, in this case. Then, you can use the dropdown menu to set the working directory manually: Session > Set Working Directory > To Source File Location. After your last click, you will see a line of code appear in the R console. Every time you start a new R session and want to work with a dataset saved in the DSS folder, you will need to run this line of code. You may, therefore, want to copy and paste it in the R script as your first line of code.

TIP: To save an R script to the DSS folder, either (a) click on File > Save As ... and select the DSS folder, or (b) manually drag the corresponding ".R" file from its current location to the DSS folder.

The code to set the working directory uses the function setwd(), which stands for "set working directory." The only required argument is the path to the folder, which should be in quotes because it is text and not the name of an object, the name of a function, the name of an argument, or a special value such as TRUE, FALSE, NA, and NULL. The path differs depending on whether you have a Mac or a Windows computer. The code to set the working directory to the DSS folder on your Desktop should resemble one of these (where *user* is your own username):

```
setwd("~/Desktop/DSS") # example of setwd() for Mac
setwd("C:/user/Desktop/DSS") # example for Windows
```

setwd() sets the working directory, that is, directs R to the folder on your computer where the dataset is saved. The only required argument is the path to the folder in quotes. Examples: setwd("~/Desktop/*folder*") for Mac, setwd("C:/*user*/Desktop/*folder*") for Windows (where *user* is your own username).

❷ LOAD THE DATASET

Now we are ready to load the dataset. R can read a variety of data formats. In this book, datasets are always provided in comma-separated values files, known as CSV files. As the name indicates, CSV files contain data separated by commas. (See figure 1.2 for a rendition of a CSV file.)

TIP: Resist the temptation to double-click on a CSV file. If you open a CSV file directly, you risk inadvertently changing or losing data.

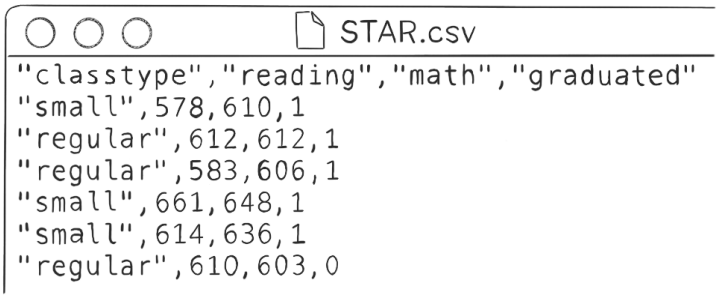


FIGURE 1.2. CSV files contain data separated by commas.

read.csv() reads CSV files. The only required argument is the name of the CSV file in quotes. Example: read.csv("*file.csv*").

To read the contents of a CSV file in R, we use the read.csv() function, which requires that we specify inside the parentheses the name of the CSV file in quotes. (We need to use quotes around the name of the CSV file because it is text and not the name of an object, the name of a function, the name of an argument, or a special value such as TRUE, FALSE, NA, and NULL.)

To store the dataset so that we can analyze it later, we need to not only read the CSV file but also save its contents as an object. We can do so by using the assignment operator <-. As we saw earlier, to the left of the assignment operator, we specify the name of the object. To its right, we specify the contents, which in this case are produced by reading the CSV file using the function read.csv().

Here, the dataset is in a file called "STAR.csv", and we choose to name the object where we store the dataset *star*. Putting it all together, the code to read and store the dataset is:

```
star <- read.csv("STAR.csv") # reads and stores data
```

After running the line of code above, the name of the object, *star*, should appear in the environment (the upper-right window in RStudio). If R gives you an error message instead, make sure that (i) you have set the working directory to the folder where the CSV file is saved, (ii) the name of the CSV file you are using in the code is exactly the same as the name of the CSV file saved in the working directory, and (iii) the extension of the CSV file in the working directory is indeed ".csv".

