An R Programming Crash Course

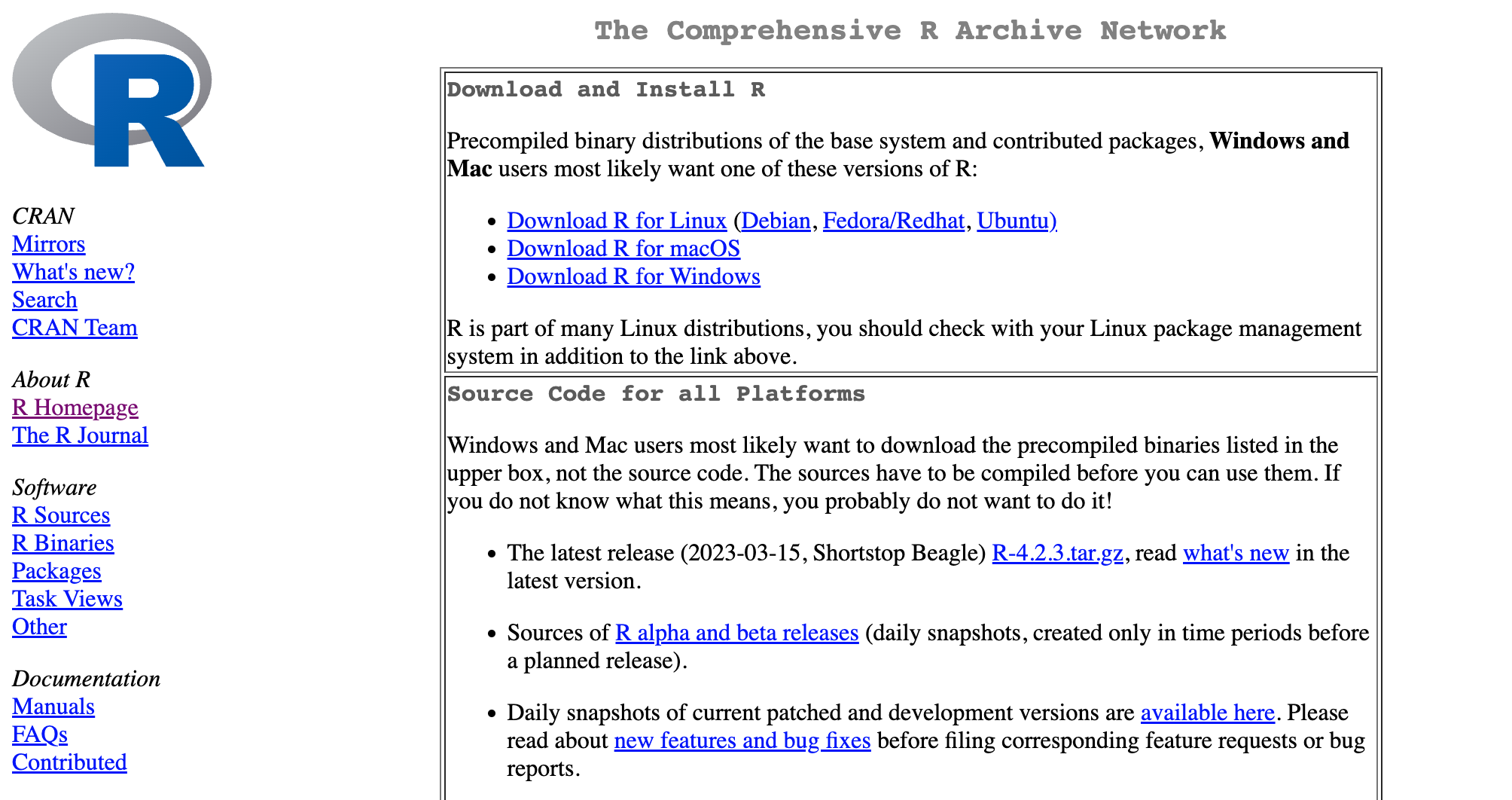
R has a well-earned reputation for being hard to learn. Especially for those who come to R without programming experience, it can be hard to figure out how things work. This chapter is designed to help those who have never used R before. I’ll start from scratch, showing you what you need to download in order to use R, and how to work with data using functions, objects, packages, and projects. If you have some experience with R, feel free to skip this chapter. But if you’re just starting out, this chapter will help you understand the basics, and help you make sense of the rest of the book.

Getting Set Up

One of the more confusing things for people just starting out is that you need two pieces of software in order to use R. The first is R itself, which provides the underlying computational tools that make R work. The second is RStudio, which makes working with R much easier. The best way to understand the relationship between R and RStudio is with this analogy from the book Modern Dive by Chester Ismay and Albert Kim. R is the engine that makes your work with data go. RStudio is like a dashboard that makes it easier to work with your data by providing a more user-friendly interface.

