Tutorial Notes
PROF 740 Foundations of Data Analytics

Getting Started with Data in R

Before we can start exploring data in R, there are some key concepts to understand first:

1. What is R and RStudio?
2. How do I code in R?
3. What are R packages?

We'll introduce these concepts in the upcoming Sections 1.1-1.3.

1.1 What are R and RStudio?

Throughout this tutorial, we will assume that you are using R via RStudio. First time users often confuse the two. At its simplest, R is like a car's engine while RStudio is like a car's dashboard as illustrated in Figure 1.1.

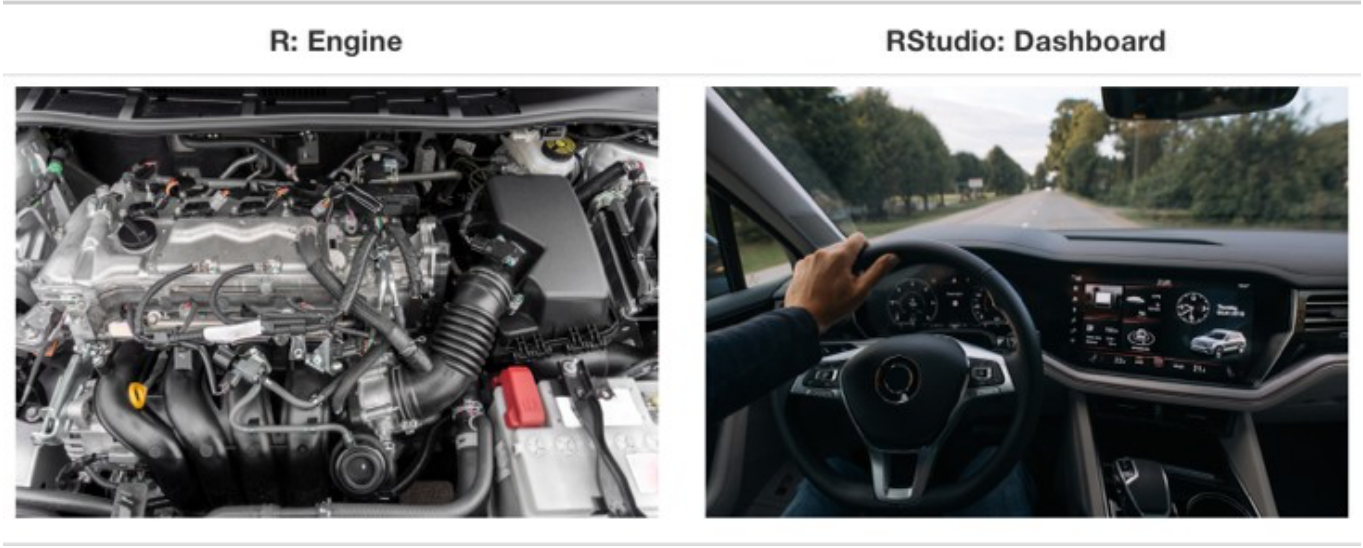


Figure 1.1: Analogy of difference between R and RStudio.

More precisely, R is a programming language that runs computations, while RStudio is an *integrated development environment (IDE)* that provides an interface by adding many convenient features and tools. So just as the way of having access to a speedometer, rearview mirrors, and a navigation system makes driving much easier, using RStudio's interface makes using R much easier as well.

1.1.1 Installing R and RStudio

You will first need to download and install both R and RStudio (Desktop version) on your computer. **It is important that you install R first and then install RStudio**.

1. **You must do this first:** Download and install R by going to https://cloud.r-project.org/.
    - If you are a Windows user: Click on "Download R for Windows", then click on "base", then click on the Download link.
    - If you are macOS user: Click on "Download R for (Mac) OS X", then under "Latest release:" click on R-X.X.X.pkg, where R-X.X.X is the version number.
    - If you are a Linux user: Click on "Download R for Linux" and choose your distribution for more information on installing R for your setup.

2. **You must do this second:** Download and install RStudio at https://www.rstudio.com/products/rstudio/download/.
    - Scroll down to “Installers for Supported Platforms” near the bottom of the page.
    - Click on the download link corresponding to your computer’s operating system.

1.1.2 Using R via RStudio

Recall our car analogy from earlier. Much as we don’t drive a car by interacting directly with the engine but rather by interacting with elements on the car’s dashboard, we won’t be using R directly but rather we will use RStudio’s interface. After you install R and RStudio on your computer, you’ll have two new *programs* (also called *applications*) you can open. **We’ll always work in RStudio and not in the R application**. Figure 1.2 shows what icon you should be clicking on your computer.

R: Do not open this	RStudio: Open this
	

Figure 1.2: Icons of R versus RStudio on your computer.

After you open RStudio, you should see something similar to Figure 1.3. (Note that slight differences might exist if the RStudio interface is updated after 2019 to not be this by default.)