❸ UNDERSTAND THE DATA

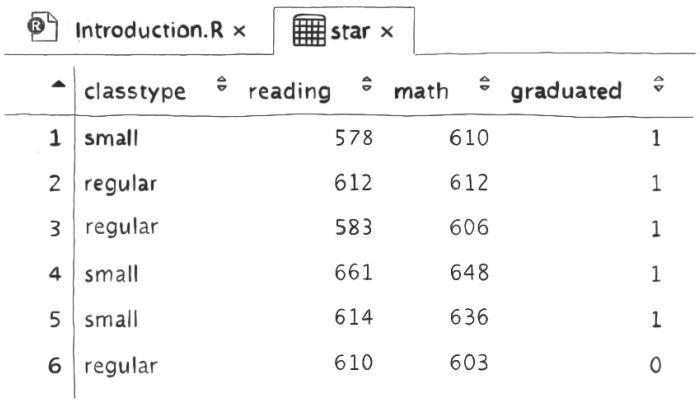
To make sense of the dataset, we should start by looking at its contents.

To look at the data, we could type the name of the object, *star*, in the R script and run it. R will show the entire contents of the dataset in the R console, which might be hard to read unless the dataset is small.

A better option is to use the function View(), which requires that we specify inside the parentheses the name of the object where the dataset is stored (without quotes). Alternatively, you can manually click on the object name in the environment. Both of these actions open a new tab in the upper-left window of RStudio with the dataset in spreadsheet form. We can then easily scroll up, down, left, and right to look at the data in an organized manner. Figure 1.3 shows how the data will be displayed if we run:

```
View(star) # opens a new tab with contents of dataset
```

View() opens a new tab in the upper-left window of RStudio with the contents of a dataset. The only required argument is the name of the object where the dataset is stored (without quotes). Example: View(*data*). Note that R is case-sensitive and the name of this function starts with uppercase V. The good news is this is the only function we will see in this book that uses any uppercase letters; all others are written in all lowercase letters.



	classtype	reading	math	graduated
1	small	578	610	1
2	regular	612	612	1
3	regular	583	606	1
4	small	661	648	1
5	small	614	636	1
6	regular	610	603	0

FIGURE 1.3. Tab that opens in the upper-left window of RStudio with entire contents of the *star* dataset as a result of either (a) running View(star) in the R script or (b) clicking on the object *star* in the environment. (To return to the R script, we can either close this tab by clicking on the gray X next to the name or by clicking on the R script tab.)

Sometimes it might be enough for us to see only the first few rows of data. For this purpose, we use the function head(), which requires that we specify inside the parentheses the name of the object where the dataset is stored (without quotes):

```
head(star) # shows the first six rows
##  classtype reading math graduated
## 1     small     578  610         1
## 2   regular     612  612         1
## 3   regular     583  606         1
## 4     small     661  648         1
## 5     small     614  636         1
## 6   regular     610  603         0
```

head() shows the first six rows or observations in a dataset. The only required argument is the name of the object where the dataset is stored (without quotes). To change the number of observations displayed, we use the optional argument n. Examples: head(*data*) shows the first six rows and head(*data*, n=3) shows the first three. In the output, the first column identifies the position of the observations, and the first row identifies the names of the variables.

By default, the function head() displays the first six lines of data. If we want R to display a different number of lines, we specify an optional argument inside the parentheses. In particular, we specify the argument named n and set it to equal the number of lines we want R to show. For example, to ask R to display the first three lines of *star*, we run:

```
head(star, n=3) # shows the first three rows
##  classtype reading math graduated
## 1     small     578  610         1
## 2   regular     612  612         1
## 3   regular     583  606         1
```

RECALL: In R functions, multiple arguments are separated by commas, and the names of arguments should not be in quotes. Failing to follow these instructions will prevent R from executing the code and result in an error message.

How do we make sense of the data inside the dataset? Knowing the following common features of datasets should help:

- Datasets capture the characteristics of a particular set of individuals or entities: citizens, organizations, countries, and so on. As we will soon learn, this dataset contains information about students who participated in Project STAR.

In a **dataframe**, each row is an observation, and each column is a variable. An **observation** is the information collected from a particular individual or entity in the study. The **unit of observation** of a dataset defines what each observation represents. The **notation** i identifies the position of the observation; the observation for which i=1 is the first observation. A **variable** captures the values of a changing characteristic for multiple individuals or entities.