Let’s download each piece and get started. To download R, go to <https://cloud.r-project.org/> and choose your operating system, as seen in Figure 2-1

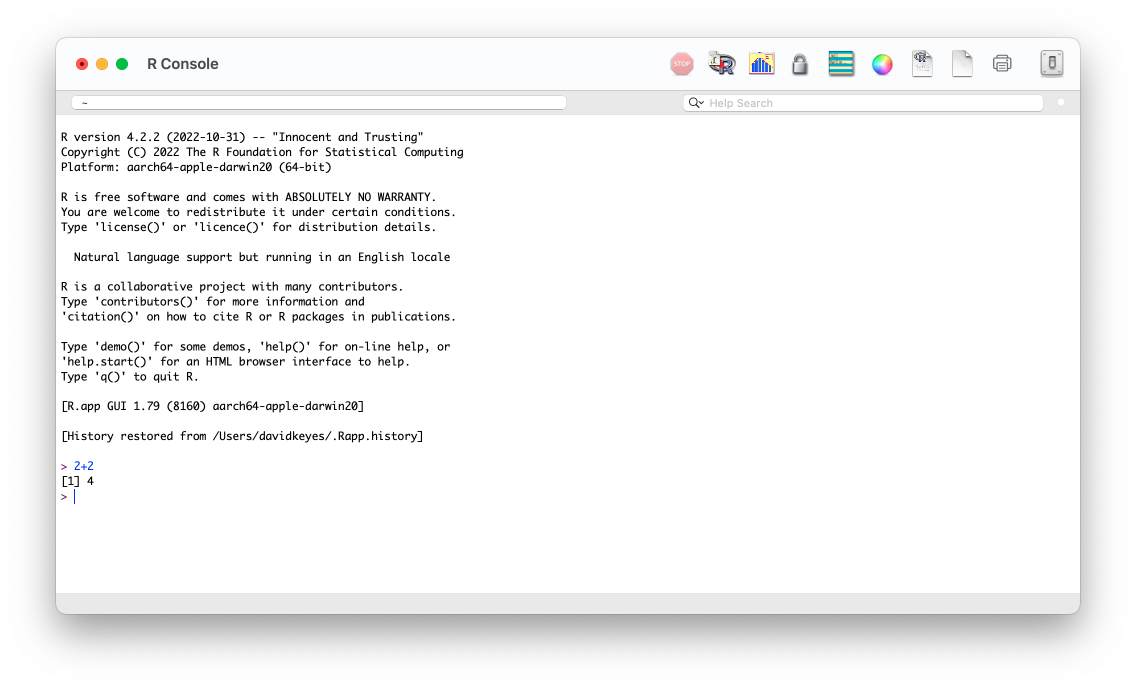
[F02001.png]



* + - * 1. The Comprehensive R Archive Network where you can download R

Once you download and install R, open it and you can work at the command line. For example, I can type 2 + 2, hit enter, and I will see 4, as seen in Figure 2-2.

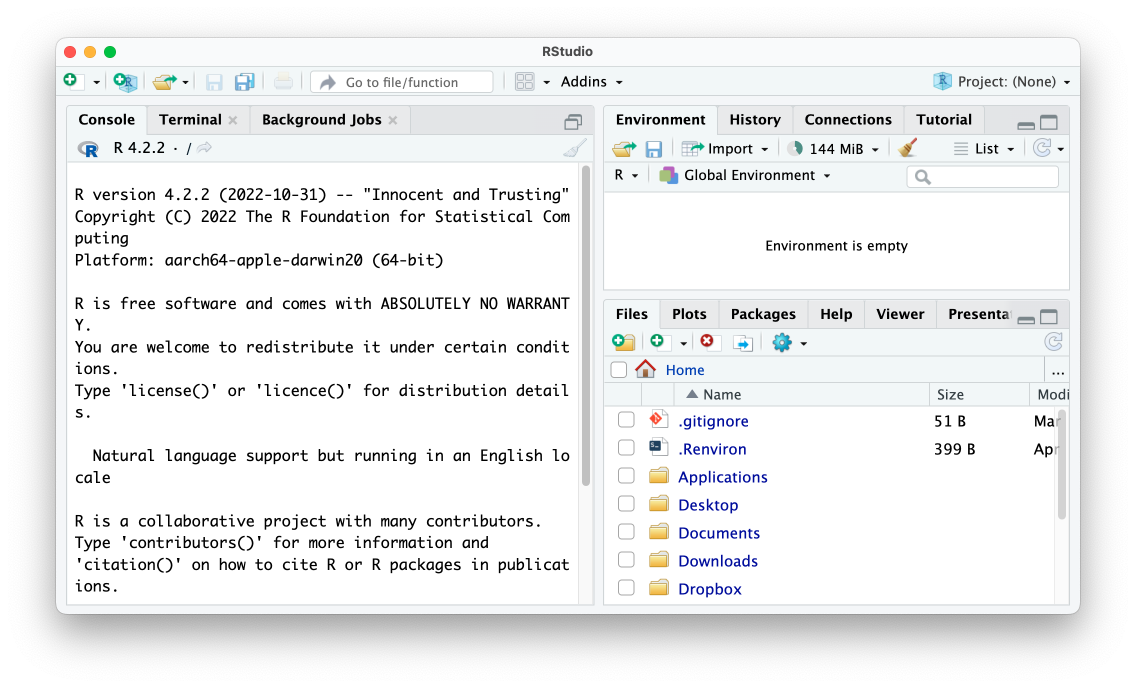
[F02002.png]



* + - * 1. The R console

Simple math problems are only the start; you can do pretty much anything in R. No matter what you’re planning to do, you’re probably not super impressed with the R interface. A few brave souls work only using the command line we’re looking at, but most do not. RStudio is where most R coders do their work. RStudio is like a skin that lives on top of R itself. It doesn’t provide new functionality to R, but it wraps R in a much more user-friendly interface, providing a way to see your files, outputs, and more. You can download RStudio at <https://posit.co/download/rstudio-desktop/>. Install RStudio as you would any other app and open it up. RStudio has several panels. The first time you open RStudio, you’ll see these the three panels shown in Figure 2-3 below.