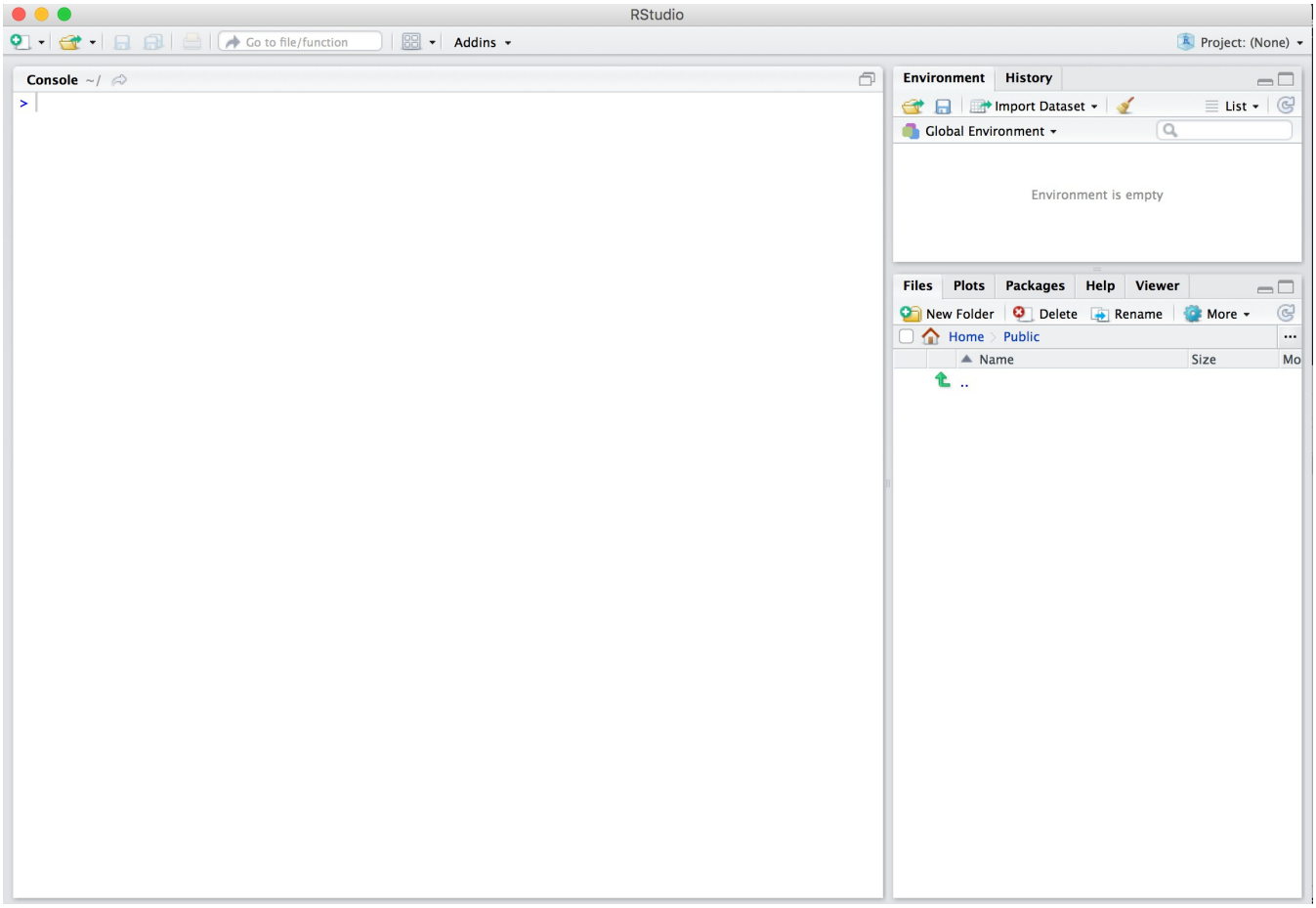


Figure 1.3: RStudio interface to R.

Note the three *panes* which are three panels dividing the screen: the *console pane*, the *files pane*, and the *environment pane*. Over the course of this tutorial, you'll come to learn what purpose each of these panes serves.

1.2 How do I code in R?

Now that you're set up with R and RStudio, you are probably asking yourself, "OK. Now how do I use R?". The first thing to note is that unlike other statistical software programs like Excel, SPSS, or Minitab that provide point-and-click interfaces, R is an interpreted language. This means you have to type in commands written in *R code*. In other words, you have to code/program in R. Note that we'll use the terms "coding" and "programming" interchangeably in this tutorial.

While it is not required to be a seasoned coder/computer programmer to use R, there is still a set of basic programming concepts that new R users need to understand. Consequently, while this book is not a book on programming, you will still learn just enough of these basic programming concepts needed to explore and analyze data effectively.