- Datasets are typically organized as **dataframes**, where rows are observations and columns are variables.
  - What is an **observation**?
    - An observation is the information collected from a particular individual or entity in the study.
    - The **unit of observation** of the dataset defines the individuals or the entities that each observation in the dataframe represents. The unit of observation in the STAR dataset is students. Hence, every row of data in the *star* dataframe represents a different student in the study.
    - We usually refer to an observation by the row number in the dataframe, which we denote as i. See, for example, that in the output of head(), the rows are labeled by their position, i. When R displays the first six observations of a dataframe, the values of i range from 1 to 6.
  - What is a **variable**?
    - A variable captures the values of a changing characteristic for the multiple individuals or entities in the study.
    - Every column of data in the *star* dataframe is a variable; each variable captures a specific feature of the students, for all the students in the study.
    - We usually refer to a variable by its name. See, for example, that in the output of head(), the columns are labeled with the variable names: *classtype*, *reading*, *math*, and *graduated*. (Note that, for easy recognition, we italicize the names of variables in the text.)
    - From time to time, in this book we define new variables for the purpose of illustrating concepts. We represent a variable and its contents using the following mathematical notation:

$$X = \{10, 5, 8\}$$

TIP: In this book, we are also going to teach you mathematical notation, a system of symbols and expressions that represent mathematical concepts. To help you keep track of the meaning of these symbols and expressions, at the end of the book we have included an index of all the mathematical notation we use.

      - On the left-hand side of the equal sign, we identify the name of the variable: X, in this case.
      - On the right-hand side of the equal sign and inside curly brackets, we have the contents of the variable: multiple observations, separated by commas. In the simple example above, the variable contains three observations: 10, 5, and 8.

- To represent each individual observation of the variable X, we use X_i (pronounced X sub i) where i stands for the observation number. The subscript i means that we have a different value of X for each value of i. Here, because there are only three observations, i can equal only 1, 2, or 3. For example, we represent the second observation (the observation for which i=2) as X_2, which in this case equals 5.

Now that we have looked at the data, we should read the description of the variables provided in table 1.1.

TABLE 1.1. Description of the variables in the STAR dataset, where the unit of observation is students.

variable	description
classtype	class size the student attended: "small" or "regular"
reading	student's third-grade reading test scores (in points)
math	student's third-grade math test scores (in points)
graduated	identifies whether the student graduated from high school: 1=graduated or 0=did not graduate

Reading table 1.1, we learn that:

- *classtype* captures the size of the class the student attended, which was either "small" or "regular"
- *reading* records the scores, measured in points, the student earned on the third-grade reading test
- *math* records the scores, measured in points, the student earned on the third-grade math test
- *graduated* indicates whether the student graduated from high school (it equals 1 if the student graduated and 0 if the student did not graduate).

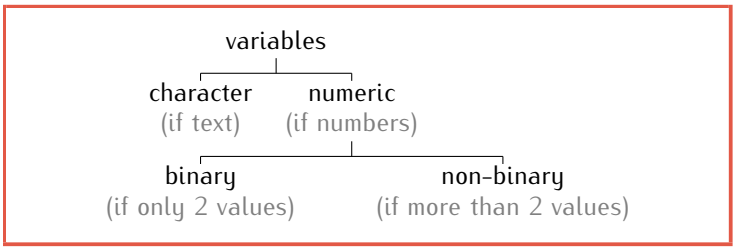
Now we can look at the first few lines of data again and substantively interpret them. For example, the first observation represents a student who attended a small class, earned 578 points on the third-grade reading test and 610 points on the third-grade math test, and graduated from high school.

❹ IDENTIFY THE TYPES OF VARIABLES INCLUDED

At this point, we should learn the typology of the variables included in the dataset. This information will be especially helpful when we need to interpret the results of the analysis.

Based on the contents of the variables, we can distinguish between the types listed in outline 1.3.

OUTLINE 1.3. Types of variables based on their contents.