[F02003.png]



* + - * 1. The RStudio editor

The left side panel should look familiar. It is what we saw when working in R. This is known as the console. You’ll use it to add code and see results. This panel, like the others we’ll discuss, has several tabs (terminal and background jobs) for more advanced usages. For now, we’ll stick to the default tab. Let’s look at the bottom right panel next. This files panel shows all of the files on my computer. Finally, the top right panel shows my environment. The environment shows the objects (discussed below) I have available to me when working in RStudio. There’s one more panel that you’ll typically have when working in RStudio. But to make it appear, we need to create an R script file.

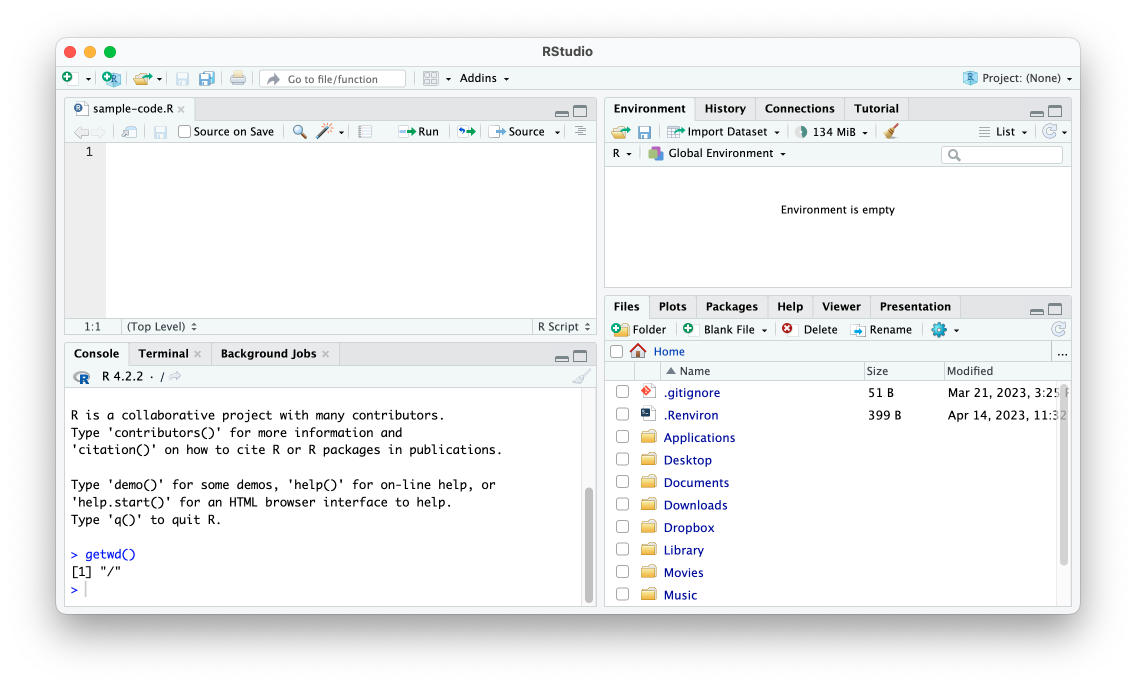
R Script Files

If you work in the console, either in RStudio or in R itself, you don’t have a record of your code. Say you sit down today and write code to import your data, analyze it, and make some graphs. You don’t want to have to recreate that code from scratch tomorrow. The way to save your code is by using files. Files allow you to save all of the code you have written. There are two types of files we’ll discuss in this book:

1. R script files, which only contain code.
2. R Markdown files, which contain code combined with text.

We’ll talk about R Markdown files starting in Chapter 7. Let’s start with R script files, which use the .R extension. To create an R script file, go to File > New File > R Script. When you create a new R script file, you’ll now have a fourth panel in the top left, which you can see in Figure 2-4. I’ll save this file in my Documents folder as sample-code.R.

[F02004.png]



* + - * 1. RStudio with four panels

I can now use the same syntax in my R script file that I did when working in just R. If I type 2 + 2 in the R script file and hit the Run button, 4 will show up in the console pane. If you’re looking to learn R, it’s probably not to help you figure out the answer to 2 + 2. Instead, you probably want to read in your own data and do analysis on it. Let’s work with some real data.

Working with Data

To explain how you work with data in R, we need go on a bit of a detour. We’ll make stops to discuss RStudio functions, objects, packages, and projects before we import and take a look at our data.

Conceptually, working with data in R is very different than working with data in a tool like Excel. In Excel, your data and any analysis you do on it all live in the same place: a spreadsheet. With R, you typically have data that lives in some external source (for example, an Excel spreadsheet or a CSV file). In order to work with this data in R, you have to run code to import it. It’s only once you’ve run this code, which is made up of functions, that you have the data available in R.

Functions

Let’s say I have a CSV file called population-by-state.csv in my Documents folder that I want to import to R. To import it into R, you might think to add a line like this in the sample-code.R file:

read.csv(file = "/Users/davidkeyes/Documents/population-by-state.csv")

This line shows the read.csv() function. Functions in R are pieces of code that you can run to do specific things. Functions have a name and arguments, which are surrounded by parentheses. Looking at the read.csv() function, the name, which appears before the open parentheses, is read.csv. Within the parentheses, we have the text file = "Documents/population-by-state.csv". Here we can see the argument file. The text after the equals sign gives the location of the file we want to read in. Arguments work in this way: the argument name, followed by the equals sign, followed by some value. This allows us to do something general (like importing a CSV) while allowing us to choose the specific file to run the function on. Functions can have multiple arguments as well, each of which is separated by a comma. For example, this would read in the same file, but skip the first row.

read.csv(file = "/Users/davidkeyes/Documents/population-by-state.csv",

skip = 1)

At this point, you might think to run the code in order to import your data. You can do so by selecting the line of code and hitting the Run button (or using the keyboard shortcut Command/Control + Enter on Mac/Windows). Running this code causes this text show up in the console pane.

