2 USING R TO DO DATA ANALYSIS

An important goal of this book is to introduce you to R, a programming language that facilitates data analysis. Everything you might want to do with data analysis requires that you use some sort of computer-based program to organize and evaluate the data. I think R is the right program for you to use, for a number of reasons. First, R is available for free. As long as you are able to access R remotely or can download it to your personal computing device, you can use it without paying. Among other things, this means that when you leave school, where you may have free or low-cost access to other programs, you can take R with you. In addition to being free, using R helps you develop some basic programming skills, which could be useful on the job market. Finally, by using R, you are more likely to understand how you are connected to the data you are using and to think a bit more systematically about the problems you are trying to solve.

One reason sometimes given for *not* using R is that there is a steep learning curve. Indeed, there is a bit of a learning curve with R, but the same is true for most statistical programs used in courses on data analysis, especially if students don't have prior experience with them. Whether using R, SPSS, Stata, SAS, or even Excel, students with no prior experience tend to struggle for the first few weeks of the semester. Also, some of the things that make R a bit more difficult to learn at the outset are also the things that provide added value from an educational perspective. Finally, it's all right for you to be challenged with something that is a bit different from what you are used to. That's one of the more valuable things about higher learning!

R is used throughout this book to generate all statistical results and graphics. In addition, as an instructional aid, in most cases, the code used to generate graphics and statistical results is provided in the text associated with those results. You can use the sample code to help you solve the R-based problems at the end of the chapters or as a guide to other problems and homework that might be assigned.

ACCESSING R

RStudio and Posit Cloud

One important tool that is particularly useful for new users of R is **RStudio**, a graphical user interface (GUI) that facilitates user-to-R interactions. I strongly recommend using the cloud-based version of RStudio, found at posit.cloud. This is a terrific online option for doing your work in the RStudio environment without having to download R and RStudio to your personal device. This is an especially attractive alternative if your personal computer is a bit out of date or underpowered. As long as you have reliable internet access, posit Cloud is the way to go, especially since, in doing so, you can avoid downloading and installing R and RStudio, a process that gives some students problems. Posit Cloud can be used by individuals for free or at a very low cost, and institutions can choose from a number of low-cost options to set up group workspaces where students can do their work. In addition, instructors who set up a class account on RStudio Cloud can take advantage of a number of options that facilitate working with their students on assignments.

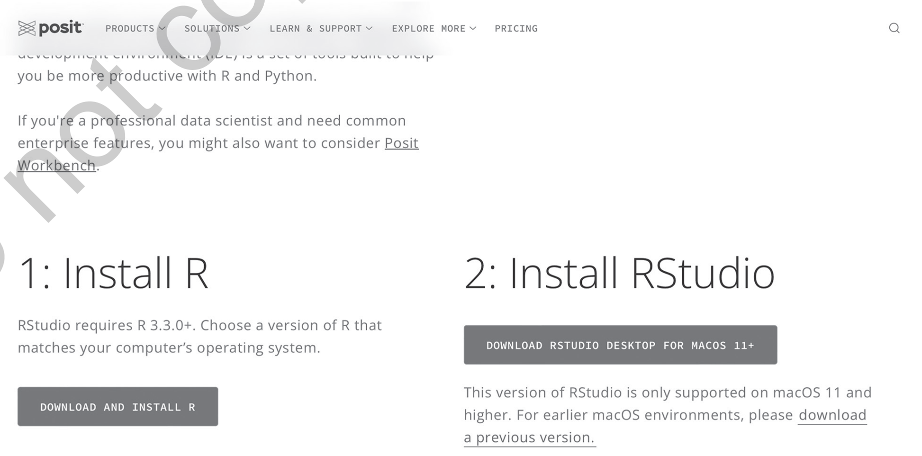
You don't need to download anything to use the cloud-based version of RStudio. Just go to posit Cloud and click on "Sign Up" if you do not have an account or "Log In" if you already have an account. Signing up for the "Cloud Free" account should provide you with the space and resources you need to do the work associated with this book. If you end up needing more resources, the "Cloud Student" option is available for just five dollars per month. If your instructor is able to set up "Cloud Instructor" plan, then you can sign up for the free plan and join a work space set up for your class.

Downloading R and RStudio

If using the cloud-based version of RStudio is not optimal for you, there are other options. If you have reliable access to computer labs on your college or university campus, there's a good chance that R and RStudio are installed and available for you to use. If so and if you don't mind having access only in the labs, then great, there's no need to download anything. That said, one of the nice things about using R is that you can download it (and RStudio) for free and have instant, anytime access on your own computer.

Downloading and installing R and RStudio is at once pretty straightforward and at the same time, a point in the process where new users encounter a few problems. First, go to the Posit.co website and click on the DOWNLOAD RSTUDIO button in the upper-right corner. This will take you to a new page, where you should scroll down to the RStudio Desktop section. Click on the DOWNLOAD RSTUDIO button again. This should take you to a page that looks like Figure 2.1.

FIGURE 2.1 ■ RStudio Download Links



development environment (IDE) is a set of tools built to help you be more productive with R and Python.

If you're a professional data scientist and need common enterprise features, you might also want to consider Posit Workbench.

1: Install R

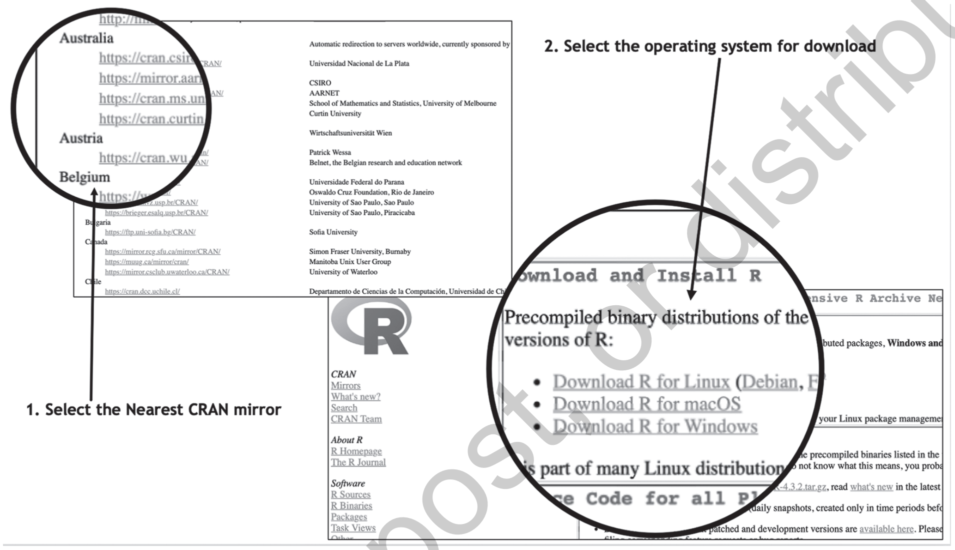
RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

2: Install RStudio

This version of RStudio is only supported on macOS 11 and higher. For earlier macOS environments, please download a previous version.

You need to install R before you can use RStudio, so click on the DOWNLOAD AND INSTALL R button. This will take you to a page on the Comprehensive R Archive Network (CRAN) that lists a collection of local links to CRAN mirrors (places from which to download R), arranged by country. You should select a CRAN that is geographically close to you. (Step 1 in Figure 2.2).

FIGURE 2.2 ■ Steps in Downloading R



Australia
https://cran.csiro...CRAN/ — Universidad Nacional de La Plata
https://mirror.aarn... — CSIRO
https://cran.ms.un...AN/ — AARNET
https://cran.curtin... — School of Mathematics and Statistics, University of Melbourne; Curtin University
Austria
https://cran.wu... — Wirtschaftsuniversität Wien
Belgium — Patrick Wessa; Belnet, the Belgian research and education network
https://...z.usp.br/CRAN/ — Universidade Federal do Parana; Oswaldo Cruz Foundation, Rio de Janeiro; University of Sao Paulo, Sao Paulo
https://brieger.esalq.usp.br/CRAN/ — University of Sao Paulo, Piracicaba
Bulgaria
https://ftp.uni-sofia.bg/CRAN/ — Sofia University
Canada
https://mirror.rcg.sfu.ca/mirror/CRAN/ — Simon Fraser University, Burnaby
https://muug.ca/mirror/cran/ — Manitoba Unix User Group
https://mirror.csclub.uwaterloo.ca/CRAN/ — University of Waterloo
Chile
https://cran.dcc.uchile.cl/ — Departamento de Ciencias de la Computación, Universidad de Ch...

Download and Install R

Precompiled binary distributions of the ... versions of R:

- Download R for Linux (Debian, F...
- Download R for macOS
- Download R for Windows

...is part of many Linux distribution...

Source Code for all Pl...