A **character variable** contains text, such as *names*={Elena, Kosuke, Kathryn}. A **numeric variable** contains numbers, such as *rank*={2, 1, 3}. A **binary variable** can take only two values; in this book, we define binary variables as taking only 1s and 0s, such as *voted*={1, 0, 1}. A **non-binary variable** can take more than two values, such as *distance*={1.452, 2.345, 0.298} and *dice_roll*={2, 4, 6}.

- The first distinction we make is between character and numeric variables. While **character variables** contain text, **numeric variables** contain numbers. For example, in the STAR dataset, *classtype* is a character variable, and *reading*, *math*, and *graduated* are numeric variables.
- Among numeric variables, we differentiate between binary and non-binary. A **binary variable** can take only two values ("bi" means two). In this book, all binary variables take only 1s and 0s to represent the presence or absence of a particular trait. In this type of binary variable, also known as a dummy variable, you may think of the 1s as capturing the positive responses to simple yes or no questions and of the 0s as capturing the negative responses. For example, in the STAR dataset, *graduated* is a binary variable that captures responses to the question, did the student graduate? The variable takes the value of 1 when the answer is yes (the student graduated) and 0 when the answer is no (the student did not graduate). (See mathematical definition in the margin.)
- In contrast, we categorize as **non-binary** all other numeric variables, that is, those that can take more than two values. For example, in the STAR dataset, both *reading* and *math* are non-binary variables because they each contain more than two different numbers.

$$graduated_i = \begin{cases} 1 \text{ if student } i \text{ graduated} \\ 0 \text{ if student } i \text{ didn't graduate} \end{cases}$$

❺ IDENTIFY THE NUMBER OF OBSERVATIONS

dim() provides the dimensions of a dataframe. The only required argument is the name of the object where the dataframe is stored (without quotes). The output is two values: the first indicates the number of observations in the dataframe; the second indicates the number of variables. Example: dim(*data*).

Finally, we should find out how many observations the dataset contains. For this purpose, we use the function dim(), which stands for "dimensions" and requires that we specify inside the parentheses the name of the object where the dataframe is stored (without quotes). This function returns two values because dataframes have two dimensions: rows and columns. The first corresponds to the number of rows, which is equivalent to the number of observations. The second corresponds to the number of columns, which is equivalent to the number of variables.

```
dim(star) # provides dimensions of dataframe: rows, columns
## [1] 1274    4
```

Given the output above, we learn that the STAR dataset contains 1,274 observations. And, as we already knew by looking at the data directly, it contains four variables. Given that each observation represents a student, we now know that we have information for 1,274 students in Project STAR. In mathematical notation, we represent the number of observations in a dataframe as *n*. In this case, we can state that n=1,274.

The **notation** *n* stands for the total number of observations in a dataframe or in a variable.

1.8 COMPUTING AND INTERPRETING MEANS

The mean, or average, of a variable is one of the foundational concepts of data analysis. In this section, we first show how to access a variable inside a dataframe in R so that we can operate with its values. Then, we explain in detail how to calculate and interpret the mean of a variable.

TIP: In chapter 3, we will see how to calculate and interpret other statistics, such as the median, standard deviation, and variance of a variable.

1.8.1 ACCESSING VARIABLES INSIDE DATAFRAMES

Suppose we want to operate with the variable *reading* inside the dataframe *star*. How do we access the values within this variable?

If we run the name of the variable, *reading*, R will give us an error message informing us that the object *reading* cannot be found. Indeed, there is no object called *reading* in the environment. If instead we run the name of the object that contains the dataframe, *star*, R will show all the values in the dataframe, not just those of the variable *reading*.

To access the values of a single variable, we use the $ character. To its left, we specify the name of the object where the dataframe is stored (without quotes). To its right, we specify the name of the variable (without quotes). In this case, *star*$*reading* is the code that instructs R to select the variable *reading* from within the object named *star*. It is equivalent to saying to R: look inside of *star* and find the variable called *reading*. (Note that when writing code, we do not use quotes around the names of elements within an object, such as the names of variables within a dataframe.) Go ahead and run:

$ is the character used to access an element inside an object, such as a variable inside a dataframe. To its left, we specify the name of the object where the dataframe is stored (without quotes). To its right, we specify the name of the variable (without quotes). Example: *data*$*variable* accesses the variable named *variable* inside the dataframe stored in the object named *data*.

```
star$reading
##   [1] 578 612 583 661 614 610 595 665 616 624
##  [11] 593 599 693 545 565 654 686 570 529 582
##  ...
```

In your R console (the lower-left window), you should see all the observations of *reading*. Here, we show you only the first 20. We use an ellipsis—three dots—to signify that more observations should appear after those displayed here.