#> rank State Pop Growth Pop2018

#> 1 1 California 39613493 0.0038 39461588

#> 2 2 Texas 29730311 0.0385 28628666

#> 3 3 Florida 21944577 0.0330 21244317

#> 4 4 New York 19299981 -0.0118 19530351

#> 5 5 Pennsylvania 12804123 0.0003 12800922

#> 6 6 Illinois 12569321 -0.0121 12723071

#> 7 7 Ohio 11714618 0.0033 11676341

#> 8 8 Georgia 10830007 0.0303 10511131

#> 9 9 North Carolina 10701022 0.0308 10381615

#> 10 10 Michigan 9992427 0.0008 9984072

--snip--

This is R confirming that it read in the CSV file and showing us the data within it. You might think you are ready to work with your data in R. But in fact all you’ve done at this point is display the result of running the code that imports your data. To use the data again, you need to save the result of running the code to an object.

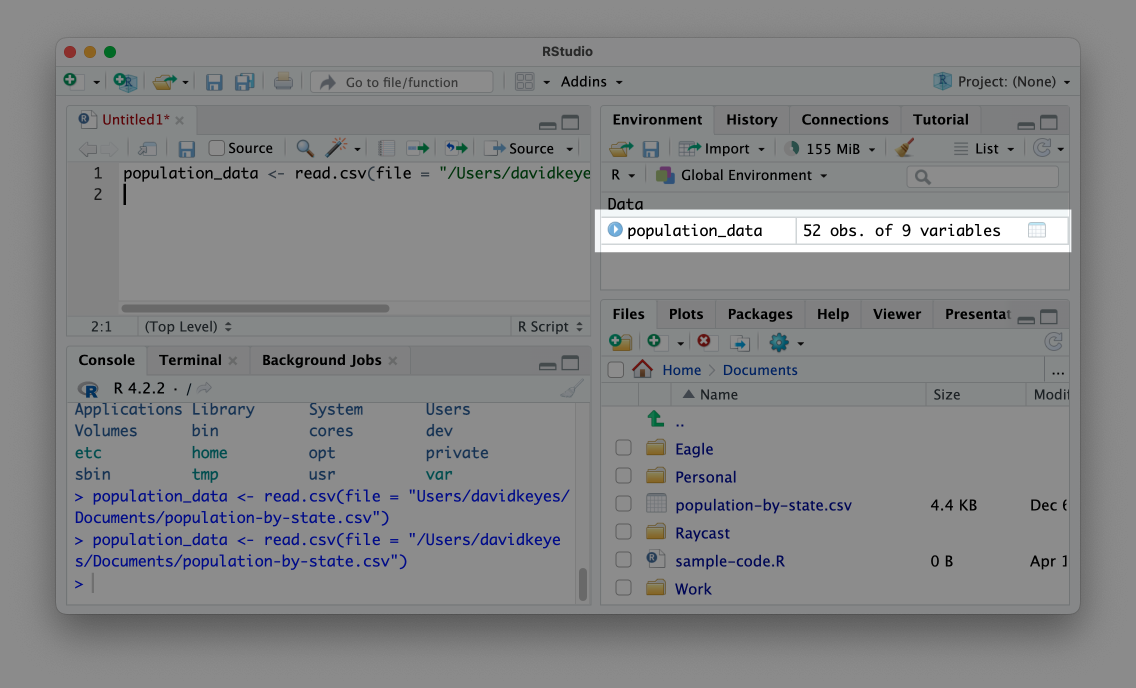
Objects

To save your data for reuse, you need to create an object. To do so, you would add to your data importing syntax from above.

population\_data <- read.csv(file = "/Users/davidkeyes/Documents/population-by-state.csv")

The second half of this code is what we used above, but we’ve added to it. In the middle you will see this: <-. Known as the assignment operator, it takes what follows it and assigns it to the item on the left. To the left of the assignment operator is population\_data. This is an object. Put together, the whole line reads in the CSV and assigns it to an object called population\_data. If you run this line of code, you will now see population\_data in your environment pane, as in Figure 2-5.

[F02005.png]



* + - * 1. An object in our environment pane

This is confirmation that your data import worked and you have the population\_data object ready for future use. Now, instead of having to rerun the code to import the data, I can simply type population\_data, run that line, and I’ll see the same output as above. Data imported to an object is known as a data frame.