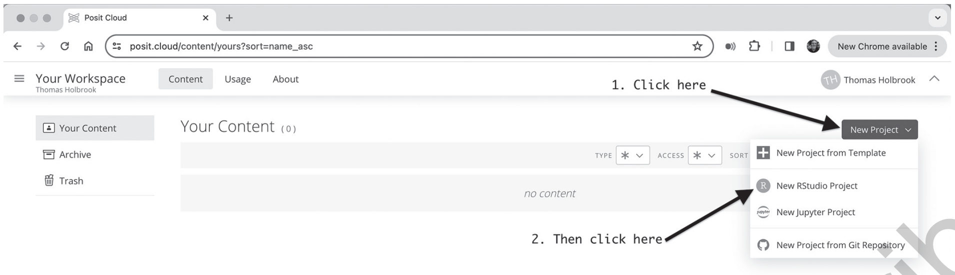
Once you choose a local CRAN, you will go to a page with separate links for the Linux, macOS, and Windows operating systems. Click on the link that matches the operating system on your computer (Step 2 in Figure 2.2), and you will be sent to a new page. If you are using a Windows operating system, you then click on Base, and you will be sent to a download link; click the download link and install the program. If you are using macOS, you will be presented with download choices based on your specific version of macOS; choose the download option whose system requirements match your operating system, and download as you would any other program from the web. For Linux, choose the distribution that matches your system and follow the installation instructions.

Downloading RStudio is much easier. Just click on the DOWNLOAD RSTUDIO DESKSTOP button (Step 2 in Figure 2.1, and RStudio will download directly to your desktop. Depending on the type of operating system you have, you may have an extra step of two left to install the program.

OPENING RSTUDIO

If you are using posit Cloud, when you log into your account, you will be sent to your posit Workspace, and that screen should look like the one in Figure 2.3. Click on the New Project tab and then on the New RStudio Project tab from the pull-down menu.

FIGURE 2.3 ■ Posit Cloud Screen After Login



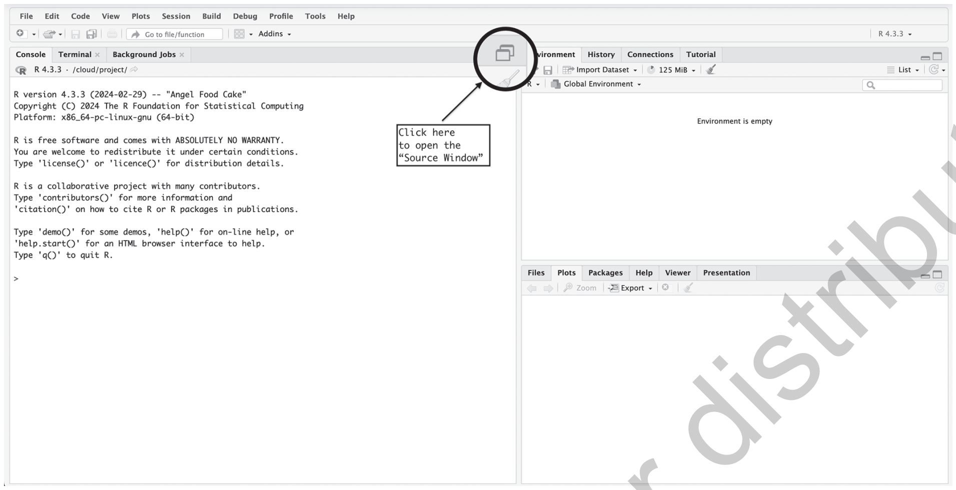
This will create a new unnamed project and will open the RStudio window. A project is sort of like a self-contained folder that contains things you use (data files, R packages, etc.) and things you create and save (script files, graphs, R output) while working in the project. Once you are in the RStudio window, you should replace "Untitled Project" with a project title, perhaps something like the course code for you class (e.g., PolSci390) or something else that is short, simple, and meaningful. Students tend to go a little overboard, creating new projects almost every week. I strongly suggest that you create just one project for the entire semester. The next time you open posit Cloud, your newly created project will be listed under "Your Content," and you just have to click on the project name to get back to your work.

If you've downloaded RStudio to your personal device, open it by clicking on the appropriate icon or file name. Initially, you should see three different RStudio windows that look something like the image in the first illustration that follows. It's possible that your screen will look a bit different due to differences in operating systems, the version of RStudio you downloaded, or some other reason, but generally, it should look like Figure 2.4.

The first thing you should do is click on the boxes in the upper-right corner of the **Console** window on the left side. This will open up an additional window, the **Source** window. Once you do this, your computer display should look like the image in Figure 2.5 (except for the annotations). Again, there may be some slight differences but not enough to matter for learning about the RStudio environment.

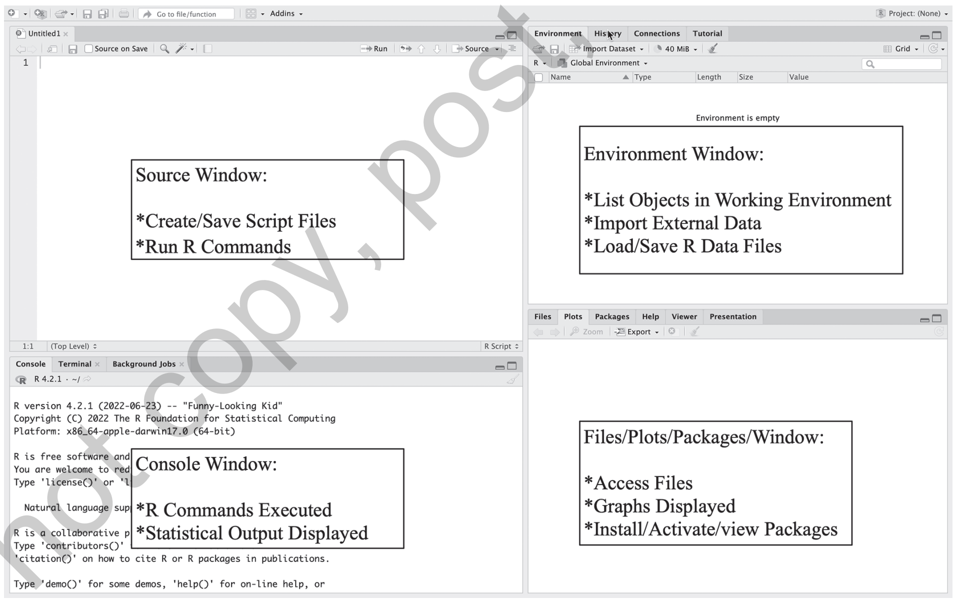
You will use the Source window a lot. In fact, almost everything you do will originate in the Source window. There is also a lot going on in the other windows but, generally, everything that happens (well, okay, *most* things) in the other windows happens because of what you do in the Source window. This is the window where you write your commands (the code that tells R what to do). Some people refer to this as the *Script* window because this is where you create the "script" file that contains your commands. Script files are very useful. They help you stay organized and can save you a lot of time and effort over the course of a semester or while you are working on a given research project.

FIGURE 2.4 ■ An Annotated Display of Initial RStudio Windows


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 4.3.3
Console Terminal Background Jobs
R 4.3.3 · /cloud/project/
Environment History Connections Tutorial
Import Dataset
125 MiB
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer Presentation
Zoom
Export
Click here to open the "Source Window"

```
R version 4.3.3 (2024-02-29) -- "Angel Food Cake"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.

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 2.5 ■ An Annotated Display of RStudio Windows



```
R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and
You are welcome to red
Type 'license()' or 'l