TIP: The number in brackets shown on the second line in your R console might not be 11 because the size of your lower-left window might be different than ours. The larger the window, the more observations R will be able to display per line, and the higher the number in brackets shown on the second line will be.

What are the numbers in brackets at the beginning of each line? They indicate the position of the observation immediately to the right. The [1] on the first line indicates that the number 578 is the first observation of the variable *reading* ($reading_1$=578). The [11] on the second line indicates that the number 593 is the 11th observation of *reading* ($reading_{11}$=593), and so on.

Now that we know how to access the contents of a variable, we can learn how to compute and interpret the mean of a variable.

1.8.2 MEANS

The **mean**, or **average**, of a variable equals the sum of the values across all observations divided by the number of observations.

The **mean**, or **average**, of a variable characterizes its central tendency. It equals the sum of the values across all observations divided by the number of observations. In mathematical notation, the mean of a variable is often represented by the name of the variable with a bar on top, like so:

$$\overline{name\ of\ the\ variable}$$

TIP: The mean of a variable is a single number, which does not vary by observation. As a result, the mean of X ($\overline{X}$) is not subscripted by i.

The formula to compute the mean of X is:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

where:

- $\overline{X}$ (pronounced X-bar) stands for the mean of X
- X_i (pronounced X sub i) stands for a particular observation of X, where i denotes the position of the observation
- n is the number of observations in the variable
- the symbol $\sum$ (the Greek letter Sigma) is the mathematical notation for summation; $\sum_{i=1}^{n} X_i$ stands for the sum of all X_i (observations of X) from i=1 to i=n, meaning from the first observation of the variable X to the last one.

For example, if X={10, 4, 6, 8, 22}, then n=5 because the variable has five observations, and the mean of X is:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} = \frac{X_1 + X_2 + X_3 + X_4 + X_5}{5}$$
$$= \frac{10 + 4 + 6 + 8 + 22}{5} = \frac{50}{5} = 10$$

mean() computes the mean of a variable. The only required argument is the code identifying the variable. Example: mean(*data*$*variable*).

To calculate the mean of a variable in R, we can use the function mean(). The only required argument is the code identifying the variable. For example, to calculate the mean of the reading test scores of the students in the STAR dataset, we run:

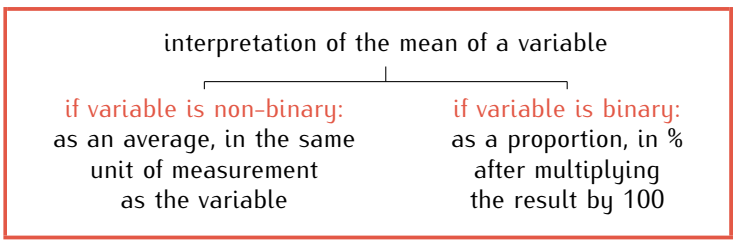
```
mean(star$reading) # calculates the mean of reading
## [1] 628.803
```

How shall we interpret this output? First, we need to figure out the quantity in which the value is measured. This is called the **unit of measurement**. When interpreting numeric results, you should make it clear whether the number is measured in points, percentages, miles, or kilometers, for example.

The **unit of measurement** is the quantity in which a value is measured. For example, depending on where you live, you might measure temperature in °F or °C and distance in miles or kilometers.

The unit of measurement of the mean of a variable depends on whether the variable is non-binary or binary. Outline 1.4 summarizes how to interpret the mean of a variable (including units of measurement) based on this distinction. (We exclude from this discussion categorical variables, whose means generally have no straightforward substantive interpretation.)