Packages

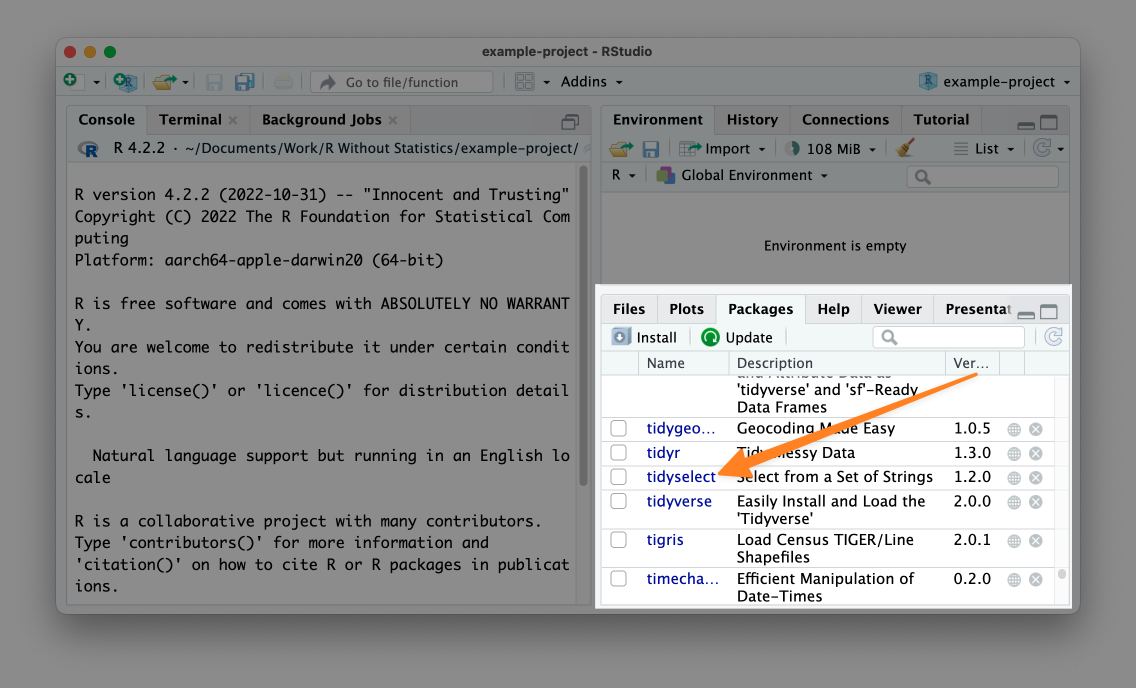
The read.csv() function that we’ve used up to this point is one of a set of functions that come from what is known as base R. They are built into R and you simply have to type the name of the function to use it. However, one of the benefits of R being an open source language is that anyone create their own code and share it with others. R users around the world make what are called packages, which provide code to do specific things.

The best analogy for understanding packages also comes from the Modern Dive book. The functionality in base R is like that built into a phone. A phone can do a lot on its own. But you usually want to install apps on your phone to do specific things. Packages are like apps, giving you specific functionality that doesn’t come built into base R.

You can install packages using the install.packages() function. For example, to install the tidyverse package, which provides a range of functions for data import, cleaning, analysis, visualization, and more, you would type install.packages("tidyverse"). I typically enter this code in the console because you only need to install a package once on your computer and so I know I won’t need to rerun this code.

To confirm that the tidyverse package has been installed correctly, click on the packages tab on the bottom right panel. Search for tidyverse and you should see it pop up, as in Figure 2-6.

[F02006.png]



* + - * 1. Confirmation that the tidyverse package is installed on my computer

Now that we’ve installed the tidyverse package, let’s use it. While you only need to install packages once per computer, you need to load packages each time you restart RStudio. You can only use functions from the tidyverse package if you first run the line library(tidyverse). I’ll go back to my sample-code.R file and re-import my data using a function from the tidyverse package.

At the top of my script I load the tidyverse. Then, I use the read\_csv() function (note the \_ in place of the .) to import my data. This alternate function to import CSV files achieves the same goal of creating an object called population\_data\_2. If we type population\_data\_2 and run the code (either by using the run button or the keyboard shortcut) you will see the output in the console.

#> # A tibble: 52 × 9

#> rank State Pop Growth Pop2018 Pop2010

#> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>

#> 1 1 California 39613493 0.0038 39461588 37319502

#> 2 2 Texas 29730311 0.0385 28628666 25241971

#> 3 3 Florida 21944577 0.033 21244317 18845537

#> 4 4 New York 19299981 -0.0118 19530351 19399878

#> 5 5 Pennsylvania 12804123 0.0003 12800922 12711160

#> 6 6 Illinois 12569321 -0.0121 12723071 12840503

#> 7 7 Ohio 11714618 0.0033 11676341 11539336

#> 8 8 Georgia 10830007 0.0303 10511131 9711881

#> 9 9 North Carolina 10701022 0.0308 10381615 9574323

#> 10 10 Michigan 9992427 0.0008 9984072 9877510

#> # ℹ 42 more rows

#> # ℹ 3 more variables: growthSince2010 <dbl>, Percent <dbl>,

#> # density <dbl>

What we see is slightly different from what we saw above using the read.csv() function. R describes the output as a tibble and only shows us the first 10 rows. This slightly different output occurs because read\_csv() imports the data not as a data frame, but as a tibble. Both are used to describe rectangular data like what you would see in a spreadsheet. While there are some small differences between data frames and tibbles, I’ll use the terms interchangeably in this book.