  Natural language sup

R is a collaborative p
Type 'contributors()'
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
```

Here's how the script file works. You start by writing a command at the cursor. You then leave the cursor in the command line and click on the "Run" icon (upper-right part of the window). Alternatively, you can leave the cursor in the command line and press `control(CNTRL) + return(Enter)` on your keyboard. When you do this, you will see that the command is copied and processed at the > prompt in the Console window. As indicated, what you do in the Source window controls what happens in the other windows. You control what is executed in the Console window by writing code in the Source window and telling R to run the code. You can also just write the command at the prompt in the Console window, but this tends to get messy, and it is easier to write, edit, and save your commands in the Source window.

Once you execute the command, the results usually appear in one of two places, either the Console window (lower left) or the Plots tab in the lower right window. If you are running a statistical function, the results will appear in the Console window. If you are creating a graph, it will appear in the Plots window. If your command has an error in it, the error message will appear in the Console window.

Anytime you get an error message, you should edit the command in the Source window until the error is corrected and make sure the part of the command that created the error message is deleted. Once you are finished running your commands, you can save the contents of the Source window as a *script* file, using the Save icon in the upper-left corner of the window. When you want to open the script file again, you can use the Open-Folder icon in the toolbar just above the Source window to see a list of your files. It is a good idea to give the script file a substantively meaningful name when saving it so you know which file to access for later uses.

Creating and saving script files is one of the most important things you can do to keep your work organized. Doing so allows you to start an assignment, save your work, and come back to it some other time without having to start over. It's also a good way to have access to any previously used commands or techniques that might be useful to you in the future. For instance, you might be working on an assignment from this book several weeks down the road and realize that you could use something from the first couple of weeks of the semester to help you with the assignment. You can then open your earlier script files to find an example of the technique you need to use. It is probably the case that most people who use R rely on this type of recovery process—trying to remember how to do something, then looking back through earlier script files to find a useful example—at least until they have acquired a high level of expertise. In fact, the writing of this textbook relied on this approach a lot!

The upper-right window in RStudio includes a number of different tabs, the most important of which for our purposes is the Environment tab, which provides a list of all objects you are using in your current session. For instance, if you import a new dataset, it will be listed here. Similarly, if you store the results of some analysis in a new object (this will make more sense later), that object will be listed here. There are a couple of options here that can save you from having to type in commands. First, the open-folder icon opens your working directory (the place where R saves and retrieves files) so you can quickly add any preexisting R-formatted datasets to your environment. You can also use the Import Dataset tab to import datasets from other formats (SPSS, Stata, SAS, Excel) and convert them to datasets that can be used in R. We will do this later in this chapter. The History tab keeps a record of all commands you have used in your R session. This is similar to what you will find in the script file you create, except that it tends to be a lot messier and includes lines of code that you don't need to keep, such as those that contain errors or multiple copies of commands that you end up using several times.

The lower-right window includes a lot of useful options. The Plots tab is probably the most often used tab in the lower-right window. This is where graphs appear when you run commands to create them. This tab also provides options for exporting the graphs. The Files tab includes a list of all files in your working directory. This is where your script file can be found after you've saved it, as well as any R-generated files (e.g., datasets, graphs) you've saved. If you click on any of the files, they will open in the appropriate window. The Packages tab lists all installed packages (usually bundles of R commands and datasets you can use–more on this later) and can be used to install new packages and activate existing packages. The Help tab is used to search for help with R commands and packages. It is worth noting that there is a lot of additional online help to be had with simple web searches. The Viewer tab is unlikely to be of much use to new R users.

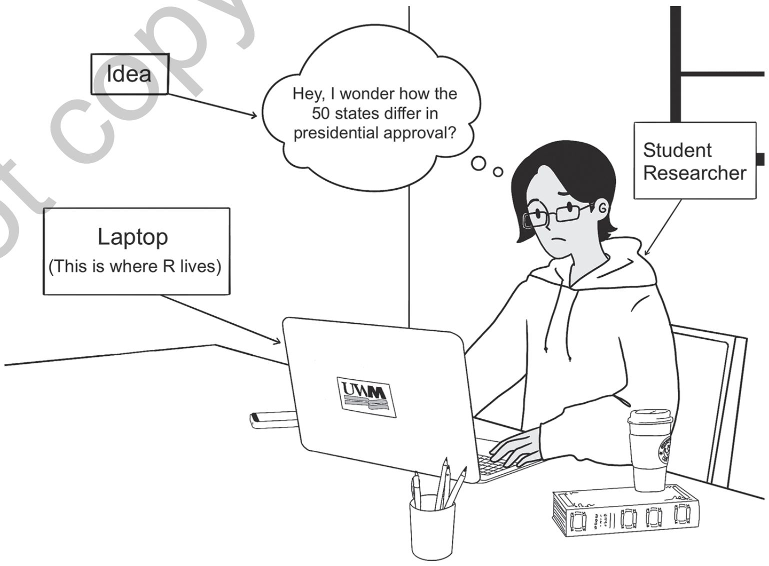
This overview of the RStudio setup is going to be most useful if you jump right in and get started. As with learning anything new, the best way to learn is to practice, make mistakes, and practice some more. There will be errors, and you can learn a lot by figuring out how to correct them.

UNDERSTANDING WHERE R (OR ANY PROGRAM) FITS IN

Before getting into the specific aspects of exactly how you tell R to do something, it is important to understand the relationship between you (the researcher), the data, and R. These connections may seem obvious to professors and experienced researchers or to students who have had a couple of research-based courses, but they are not so obvious to a lot of students, especially those with relatively little experience. For many students, there is a *black box* element to the research process—they're given some data and told what commands to use and how the results should be interpreted but without really understanding the process that ties things together. Understanding how things fit together is important because it helps reduce user anxiety and facilitates somewhat deeper learning.

The five panes that follow in the *R and Amazing College Student* comic strip (below) are a summary of how the researcher is connected to both R and the data and how these connections produce results. The first thing to realize is that R does things (performs operations) because you tell it to. Without your ideas and input R does nothing, so *you are in control*. It all starts with you, the researcher. Keep this in mind as you progress through this book.

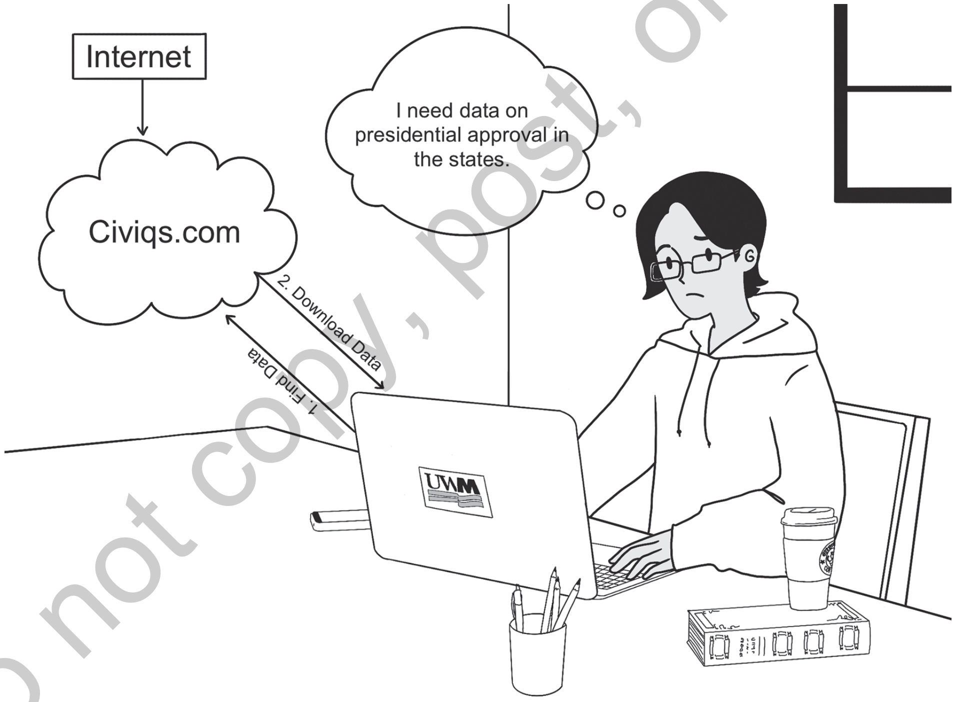
PANE 1 ■ You Are in Charge


Idea
Hey, I wonder how the 50 states differ in presidential approval?
Student Researcher
Laptop
(This is where R lives)
UWM

You, the college student, are sitting at your desk with your laptop, thinking about things that interest you (first pane). Maybe you have an idea about how two variables are connected to each other, or maybe you are just curious about some sort of statistical pattern. In this scenario, let's suppose that you are interested in how the states differ from each other with respect to presidential approval in the first few months of the Biden administration. No doubt, President Biden is more popular in some states than in others, and it might be interesting to look at how states differ from each other. In actuality, you are probably interested in discovering patterns of support and testing some theory about *why* some states have high approval levels and some states have low approval levels. For now, though, let's just focus on the differences in approval.

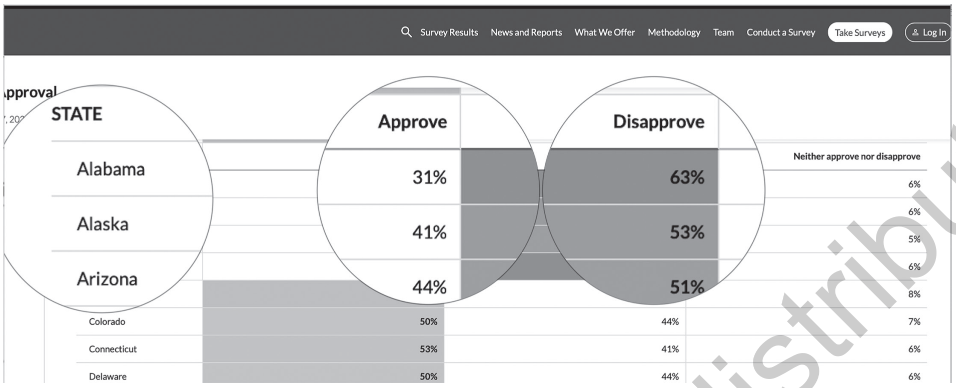
Odds are that you don't have this information at your finger tips, so you need to go find it somewhere. You're still sitting with your laptop, so you decide to search the internet for this information (second comic pane).

PANE 2 ■ Get Some Data



You find the data at civiqs.com, an online public opinion polling firm. After digging around a bit on the civiqs.com site, you find a page that has information that looks something like Figure 2.6.

FIGURE 2.6 ■ Presidential Approval Data From civiqs.com



STATE	Approve	Disapprove	Neither approve nor disapprove
Alabama	31%	63%	6%
Alaska	41%	53%	6%
Arizona	44%	51%	5%
			6%
			8%
Colorado	50%	44%	7%
Connecticut	53%	41%	6%
Delaware	50%	44%	6%

For each state, this site provides an estimate of the percentage who approve, disapprove, and who neither approve nor disapprove of the way President Biden is handling his job as president, based on surveys taken from January to June of 2021. This is a really nice table, but it's still a little bit hard to make sense of the data. Suppose you want to know which states have relatively high or low levels of approval? You could hunt around through this alphabetical listing and try to keep track of the highest and lowest states, or you could use a program like R to organize and process data for you. To do this, you need to download the data to your laptop and put it into a format that R can use (second comic pane). In this case, that means copying the data into a spreadsheet that can be imported to R (this has been done for you, saved as "Approve21.xlsx") or using more advanced methods to have R go right to the website and grab the relevant data (beyond the scope of what we are doing). Once the data are downloaded, you are ready to use R.

TIME TO USE R

Downloading and Importing Data

The first thing you should do is create a new folder on your personal computer, making sure that it is not buried very deep in the tangle of directories we all have on our computers. It might be helpful to put this folder on your desktop and give it short, meaningful name. Now, you need to **Download the Textbook Data Sets** to your new folder. Go to this book's page on Sagepub.com and look up the student resources, among which you will find the datasets (a collection of files with.rda and.xlsx extensions), and download all of these files into the folder you created. If you are working in posit Cloud, click on the Files tab in the lower-right window, then on the Upload tab, and go to the folder you created on your computer and upload all of the course files into posit Cloud. You will be using these files throughout the remaining chapters of this book.

From this point on, the process is all about you, the researcher, requesting things from R, with R providing responses. Hopefully, R will give you what you asked for, but it might also tell you that your request created an error. That's part of the process. The first step is to make sure the Source window in RStudio is open, so you can easily edit and save your commands from this session. If it is not open, you can reduce the size of the Console window, as instructed, or click on the Plus icon in the upper left corner and select "R Script."

Now, let's see how we can use R to take a closer look at the presidential approval data, which has motivated the student in Comic Pane 1. First things first, tell R to get the data. For demonstration purposes, we will import the Biden approval data we got from civiqs.com, stored as "Approve21.xlsx." In RStudio, you can do this by clicking the `Import Dataset` tab in the Environment/History window, then choose "From Excel," then click on "Browse" to find the file you want to open ("Approve21.xlsx"), and click on "Import."

If you do this, the following commands will be executed. Note that you may be prompted to install a missing package (`readxl`). If so, click "yes."

In segments like the following one, **italicized text that starts with # are comments or instructions to help you understand what's going on, and the boldface lines are commands that tell R what to do.**