TIP: Categorical, or factor, variables take a fixed number of values, where each value represents a qualitative outcome. For example, we can capture adult education levels in a categorical variable where 1=no qualifications, 2=high school diploma, and 3=undergraduate degree.



OUTLINE 1.4. Interpretation of the mean of a variable based on the type of variable.

When the variable is non-binary, the mean should be interpreted in the same unit of measurement as the values in the variable. For example, in the output above, because *reading* is a non-binary variable measured in points, the mean of *reading* is also in points. We can, therefore, interpret the output as meaning that the students in Project STAR scored 629 points, on average, on the third-grade reading test.

TIP: It is good practice to round numeric results to meaningful decimal places. This usually means no more than two decimals, but often one or none.

When the variable is binary, the mean should be interpreted as a percentage, after multiplying the result by 100. Why? Because the mean of a binary variable is equivalent to the proportion of the observations that are 1s (that have the characteristic identified by the variable). Let's go over a simple example to see how this works. Suppose we want to calculate the mean of the first six observations of the binary variable *graduated*. As we saw earlier in the output of head(), these are {1, 1, 1, 1, 1, 0}. The average of these six observations would be:

$$\frac{\text{sum of observations}}{\text{number of observations}} = \frac{1+1+1+1+1+0}{6} = \frac{5}{6} = 0.8333\ldots$$

Notice that the fraction 5/6 is equivalent to the proportion of the observations that are 1s. (See TIP in the margin.) Now, to convert the result from decimal form (0.83) into a percentage, we multiply it by 100 (0.83×100=83%).

TIP: The proportion of observations that meet a criterion is calculated as:

$$\frac{\text{number of observations that meet criterion}}{\text{total number of observations}}$$

To interpret the resulting decimal as a percentage, we multiply it by 100. For example, if X={1, 1, 1, 1, 1, 0}, the proportion of observations in X that are 1s is 83% (5/6 = 0.8333... and 0.83×100=83%).

When the variable is binary, the numerator of the mean, the sum of the 1s and 0s, is equivalent to the number of observations that meet the criterion. As a result, the mean is equivalent to the proportion of observations in the variable that are 1s.

Putting it all together, we interpret the average of the first six observations of *graduated* as indicating that about 83% of the first six students in the STAR dataset graduated.

Now, let's compute the mean of all the observations within the binary variable *graduated* (rather than just the first six). We do so by running:

```
mean(star$graduated) # calculates the mean of graduated
## [1] 0.8697017
```

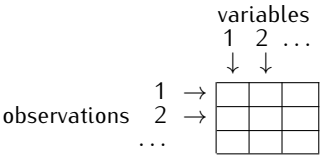
How shall we interpret this output? Since the variable is binary, the output means that about 87% of all the students in Project STAR received a high school diploma (0.87×100=87%).

1.9 SUMMARY

We began this chapter by providing an overview of the book and its main features. We then argued that knowing how to perform and evaluate data analyses is particularly useful for studying society and human behavior but can be helpful to anyone. These skills are also highly marketable in the current era of big data. We then got our computers ready and became acquainted with R and RStudio, the two programs we use. Finally, in preparation for the next chapter's data analysis, we learned how to load and make sense of data and how to compute and interpret means.