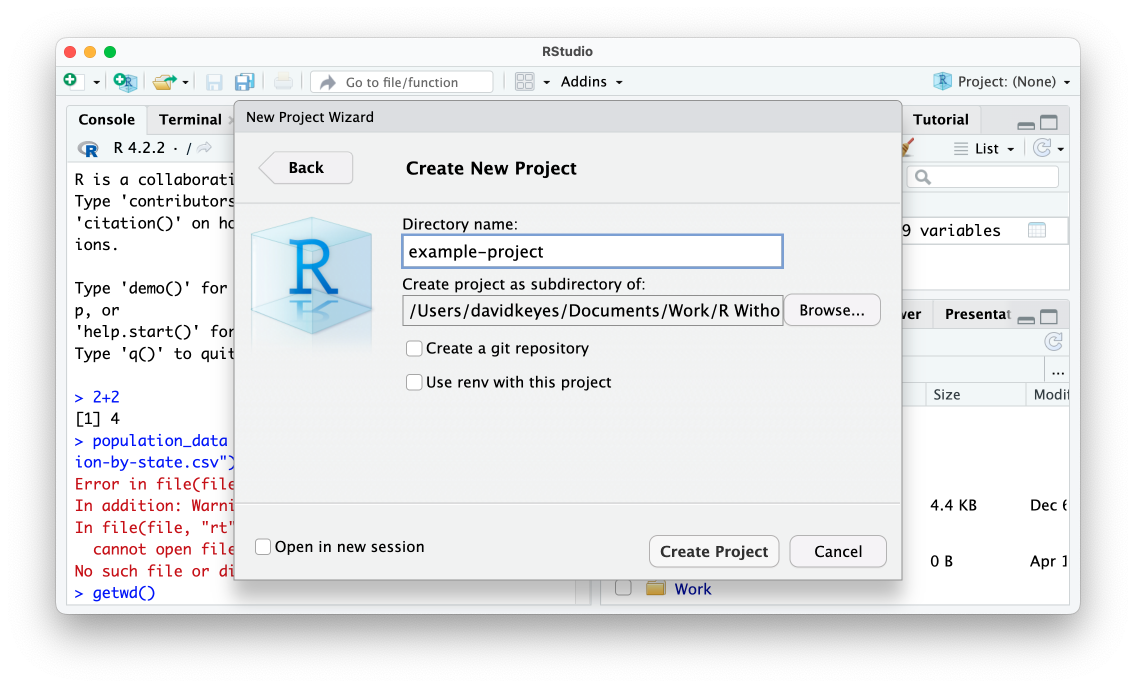
RStudio Projects

So far, we’ve imported a CSV file from the Documents folder. But the path to the file on my computer was /Users/davidkeyes/Documents/population-by-state.csv. Since others will not have this exact location on their computer, if they try to run my code, it won’t work. There’s a solution to this problem, and it’s called RStudio projects.

By working in a project, you can use what are known as relative paths to your files. Instead of having to write out read\_csv(file = "/Users/davidkeyes/Documents/population-by-state.csv"), you can put the CSV file in your project and then call it using read\_csv(file = "population-by-state.csv"). This makes it easier for you, and enables others to use your code.

To create a new RStudio project, go to File > New Project. Select either New Directory or Existing Directory and choose where to put your project. If you choose New Directory, you’ll need to specify that you want to create a new project. I’ll do this and then choose a name for the new directory and where it should live. As seen in Figure 2-7, you can leave the two checkboxes that ask about creating a git repository and using renv unchecked (these are for more advanced purposes).

[F02007.png]

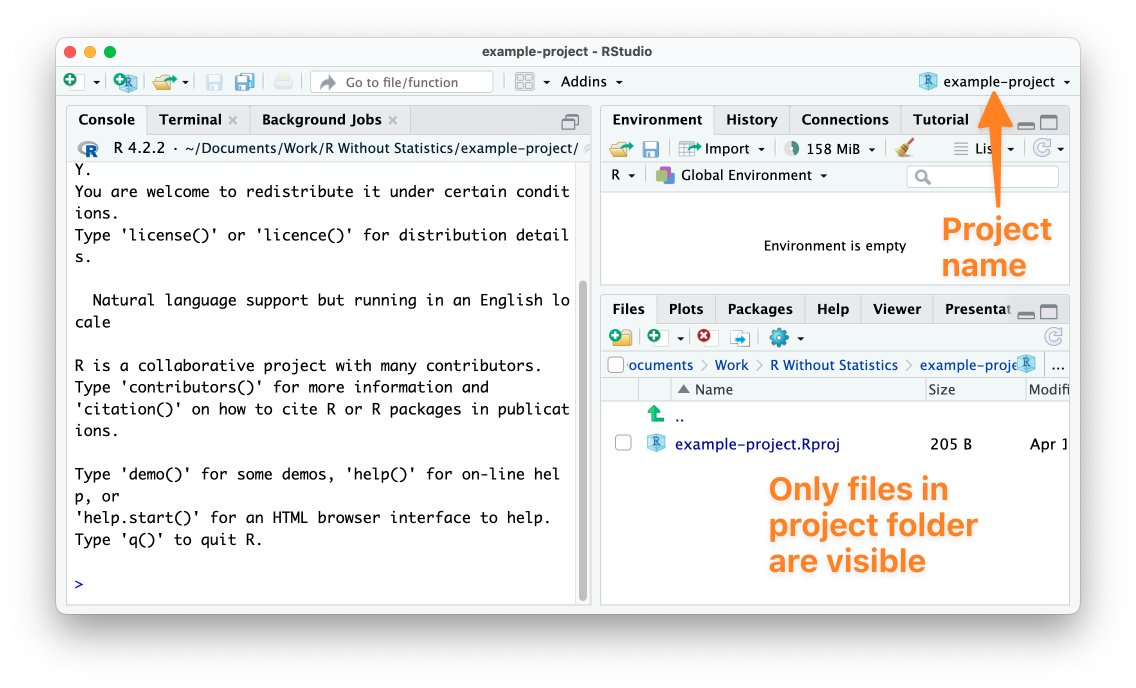


* + - * 1. The RStudio prompt to create a new project

Having now created this project, there are two major differences in RStudio’s appearance:

First, the files pane no longer shows every file on my computer, but instead only shows files in the example-project directory. Right now that’s just the example-project.Rproj file that indicates the folder contains a project. Second, at the very top right of RStudio, you can see the name of the example-project project (it had previously said Project: (None)). If you want to make sure you’re working in a project, make sure you see its name here. Both of these changes can be seen in Figure 2-8 below.