```
#Tell R to install the `readxl` package (if you haven't already done so)
install.packages("readxl")

#Tell R to make "readxl" available for use
library(readxl)

#Read the Excel file into R, call it Approve21
Approve21 <- read_excel("Approve21.xlsx")

#Show the data set as a spreadsheet
View(Approve21)
```

You can also copy these commands into a script file and run them. If you do this and get an error message ("Error: 'path' does not exist: 'Approve21.xlsx'"), there is a problem with the file location. This topic is taken up at the end of the chapter, so for right now, just access the file using the `Import` tab as described above. If you still have a problem, check in with your instructor.

These lines of code use the `read_excel` command to convert the Excel file into a dataset that R can use. This new dataset is named `Approve21`. The first thing you will see after importing the data is a spreadsheet view of the dataset, which pops up in the Source window. Exit this window by clicking on the x on the right side of the Spreadsheet View tab. Some of the information we get with R commands that follow can also be obtained by inspecting the spreadsheet view of the dataset, but that option doesn't work very well for larger datasets.

At this point, in RStudio, you should see that the import commands have been executed in the Console (lower-left) window. Also, if you look at the Environment/History (upper-right) window, you should see that a dataset named `Approve21` (the civiqs.com data) has been added. This means that this dataset is now available for you to use. If you don't see these things or if you get an error message, check in with your course instructor before proceeding.

In the remainder of this chapter, you will use a set of simple commands to examine the presidential approval data. R commands usually tell R to do something to the entire dataset or to a particular variable from the dataset. The first couple of commands used in the following section focus on the entire dataset.

Examine the Dataset

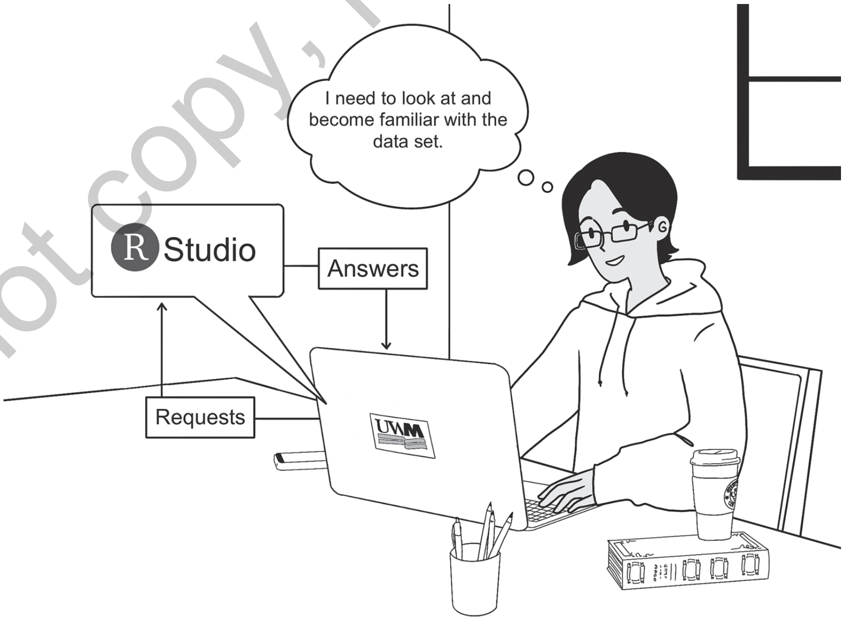
Usually, before jumping into producing graphs or statistics, we want to examine the dataset to become more familiar with it. This is also an opportunity to make sure there are no obvious problems with the data. First, let's check the dimensions of the dataset (how many rows and columns).

```
#Remember, `Approve21` is the name of the data set
#Get dimensions of the data set

dim(Approve21)
[1] 50 5
```

Ignoring "[1]," which just tells you that this is the first line of output, the first number produced by `dim()` is the number of rows in the dataset, and the second number is the number of columns. The number of rows is the number of cases or observations, and the number of columns is the number of variables in the dataset. According to these results, this dataset had 50 cases and five variables. (If you look in the Environment/History window, you should see this same information listed with the dataset as "50 obs. of 5 variables.")

Let's take a second to make sure you understand what's going on here. As illustrated in the comic pane that follows, you sent requests to R in RStudio ("Hey R, use `Approve21` as the

PANE 3 ■ Interact With R Through Rstudio



dataset, and tell me its dimensions"), and R sent information back to you. Check in with your instructor if this doesn't make sense to you.

Do 50 rows and 5 columns make sense, given what we know about the data? It makes sense that there are 50 rows, since we are working with data from the states, but if you compare this to the original civiqs.com data in Figure 2.6, it looks like there is an extra column in the new dataset. To verify that the variables and cases are what we think they are, we can tell R to show us some more information about the dataset. First, we can have R show the names of the variables (columns) in the dataset.

```
#Get the names of all variables in "Approve21"
names(Approve21)
[1]  "state"    "stateab"    "Approve"    "Disapprove"    "Neither"
```

Okay, this is helpful. Now we know that the data in the first and second columns contain some sort of state identifiers, and it looks like the last three columns contain data on the percentage of who approve, disapprove, or neither approve or disapprove of President Biden. If you compare this to the columns in Figure 2.6, you will note that what appears to be a state identifier, `stateab`, was added to the dataset after the data were downloaded from civiqs.com.

Now, suppose we want to get information about specific variables in the dataset. When using both the `dim()` and `names()` commands, we asked R to give us specific information about the dataset named `Approve21`. Generally, when asking R to do something with a specific variable (column) from a dataset, we follow the format `DatasetName$VariableName` to identify the variable, separating the dataset and variable name with a dollar sign. Following, we use the `class()` command to get information about the level of measurement of a couple of variables.

```
# Get the "class" of a few variables.
class(Approve21$state)
[1] "character"
class(Approve21$stateab)
[1] "character"
class(Approve21$Approve)
[1] "numeric"
```

This confirms that `state` and `stateab` are character (non-numeric) variables and `Approve` is numeric, as expected (go ahead and run this command for the remaining variables!). This is an important step because it can help flag certain types of errors. If there had been a coding error in which a stray character or symbol was entered in place of a neighboring number on the keyboard (e.g., 3o or 3p instead of 30) for `Approve`, then it would have shown up here as a character variable instead of the expected numeric variable.

We don't know from this output if the dataset uses a state abbreviation or the full state names to identify the cases, and we also don't know if `Approve`, `Disapprove`, and `Neither` are expressed as percentages or proportions. So, we need to take a closer look at the data. One useful pair of commands, `head()` and `tail()`, can be used to look at the first and last several rows in the dataset, respectively.