1.10 CHEATSHEETS

1.10.1 CONCEPTS AND NOTATION

concept/notation	description	example(s)
dataframe	structure of data in which each row is an observation and each column is a variable  	variables *classtype* and *reading* in dataframe form: i | *classtype* | *reading* 1 | small | 725 2 | regular | 692 3 | small | 725 ... | ... | ...
observation	information collected from a particular individual or entity in the study; each row in a dataframe is an observation	the first observation in the dataframe above represents a student who attended a small class and scored 725 points on the reading test
unit of observation	defines what each observation represents	students, schools, states, countries, ...
i	identifies the position of the observation	the observation for which i=3 is the third observation in a dataframe or in a variable
variable (X)	captures the values of a changing characteristic for multiple individuals or entities; each column in a dataframe is a variable	X={10, 8, 12}
character variable	variable that contains text	*names* = {Elena, Kosuke, Kathryn}
numeric variable	variable that contains numbers	*rank* = {2, 1, 3, 5, 4}
binary variable	variable that can take only two values, in this book 1s and 0s	*voted* = {0, 1, 1, 0, 1}
non-binary variable	variable that can take more than two values	*distance* = {2.568, 5.367, 7.235} *dice_roll* = {2, 5, 4, 3, 1}
n	stands for the total number of observations in a dataframe or in a variable	in the variable *dice_roll* above, n=5
$\sum$	Greek letter Sigma; mathematical notation for summation; $\sum_{i=1}^{n}$ means "sum what follows for all the observations, from i=1 to i=n" (the first to the last)	if X={10, 8, 12}, then: $\sum_{i=1}^{n} X_i = X_1 + X_2 + X_3$ $= 10 + 8 + 12 = 30$
unit of measurement	quantity in which a value is measured	°F, °C, miles, kilometers, points, percentages, percentage points, ...
mean or average of a variable ($\overline{X}$)	characterizes the central tendency of the variable; equals the sum of the values across all observations divided by the number of observations: $\text{mean of } X = \overline{X} = \frac{\sum_{i=1}^{n} X_i}{n}$ unit of measurement of $\overline{X}$: - if X is non-binary: in the same unit of measurement as X - if X is binary: in percentages, after multiplying the result by 100	if X={10, 8, 12}, then: $\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} = (X_1 + X_2 + X_3)/3$ $= (10 + 8 + 12)/3 = 30/3 = 10$

1.10.2 R SYMBOLS AND OPERATORS

code	description	example(s)
+ − * /	some of the arithmetic operators recognized by R	(4 − 1 + 3) * (2 / 3)
<-	assignment operator; creates new objects or overwrites existing ones (if an object with the same name already exists); to its left, we specify the name of the object (without quotes); to its right, we specify the contents of the object; the name of an object cannot begin with a number or contain spaces, $, or %	*object_name* <- *object_contents*
"	when writing code, the names of objects, names of functions, and names of arguments as well as special values such as TRUE, FALSE, NA, and NULL should not be in quotes; all other text should be in quotes; numbers should not be in quotes	"this is text" *object_name*
()	the names of functions are always followed by parentheses; inside the parentheses, we write the argument(s) of the function, separated by commas if there is more than one argument; we enter required arguments first and in the order expected by R; we specify optional arguments by including their names in the specification (without quotes)	*function_name*(*required_argument*) *function_name*(*required_argument*, *optional_argument_name* = *optional_argument*)
#	character used to comment code; R ignores everything that follows this character until the end of the line	*executable_code* # *comment*
$	character used to access an element inside an object, such as a variable inside a dataframe; to its left, we specify the name of the object where the dataframe is stored (without quotes); to its right, we specify the name of the variable (without quotes)	*data*$*variable* # accesses the variable named *variable* inside the dataframe stored in the object named *data*

1.10.3 R FUNCTIONS

function	description	required argument(s)	example(s)
sqrt()	calculates the square root	what we want to compute the square root of	sqrt(4)

setwd()	sets the working directory, that is, directs R to the folder on your computer where the dataset is saved	path to folder in quotes	setwd("~/Desktop/*folder*")
			# for Mac
			setwd("C:/*user*/Desktop/*folder*")
			# for Windows, where user is your own username

read.csv()	reads CSV files	name of CSV file containing the dataset in quotes	read.csv("*file.csv*")
View()	opens a tab with the entire contents of a dataframe	name of object where the dataframe is stored	View(*data*)
head()	shows the first six observations in a dataframe; in the output, observations are identified by their position, *i*, and variables by their names	name of object where the dataframe is stored optional argument n: changes the number of observations displayed	head(*data*) # shows first six observations head(*data*, n=2) # shows first two observations
dim()	provides the dimensions of a dataframe; output is: [1] number of rows or observations [2] number of columns or variables	name of object where the dataframe is stored	dim(*data*)
mean()	calculates the mean of a variable	code identifying the variable (see $)	mean(*data*$*variable*)

Index of Concepts

Index of Mathematical Notation

Index of R and RStudio