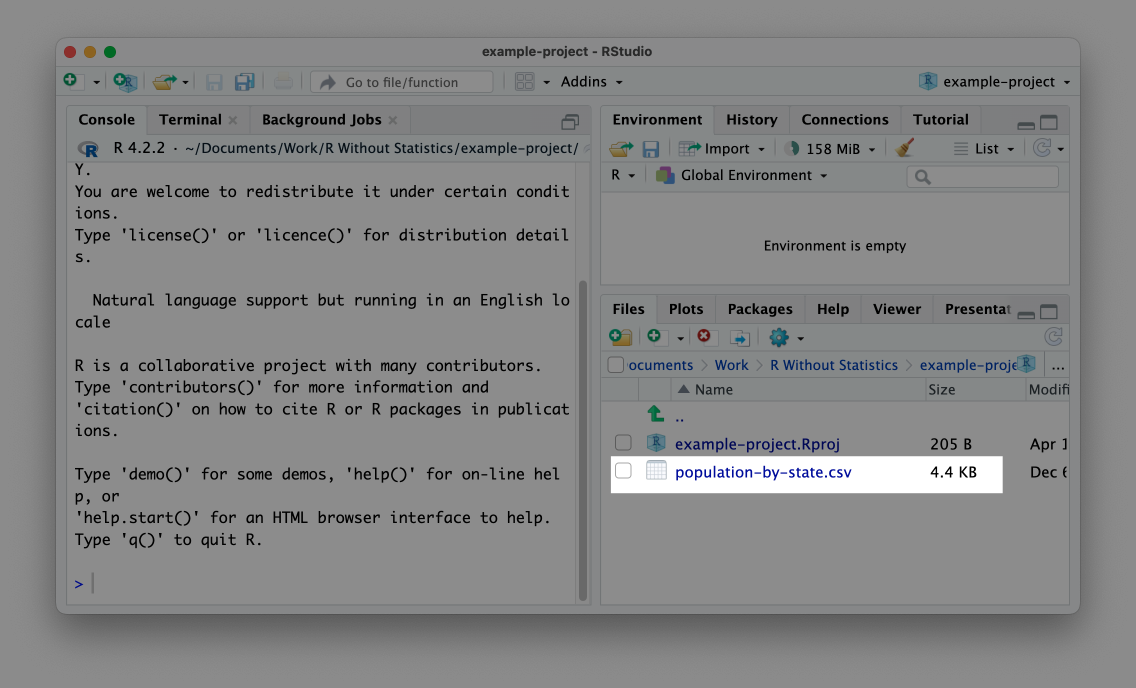
[F02008.png]



* + - * 1. RStudio with an active project

Now that I’ve created a project, I’ll use the Finder on my Mac computer to copy the population-by-state.csv file into the example-project directory. Once I do this, I can see it in the RStudio files pane, as in Figure 2-9.

[F02009.png]



* + - * 1. A CSV file visible in the files pane in RStudio

With this CSV file in my project, I can now import it more easily. As before, I’ll start by loading the tidyverse package. After that, I can remove the reference to the Documents folder and import my data by simply using the name of the file:

library(tidyverse)

population\_data\_2 <- read\_csv(file = "population-by-state.csv")

I’m able to import the population-by-state.csv file in this way because the RStudio project sets the working directory to be the root of my project. With the working directory set in this way, all references to files are relative to the .Rproj file at the root of the project (this is where the name relative paths comes from). Now that we’re working in a project, anyone can run this code because it imports the data from a location that they are guaranteed to have on their computer.

Data Analysis with R

Now that we’ve imported data, let’s do a bit of analysis on it. Below is a code snippet that calculates the mean population of all states using the summarize() function.

summarize(.data = population\_data\_2,

mean\_population = mean(Pop))

You can see that I’m using population\_data\_2 with the .data argument, telling the summarize() function (which comes from the tidyverse) to use that data frame. The second half of the code creates a new variable called mean\_population, which is calculated by using the mean() function on the Pop variable. Running this code will return a tibble with a single variable (mean\_population) that is of type double (meaning numeric) and has a value of 6433422, the mean population of all states.

#> # A tibble: 1 × 1

#> mean\_population

#> <dbl>

#> 1 6433422.

This is a basic example of data analysis, but you can do a lot more with the tidyverse. One advantage of working with the tidyverse is that it uses what’s known as the pipe for multi-step operations. The tidyverse pipe, which is written with the text %>%, allows us to break steps into multiple lines (the functionally equivalent so-called native pipe uses the text |>). For example, I can rewrite the code above to do the same thing using the pipe.

population\_data\_2 %>%

summarize(mean\_population = mean(Pop))

This code says: start with the population\_data\_2 data frame, then run the summarize() function on it, creating a variable called mean\_population by calculating the mean of the Pop variable.

The pipe becomes even more useful when we use multiple steps. Let’s say, for example, we want to calculate the mean population of the five largest states. The code below adds a line that uses the filter() function (also from the tidyverse) to only include states where the rank variable (which is the rank of the total population size of all states) is less than or equal to five (in other words, rank one through five). Then, it uses summarize() function as we did before.

population\_data\_2 %>%

filter(rank <= 5) %>%

summarize(mean\_population = mean(Pop))

Running this code shows us the mean population of the five largest states.