```
#List the first several rows of data
head(Approve21)

# A tibble: 6 × 5
   State       stateab  Approve  Disapprove  Neither
   <chr>       <chr>      <dbl>       <dbl>    <dbl>
1  Alabama     AL            31          63        6
2  Alaska      AK            41          53        6
3  Arizona     AZ            44          51        5
4  Arkansas    AR            31          63        6
5  California  CA            57          35        8
6  Colorado    CO            50          44        7
```

Here, you see the first six rows of the dataset and learn that `state` uses full state names, `stateab` is the state abbreviation, and the other variables are expressed as whole numbers. Note that this command also gives you information on the level of measurement for each variable: "chr" means the variable is a character variable, and "dbl" means that the variable is a numeric variable that could have a decimal point (though none of these variables do).

We can also look at the bottom six rows.

```
#List the last several rows of data
tail(Approve21)

# A tibble: 6 × 5
   State          stateab Approve  Disapprove  Neither
   <chr>          <chr>     <dbl>       <dbl>    <dbl>
1  Vermont        VT           58          36        7
2  Virginia       VA           48          46        6
3  Washington     WA           53          39        7
4  West Virginia  WV           23          72        5
5  Wisconsin      WI           47          48        5
6  Wyoming        WY           26          68        5
```

If we pay attention to the values of `Approve` and `Disapprove` in these two short lists of states, we see quite a bit of variation in the levels of support for President Biden in his first six months in office. Not Surprisingly, the president was fairly unpopular in conservative places, such as Alabama, Arkansas, West Virginia, and Wyoming, and fairly popular in more liberal states, such as California, Vermont, and Washington. However, because the dataset is ordered alphabetically by state name, it is hard to get a more general sense of what types of states give President Biden high or low marks for how he is handling his job. You could "eyeball" the entire dataset and try to keep track of states that are relatively high or low on presidential approval, but that is not very efficient and is probably prone to some level of error.

To get a listing of the lowest and highest approval states, we still use the `head()` and `tail()` but add the `order()` command to list the states in order of approval rather than alphabetically, by state. The following command is a bit advanced, so I'll break it down. In broad terms, the right side of the arrow is telling R to take the Approve21 data and reorder the rows so they are listed in order of the values on `Approve`. The square brackets `[ ]` are usually used to subset a dataset, and here they are saying to use the `order` command to sort the `Approve21` dataset by the values of the `Approve` variable. Let's see how this changes the head and tail of the dataset (if you type this into your computer, make sure everything is exactly the same).

```
#Sort by "Approve" and copy over original data set
Approve21 <- Approve21[order(Approve21$Approve),]

#list the first several rows of data, now ordered by "Approve"
head(Approve21)
# A tibble: 6 × 5
   State          stateab  Approve  Disapprove  Neither
   <chr>          <chr>      <dbl>       <dbl>    <dbl>
1  West Virginia  WV            23          72        5
2  Wyoming        WY            26          68        5
3  Oklahoma       OK            29          65        6
4  Idaho          ID            30          65        5
5  Alabama        AL            31          63        6
6  Arkansas       AR            31          63        6
```

This shows us the first six states in the dataset, those with the lowest levels of presidential approval. Our initial impression from the alphabetical list was verified: President Biden is given his lowest marks in states that are usually considered very conservative. There is also a bit of a regional pattern, as these are all Southern or Mountain West states.

Now, let's look at the states with the highest approval levels by looking at the last six states in the dataset. Notice that you don't have to sort the dataset again because the earlier sort changed the order of the observations until they are sorted again for some reason.

```
#list the last several rows of data, now ordered by "Approve"
tail(Approve21)

# A tibble: 6 × 5
   State          stateab  Approve  Disapprove  Neither
   <chr>          <chr>      <dbl>       <dbl>    <dbl>
1  California     CA            57          35        8
2  Rhode Island   RI            57          37        7
```

```
3  Vermont        VT          58         36         7
4  Maryland       MD          59         33         8
5  Hawaii         HI          61         33         6
6  Massachusetts  MA          63         30         7
```

Again, our initial impression is confirmed. President Biden gets his highest marks in states that are typically considered fairly liberal states. There is also an East Coast and Pacific West flavor to this list of high approval states.

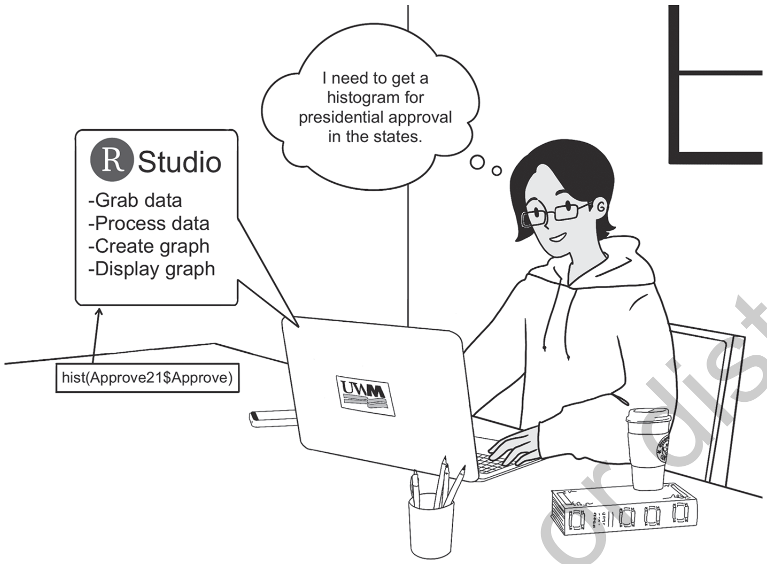
It is much neater and less distracting to show just the variables of interest, in this case `state` and `Approve`, instead of the entire dataset. To do this, you can modify the `head` and `tail` commands to designate that certain columns be displayed (shown below with `head`). The language within the brackets, `c('state', 'Approve')`, is used to tell R to combine the columns headed by `state` and `Approve` from the dataset `Approve21`.

```
#list the first several rows of data for "state" and "Approve"
head(Approve21[c('state', 'Approve')])

# A tibble: 6 × 2
   State          Approve
   <chr>            <dbl>
1  West Virginia       23
2  Wyoming             26
3  Oklahoma            29
4  Idaho               30
5  Alabama             31
6  Arkansas            31
```

Get a Graph

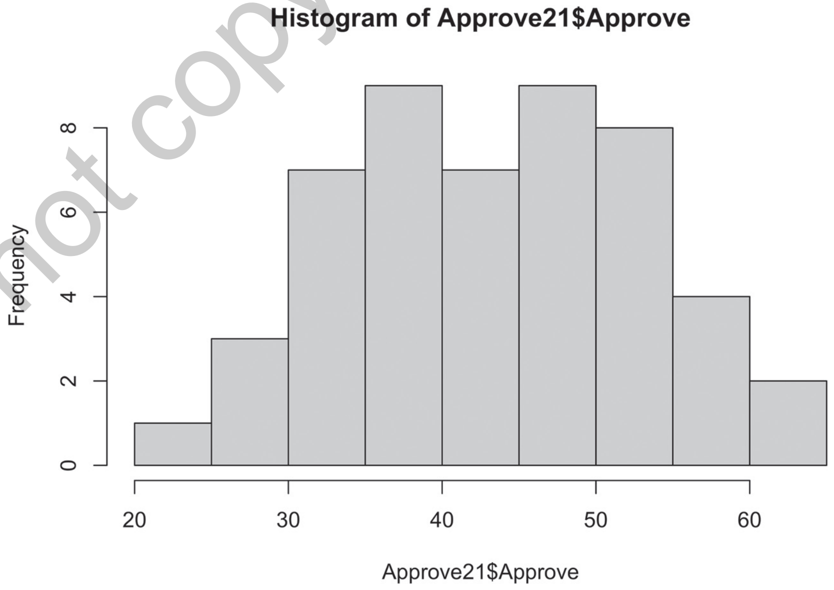
The preceding commands helped you get more familiar with the shape of the dataset, the nature of the variables, and the types of states at the highest and lowest end of the distribution; but it would be nice to have a more general sense of the variation in presidential approval among the states. For instance, how are states spread out between the two ends of the distribution? Are they clustered in the middle, or are they spread out more evenly. You will learn about a number of statistical techniques that provide information about the general tendency of the data, but graphing the data as a first step can also be very effective. **Histograms** are a particularly helpful type of graph for this purpose. To get a histogram, you need to tell R to get the dataset, use a particular variable, and summarize its values in the form of a graph (as in Pane 4).

PANE 4 ■ Tell R You Want a Histogram



The following very simple command will produce a histogram.

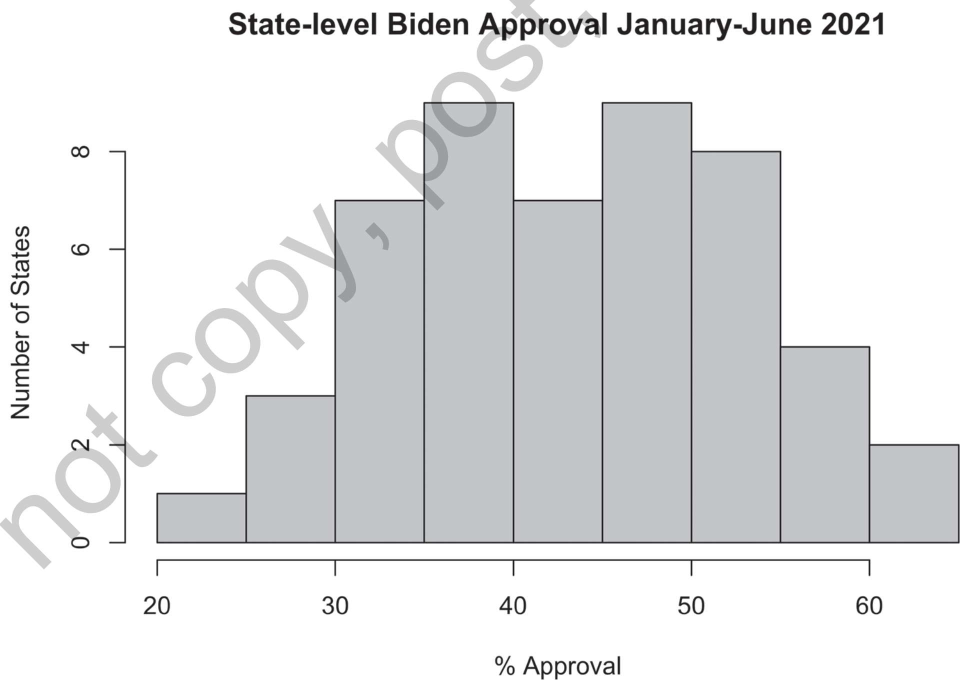
```
#This command tells R to use the data in the "Approve" column
# of the "Approve21" data set to make a histogram.

hist(Approve21$Approve)
```



Here, the numbers on the horizontal axis represent approval levels, and the values on the vertical axis represent the number of states in each category. Each vertical bar represents a range of values on the variable of interest. You'll learn more about histograms in the next few chapters, but you can see in this graph that there is quite a bit of variation in presidential approval, ranging from 20%–25% at the low end to 60%–65% at the high end, with the most common outcomes falling between 35% and 55%.

One of the important things you can do in R is modify the appearance of graphs to make them look a bit nicer so they are easier to understand. This is especially important if the information is being shared with people who are not as familiar with the data as the researcher is. In this case, you modify the graph title (`main`), the horizontal (`xlab`), and the vertical (`ylab`) labels.

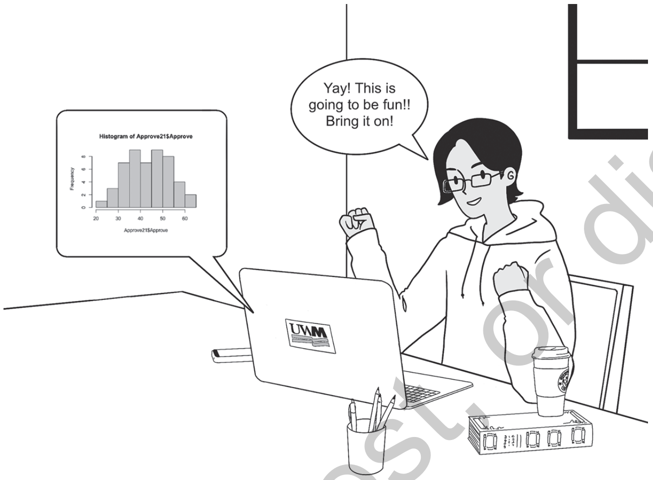
```
# Axis labels in quotes and commands separated by commas
hist(Approve21$Approve,
     main = "State-level Biden Approval January-June 2021", #Graph title
     ylab = "Number of States", #vertical axis label
     xlab = "% Approval") #Horizontal axis label
```



Note that all of the new labels are in quotes and the different parts of the command are separated by commas. You will get an error message if you do not use quotes and commas appropriately.

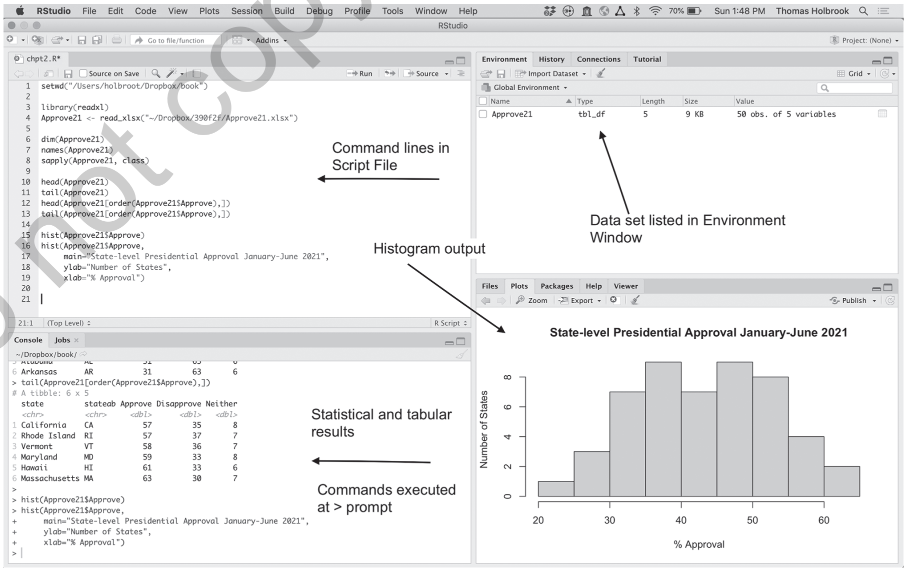
Making these slight changes to the graph improved its appearance and provided additional information to facilitate interpretation. The cartoon student asked R for a histogram of a specific variable from a specific dataset that they had previously told R to load, and R gave them this nice-looking graph. That, in a nutshell, is how this works. The cartoon student is happy that they have learned a lot, and they are ready to learn more!

PANE 5 ■ Making Progress


Yay! This is going to be fun!! Bring it on!
Histogram of Approve21$Approve
Frequency
Approve21$Approve
UWM

Before moving on, let's check in with RStudio to see what the various windows look like after running all of the commands listed previously (Figure 2.7). Here, you can see the script file

FIGURE 2.7 ■ The RStudio Windows After Running Several Commands



```
setwd("/Users/holbroot/Dropbox/book")

library(readxl)
Approve21 <- read_xlsx("~/Dropbox/390f2f/Approve21.xlsx")

dim(Approve21)
names(Approve21)
sapply(Approve21, class)

head(Approve21)
tail(Approve21)
head(Approve21[order(Approve21$Approve),])
tail(Approve21[order(Approve21$Approve),])

hist(Approve21$Approve)
hist(Approve21$Approve,
     main="State-level Presidential Approval January-June 2021",
     ylab="Number of States",
     xlab="% Approval")

```