#> # A tibble: 1 × 1

#> mean\_population

#> <dbl>

#> 1 24678497

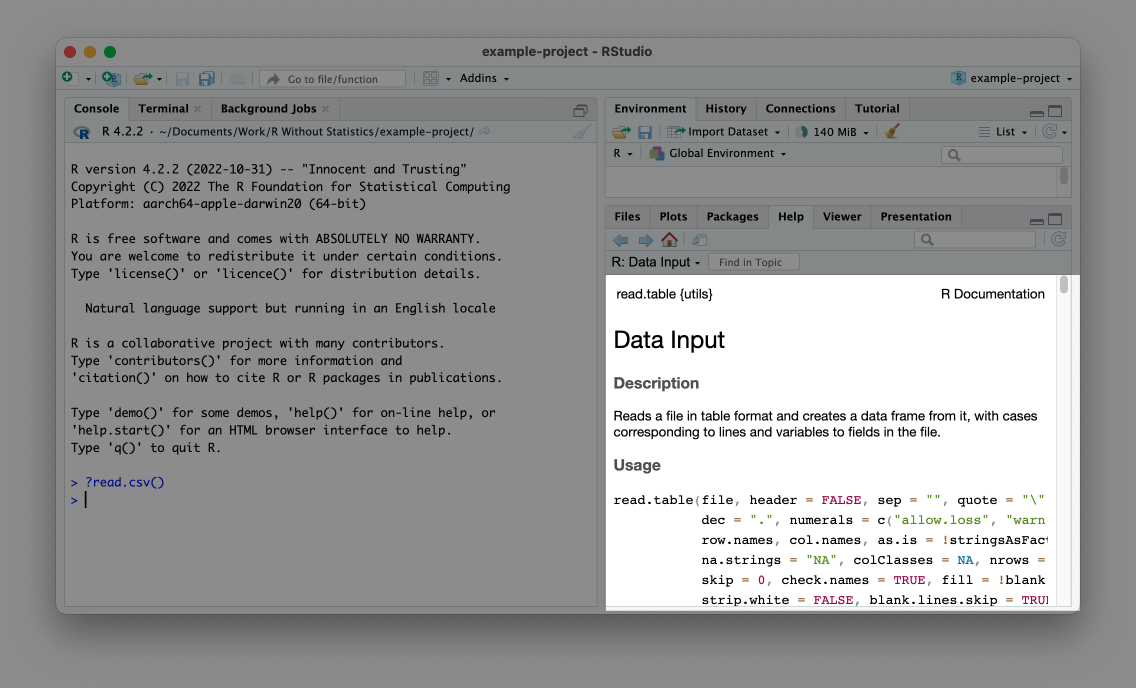
Combining functions using the pipe lets us do multiple things on our data in a way that keeps our code readable and easy to understand. We’ve introduced only two functions for analysis at this point, but the tidyverse has many functions that enable you to do nearly anything you could hope to do with your data. In fact, while I’ve been referring to the tidyverse as a single package, it is actually a collection of packages that do data importing, analysis, visualization, and more. The book R for Data Science by Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund is the bible of tidyverse programming and worth reading for more details on how its many packages work. In this book, I’ll introduce you to a number of packages, but because of how useful it is, the tidyverse will appear in every single piece of R code I write.

How to Get Help

Now that you’ve learned about the basics of how R works, you’re probably ready to dive in. When you do, you’re going to encounter errors. Everyone does, and it’s just part of working in R. Learning how to get help when you do run into issues is a key part of learning to use R successfully. There are two main strategies you can use to get unstuck.

The first is to read the documentation for functions. To access the documentation for any function, simply type the ? and then the name of the function. For example, if I run the line ?read.csv, I will see the documentation pop up in the bottom right panel, as in Figure 2-10.

[F02010.png]



* + - * 1. The documentation for the read.csv() function

Help files can be a bit hard to decipher but at their core, they tell you what package the function comes from, what it does, its arguments, and some examples of how to use it. For additional guidance on reading documentation, I recommend the appendix of Kieran Healy’s book Data Visualization: A practical introduction (a free online version is available at <https://socviz.co/appendix.html>).

In addition to providing help files in RStudio, many R packages have documentation websites. I find these easier to read and tend to use them when I am confused about how to use a function. In addition, many packages have longer articles known as vignettes that provide an overview of how the package works. Reading these can help you see how individual functions can be used in the context of a larger project. Every package I discuss in this book has a good documentation website.

In Conclusion: Invest Time Now to Learn R and Save Time Later

Getting started with R can be challenging. I myself experienced many frustrations early on that I’ve since realized are quite common. This chapter has, hopefully, helped you to see how you can get started with R. Understanding how functions, objects, packages, and projects work is key to ensuring that you can successfully use R to work with your data.

If R feels challenging, just know that it will get better with time. Best of all, the time you invest in learning to use R will repay itself many times over. My favorite example to show this is one I discuss in Chapter 8, which discusses a technique called parameterized reporting to automatically produce dozens, hundreds, or even thousands of reports at once. At my company, R for the Rest of Us, we worked with a client to produce reports on demographics and housing data for each of the 170 plus towns and counties in the state of Connecticut. Doing this by hand would have taken the client hundreds of hours. Using R, we were able to automate the process so they can generate the reports simply by running code. If you make multiple reports by hand, think of the hours you’re spending. Reframe the time it takes to learn R as an investment in never having to do this manual labor again. When you’re struggling to make sense of an inscrutable error message, you will get frustrated. But, with all of the time you’ll save, I promise that it is worth the effort to learn R.