```
6 Arkansas     AR          31         63       6
> tail(Approve21[order(Approve21$Approve),])
# A tibble: 6 x 5
  state        stateab Approve Disapprove Neither
  <chr>        <chr>     <dbl>      <dbl>   <dbl>
1 California   CA           57         35       8
2 Rhode Island RI           57         37       7
3 Vermont      VT           58         36       7
4 Maryland     MD           59         33       8
5 Hawaii       HI           61         33       6
6 Massachusetts MA          63         30       7
>
> hist(Approve21$Approve)
> hist(Approve21$Approve,
+      main="State-level Presidential Approval January-June 2021",
+      ylab="Number of States",
+      xlab="% Approval")
> 
```

RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help
Sun 1:48 PM Thomas Holbrook
chpt2.R*
Source on Save
Run
Source
21:1 (Top Level) R Script
Console Jobs
~/Dropbox/book/
Environment History Connections Tutorial
Import Dataset
Global Environment
Name Type Length Size Value
Approve21 tbl_df 5 9 KB 50 obs. of 5 variables
Files Plots Packages Help Viewer
Zoom Export Publish
State-level Presidential Approval January-June 2021
Number of States
% Approval
Command lines in Script File
Histogram output
Data set listed in Environment Window
Statistical and tabular results
Commands executed at > prompt

(named "chpt2.R") that includes all of the commands previously run. This file is now saved in the working directory and can be used later. The Environment window lists the `Approve21` dataset, along with a little bit of information about it. The Console window shows the commands that were executed, along with the results for the various data tables we produced. Finally, the Plot window in the lower-right corner shows the histogram produced by the histogram command.

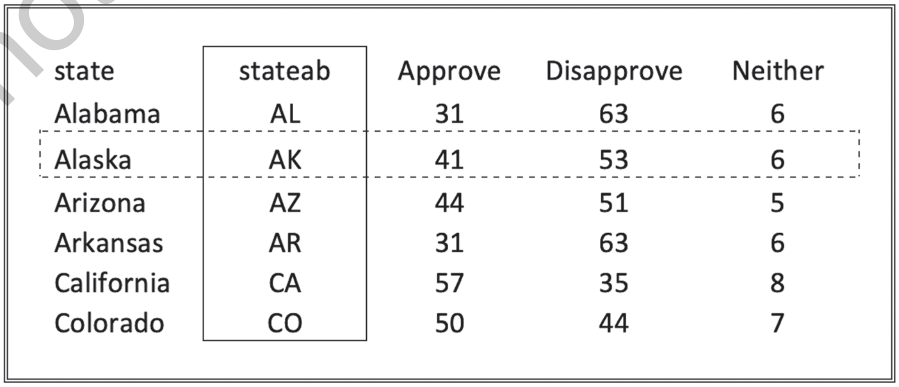
SOME R TERMINOLOGY

Now that you have already used R a bit, we need to cover a bit of terminology that sheds light on how R operates. It is no accident that this discussion takes place after you have already been using R. From a learning perspective, it is important to have a bit of hands-on experience with the tools you will be using before getting into definitions and terminology related to those tools. Without walking through the process of using R to produce results, much of the following discussion would be too abstract and you might have trouble making the connections between terminology and actually using R. As you will see, we have already been using many of the terms discussed.

Data Frames

One of the most important things to understand is the concept of a **data frame**. A data frame is a collection of columns and rows of data, as depicted in Figure 2.8. Columns in a data frame represent the values of the variables we are interested in. The column headings are the variable names we use when telling R what to do. In Figure 2.8, the highlighted column (surrounded by a solid line) contains the state abbreviations. If you want R to use the state abbreviation, you will refer to the column as `stateab`. The rows represent the individual cases or observations. The highlighted row below (surrounded by the dashed line) includes information from all five columns. Although we know the second row includes outcomes for Alaska, we wouldn't normally refer to this as "row Alaska." Instead, from R's perspective, this is "row 2" (the top row in a data frame is not considered row 1, as it is where the column names are stored). This collection of rows and columns together constitute a data frame (surrounded by the solid double line). For our purposes, the terms data frame and dataset are used interchangeably.

FIGURE 2.8 ■ An Illustration of a Data Frame



state	stateab	Approve	Disapprove	Neither
Alabama	AL	31	63	6
Alaska	AK	41	53	6
Arizona	AZ	44	51	5
Arkansas	AR	31	63	6
California	CA	57	35	8
Colorado	CO	50	44	7

One of the examples shown earlier used the `names(Approve21)` command to get the names of all five columns (variables). In this case, we told R to do something with the entire dataset. However, when getting the class of selected variables, sorting the dataset, and producing the histogram, we told R to use just one column from the dataset. We specified that R should go to `Approve21` and get information from the column headed by `Approve`, and we communicated this as `Approve21$Approve`.

Sometimes, it can hard to connect with and understand the different elements of a data frame. To make these things a bit more concrete and hopefully more understandable, I assemble part of this dataset, using the `c()` function in R to combine data points into vectors (lists of values that can be stored together) representing individual variables, and then join all of those vectors using `data.frame` to create a dataset.

Let's use the first four rows of data from Figure 2.8 to demonstrate this. The first variable is "state," so we can begin by creating a vector named state that includes the first four state names in order.

```
#create vector of state names and store as "state"
state <- c("Alabama", "Alaska", "Arizona", "Arkansas")

#Show state names
state
[1] "Alabama"   "Alaska"   "Arizona"   "Arkansas"
```

As you see, the vector named `state` has the first four state names.

Now, let's do this for the first four observations of the remaining columns.

```
#create vector of state abbreviations and store as "stateab"
stateab <- c("AL", "AK", "AZ", "AR")

#Show state abbreviations
stateab
[1] "AL" "AK" "AZ" "AR"

#create vector of state abbreviations and store as "Approve"
Approve <- c(31, 41, 44, 31)

#Show approval values
Approve
[1] 31 41 44 31

#create vector of state abbreviations and store as "Approve"
Disapprove <- c(63,53,51,63)

#Print (show) disapproval values
Disapprove
[1] 63 53 51 63

#create vector of state abbreviations and store as "Neither"
Neither <- c(6,6,5,6)

#Print (show) neither values
Neither
[1] 6 6 5 6
```

As you can see, all of the information from the first four rows of the approval dataset are now represented by five different vectors. To create a data frame, we need to stitch these vectors together, using the `data.frame` function.

```
#Combine separate vectors into a single data frame
df <- data.frame(state, stateab, Approve, Disapprove, Neither)
#Show the data frame
df
      state  stateab  Approve  Disapprove  Neither
1   Alabama       AL       31          63        6
2    Alaska       AK       41          53        6
3   Arizona       AZ       44          51        5
4  Arkansas       AR       31          63        6
```

Now you have a small data frame named `df` with the same information found in the first few rows of data in Figure 2.8. Hopefully, this illustration helps you connect to the concept of a data frame. Fortunately, researchers don't usually have to input data in this manner, but it is a helpful tool to use here to illustrate the concept of a data frame.

As a quick aside, when we imported the `Approve21` dataset, R created a "tibble," which is a particular type of data frame, labeled by R as `tbl_df` (for tibble data frame). The classic R dataset is usually identified as a `data.frame` and has some features and characteristics that are different from a `tibble` dataset. We ended up with a `tbl_df` because the default for the Import Dataset tab in the Environment window is to create a tibble when importing data. This distinction is not something you need to worry about, and it is only mentioned here so you don't get confused if you come across a `tbl_df` instead of a `data.frame`.

Objects and Functions

Everything in R is an **object**. I know that doesn't sound very useful, but it is true. Everything you've seen in this chapter is an object; the data frames, the variables inside the data frames, the vectors we created, the histogram, and the commands we used to tell R what to do. These things are all objects. **Functions** are a particular type of object. You can think of functions as commands that tell R what to do. Anything R does, it does with a function, and those functions tell R to do something with an object.

As an example, the code we used to create the histogram also includes both a function and an object.

```
hist(Approve21$Approve,
     main = "State-level Biden Approval January-June 2021",
     xlab = "% Approval")
```

Here, `hist` is a function call (telling R to use a function called "hist"), and `Approve21$Approve` is an object. The function `hist` is performing operations on the object

Approve21$Approve. Other "action" parts of the function are main, xlab, and ylab, which are used to create the main title and the labels for the *x* axis, and the *y* axis.

We also just used two functions (c and data.frame) to create six objects: the five vectors of data (state, stateab, Approve, Disapprove, and Neither) and the data frame where they were joined (df).

Storing Results in New Objects

One of the really useful things you can do in R is create new objects, either to store the results of some operation or to add new information to existing datasets. For instance, at the very beginning of the data example in this chapter, we imported data from an Excel file and stored it in a *new object* named Approve21 using the following command.

```
Approve21 <- read_excel("Approve21.xlsx")
```

Here and in multiple places throughout this chapter, I used <- to tell R to put the results of some operation into a new object, usually a dataset or a variable within a dataset. Pay close attention to this, as you will use the <- a lot. This is a convention in R that I think is helpful because it is literally pointing at the new object and saying to take the result of the right side action and **assign** it to the object on the left. We also updated an object when we sorted Approve21 and replaced the original data with sorted data by level of approval, as shown next.

```
Approve21 <- Approve21[order(Approve21$Approve),]
```

We can also modify existing datasets by creating new variables based on **transformations of existing variables** (more on this in Chapter 4). For instance, suppose that you want to express presidential approval as a proportion instead of a percentage. You can do that by dividing Approve21$Approve by 100, but you don't want to replace the original variable, so the results are stored in a new variable, Approve21$Approve_prop, which is added to the original dataset.[1]

```
#Calculate and add "Approve_prop" to Approve21
Approve21$Approve_prop <- Approve21$Approve/100
```

You can check with the names function to see if the new variable is now part of the dataset.

```
names(Approve21)
[1]   "state"            "stateab"   "Approve"   "Disapprove"   "Neither"
[6]   "Approve_prop"
```

The new object is listed as the sixth variable in the dataset. You can also look at its values to make sure they are expressed as proportions rather than percentages. Typing just object name at the > prompt will generate a listing of all values for that object.

```
Approve21$Approve_prop
[1] 0.23 0.26 0.29 0.30 0.31 0.31 0.31 0.32 0.33 0.34 0.35 0.36 0.36 0.36 0.37
[16] 0.37 0.37 0.39 0.39 0.39 0.41 0.41 0.42 0.43 0.44 0.44 0.45 0.46 0.47 0.47
[31] 0.47 0.48 0.48 0.50 0.50 0.50 0.51 0.52 0.52 0.52 0.53 0.53 0.53 0.54 0.57
[46] 0.57 0.58 0.59 0.61 0.63
```

It looks like all of the values of `Approve21$Approve_prop` are indeed expressed as proportions. The bracketed numbers do not represent values of the object. Instead, they tell the order of the outcomes in the dataset. For instance, [1] is followed by .23, .26, and 29, indicating the value of the first observation is .23, the second is .26, and the third is .29. Likewise, [16] is followed by .37, .37, and .39, indicating the value for the sixteenth observation is .37, the seventeenth is .37, and the eighteenth is .39, and so on.

Packages and Libraries

Packages and libraries are very important objects in the R environment. **Packages** are collections of functions, objects (usually data sources), and code that enhance the base functions of R. For instance, the function `hist` (used to create a histogram) is part of a package called `graphics` that is automatically installed as part of the R base, and the function `read_xlsx` (used to import the Excel spreadsheet) is part of a package called `readxl`. Once you have installed a package, you should not have to reinstall it unless you start an R session on a different device.

If you need to use a function from a package that is not already installed, you use the following command.

```
install.packages("package_name")
```

You can also install packages using the Packages tab in the lower-right window in RStudio. Just go to the Install icon, search for the package name, and click on "Install." When you install a package, you will see a lot of stuff happening in the Console window. Once the > prompt returns without any error messages, the new package is installed. In addition to the core set of packages that most people use, we will rely on a number of other packages in this book.

Each chapter in this book begins with a list of the packages that will be used in the chapter, and you can install them as you work through the chapters, if you wish. Alternatively, you can copy, paste, and run the code that follows to install all of the packages used in this book in one fell swoop.

```
install.packages(c("dplyr", "descr", "DescTools", "effectsize",
                   "gplots", "Hmisc", "lmtest", "ppcor", "readxl",
                   "sandwich", "stargazer"))
```

Libraries are where the packages are stored for active use. Even if you have installed all of the required packages, you generally can't use them unless you have loaded the appropriate library.

```
#Note that you do not use quotation marks for package names here.
library(package_name)
```

You can also load a library by going to the Packages tab in the lower-right window and clicking on the box next to the already installed package. This command makes the package available for you to use. I used the command `library(readxl)` before using `read_xlsx()` to import the presidential approval data. This told R to make the functions in `readxl` available for me to use. If I had not done this, I would have gotten an error message something like "Error: could not find function `read_xlsx`." You will also get this error if you misspell the function name (happens a lot). Installing the packages and attaching the libraries allows us to access and use any functions or objects included in these packages.

MANAGING FILES AND OUTPUT

Working Directory

When you create files, ask R to use files, or import files to use in R, they are stored in your **working directory.** Organizationally, it is important to know where this working directory is or to change its location so you can find your work when you want to pick it up again. The best practice in R is to designate a specific folder on your computer as your working directory. If you opened a new folder and downloaded the book files there as I suggested earlier in this chapter, you should use that as your working directory. You can set this as your working directory, as follows.

```
# Here, the working directory is set to a folder on my desktop.
setwd("/Users/username/Desktop/PolSci390")
```

You can also check on the location of the working directory to make sure it is where you think it is with the `getwd()` command.

```
getwd()
[1] "/Users/username/Desktop/PolSci390"
```

The nice thing about keeping all of your work (data files, script files, etc.) in the working directory is that you do not have to include the sometimes cumbersome file path names when accessing the files So, instead of having to write something like `readxl("/Users/username/Desktop/PolSci390/Approve21.xlsx")`, you just have to write `readxl("Approve21.xlsx")`. If you need to use files that are not stored in your working directory and you don't remember their location, you can use the `file.choose()` command to locate them on your computer. The output from this command will be the correct path and file name, which you can then copy and paste into R.

If you are using **posit Cloud** and you have uploaded the course files, all of those files should be in the working directory in the cloud. You can check this by clicking on the Files tab in the lower-right window of RStudio. This should show you all of the uploaded files, and any work you save in the posit Cloud environment should be saved there.

Save Your Work

Before wrapping up the material in this chapter, you should save your work. This means not just saving the script file but also saving the `Approve21` dataset as an R data file. To save the script file, just go to the File icon at the top of the Source window, click the icon, and assign your file an intuitive name that you will be able to remember. I use "chpt2" as the name, which R saves as `chpt2.R`. This file will be saved to the working directory. Now, any time you need to use commands from this file, you can click on the folder icon at the top of the RStudio window, locate the file, and open it.

You should also save the `Approve21` dataset as an R data file so you won't need to import the Excel version of this dataset when you want to use it again. The command for saving an object as an R data file is `save(objectname, file = filename.rda)`. Here, the object is `Approve21`, and I want to keep that name as the file name, so I use the following command.

```
save(Approve21, file = "Approve21.rda")
```

This will save the file to your working directory. When you want to use this file in the future, you can use the `load()` command to access the dataset.

```
load("Approve21.rda")
```

Creating Documents

If you are working with R, odds are that you are working on something that you need to share with others. For most of you, this means turning in end-of-chapter assignments or perhaps a longer writing assignment. It's also possible that you need to submit a report of some sort for your job or you need to put together a presentation on some subject. Either way, you need to integrate the on-screen results you get using R into some sort of document. There are a couple of options for integrating R results into documents: using a standard word processing program or by creating a Quarto document.

Standard Word Processing Document

First, you can copy and paste the findings into another document, probably using Microsoft Word or some similar word-processing product. When you do this, you should make sure you format the pasted output using a *fixed-width font* ("Courier" and "Courier New" work well). Otherwise, the columns in the R output will wander all over the place, and it will be hard to evaluate your work. You should always check to make sure the columns are lining up on paper the same way they did on screen. You might also have to adjust the font size to accomplish this. If you save your document as a PDF, make sure to double-check the formatting in the final product.

To copy graphs, you can click "Export" in the graph window and save your graph as an image (make sure you keep track of the file location). Then, you can insert the graph into your document. Alternatively, you can also click "copy to clipboard" after clicking on "Export" and copy and paste the graph directly into your document. You might have to resize the graphs after importing, following the conventions used by your document program.

You may have to deal with other formatting issues, depending on what document software you are using. The main thing is to always take time to make sure your results are presented as neatly as possible. In most cases, it a good idea to include only your final results, unless instructed otherwise.

Quarto Document

Alternatively, you can create a Quarto document. Quarto is a posit Cloud product that allows you to integrate your code, the results of your analysis, and your discussion of the results into a single document, all while working in the RStudio environment. Generally, Quarto documents look much better than those created by copying and pasting results into a standard word-processing program. Another benefit to Quarto is that learning to use it gives you another important skill that you can add to your resume. Although your experience with Quarto may be a bit bumpy at the beginning, it is really a breeze once you've created a couple of documents. You can open a Quarto document by clicking on the New File tab, where you open your script file. If you decide to use Quarto or if your instructor requires it, you will find a brief tutorial in Appendix B.

NEXT STEPS

Everything in this chapter will help you get started on your journey to using R for political and social data analysis. That said, this information is really just the very small piece of the tip of a very large iceberg of information. There is a lot more to learn. I say this not to intimidate or cause concern but because I think this is a reason for excitement. For most students, doing data analysis and working with R are both new experiences, and that's exactly what makes this an exciting opportunity. This book is designed to make it possible for any student to learn and improve their skill level, as long as they are willing to commit to putting in the time and effort required.

You will learn more about data analysis and how to use additional R commands in the next several chapters, and you probably will suffer through some frustration as R gives you the occasional error message, but if you stay on top of things (on schedule or ahead of schedule) and ask for help before it's too late, you will be fine.

EXERCISES

Concepts and Calculations

1. The following lines of R code were used earlier in the chapter. For each one, identify the object(s) and function(s). Explain your answer.
   - `class(Approve21$Disapprove)`
   - `save(Approve21, file = "Approve21.rda")`
   - `Approve21$Approve_prop <- Approve21$Approve/100`
   - `Approve21 <- read_excel("Approve21.xlsx")`

2. In your own words, describe what a script file is and what its benefits are.

3. The following command saves a copy of the `states20` dataset: `save(states20, file = "states20.rda")`. Is this file being saved to the working directory or to another directory? How can you tell?

4. After using `head(Approve21)` to look at the first few rows of the Approve21 dataset, I pasted the screen output to my MS Word document and it looked like this:

   state stateab Approve Disapprove Neither

   Alabama AL 31 63 6
   Alaska AK 41 53 6
   Arizona AZ 44 51 5
   Arkansas AR 31 63 6
   California CA 57 35 8
   Colorado CO 50 44 7

Why did I end up with these crooked columns that don't align with the variable names? How can I fix it?

R Problems

1. I tried to load the `anes20.rda` data file using the following command: `load(Anes20.rda)`. It didn't work, and I got the following message. **Error in load(Anes20.rda): object 'Anes20.rda' not found**. What did I do wrong? (Hint: there are two errors.)

2. Load the `countries2` dataset and get the names of all of the variables included in it. Based just on what you can tell from the variable names, what sorts of variables are in this dataset? Identify one variable that looks like it might represent something interesting to study (a potential dependent variable), and then identify another variable that you think might be related to the first variable you chose.

3. Use the `dim` function to tell how many variables and how many countries are in the dataset.

4. Use the `class()` function to identify the level of measurement of `ccode` and `matmort` from the `countries2` dataset.
5. Use the `Approve21` dataset and create a new object, `Approve21$net_approve`, which is calculated as the percentage in the state who approve of the Biden's performance MINUS the percentage in the state who disapprove of Biden's performance. Sort the dataset by net approve using the following command: `Approve21 <- Approve21[order(Approve21$net_approve), ]`, and list the six highest and lowest states. Say a few words about the types of states in these two lists.
6. Produce a histogram of `Approve21$net_approve` and describe what you see. Be sure to provide substantively meaningful labels for the histogram.