1.2.1 Basic programming concepts and terminology

We now introduce some basic programming concepts and terminology. Instead of asking you to memorize all these concepts and terminology right now, we'll guide you so that you'll "learn by doing." To help you learn, we will always use a different font to distinguish regular text from `computer_code`. The best way to master these topics is, in our opinions, through deliberate practice with R and lots of repetition.

- Basics:
  - *Console pane*: where you enter in commands.
  - *Running code*: the act of telling R to perform an act by giving it commands in the console.
  - *Objects*: where values are saved in R. We'll show you how to *assign* values to objects and how to display the contents of objects.
  - *Data types*: integers, doubles/numerics, logicals, and characters. Integers are values like -1, 0, 2, 4092. Doubles or numerics are a larger set of values containing both the integers but also fractions and decimal values like -24.932 and 0.8. Logicals are either `TRUE` or `FALSE` while characters are text such as "cabbage", "Hamilton", "The Wire is the greatest TV show ever", and "This ramen is delicious." Note that characters are often denoted with the quotation marks around them.
- *Vectors*: a series of values. These are created using the `c()` function, where `c()` stands for "combine" or "concatenate." For example, `c(6, 11, 13, 31, 90, 92)` creates a six element series of positive integer values .
- *Factors*: *categorical data* are commonly represented in R as factors. Categorical data can also be represented as *strings*. We'll study this difference as we progress through the book.
- *Data frames*: rectangular spreadsheets. They are representations of datasets in R where the rows correspond to *observations* and the columns correspond to *variables* that describe the observations. We'll cover data frames later in Section 1.4.
- *Conditionals*:
  - Testing for equality in R using `==` (and not `=`, which is typically used for assignment). For example, `2 + 1 == 3` compares `2 + 1` to `3` and is correct R code, while `2 + 1 = 3` will return an error.
  - Boolean algebra: `TRUE/FALSE` statements and mathematical operators such as `<` (less than), `<=` (less than or equal), and `!=` (not equal to). For example, `4`

`+ 2 >= 3` will return `TRUE`, but `3 + 5 <= 1` will return `FALSE`.
    - Logical operators: `&` representing "and" as well as `|` representing "or." For example, `(2 + 1 == 3) & (2 + 1 == 4)` returns `FALSE` since both clauses are not `TRUE` (only the first clause is `TRUE`). On the other hand, `(2 + 1 == 3) | (2 + 1 == 4)` returns `TRUE` since at least one of the two clauses is `TRUE`.
- *Functions*, also called *commands*: Functions perform tasks in R. They take in inputs called *arguments* and return outputs. You can either manually specify a function's arguments or use the function's *default values*.
    - For example, the function `seq()` in R generates a sequence of numbers. If you just run `seq()` it will return the value 1. That doesn't seem very useful! This is because the default arguments are set as `seq(from = 1, to = 1)`. Thus, if you don't pass in different values for `from` and `to` to change this behavior, R just assumes all you want is the number 1. You can change the argument values by updating the values after the `=` sign. If we try out `seq(from = 2, to = 5)` we get the result `2 3 4 5` that we might expect.
    - We'll work with functions a lot throughout this book and you'll get lots of practice in understanding their behaviors. To further assist you in understanding when a function is mentioned in the book, we'll also include the `()` after them as we did with `seq()` above.

This list is by no means an exhaustive list of all the programming concepts and terminology needed to become a savvy R user; such a list would be so large it wouldn't be very useful, especially for novices. Rather, we feel this is a minimally viable list of programming concepts and terminology you need to know before getting started. We feel that you can learn the rest as you go. Remember that your mastery of all of these concepts and terminology will build as you practice more and more.

1.2.2 Errors, warnings, and messages

One thing that intimidates new R and RStudio users is how it reports *errors*, *warnings*, and *messages*. R reports errors, warnings, and messages in a glaring red font, which makes it seem like it is scolding you. However, seeing red text in the console is not always bad.

R will show red text in the console pane in three different situations:

- **Errors**: When the red text is a legitimate error, it will be prefaced with "Error in…" and will try to explain what went wrong. Generally, when there's an error, the code will not

run. For example, we'll see in Subsection 1.3.3 if you see `Error in ggplot(...) : could not find function "ggplot"`, it means that the `ggplot()` function is not accessible because the package that contains the function (`ggplot2`) was not loaded with `library(ggplot2)`. Thus you cannot use the `ggplot()` function without the `ggplot2` package being loaded first.

- **Warnings**: When the red text is a warning, it will be prefaced with "Warning:" and R will try to explain why there's a warning. Generally, your code will still work, but with some caveats. For example, if you create a scatterplot based on a dataset where two of the rows of data have missing entries that would be needed to create points in the scatterplot, you will see this warning: `Warning: Removed 2 rows containing missing values (geom_point)`. R will still produce the scatterplot with all the remaining non-missing values, but it is warning you that two of the points aren't there.
- **Messages**: When the red text doesn't start with either "Error" or "Warning", it's *just a friendly message*. You'll see these messages when you load *R packages* in the upcoming Subsection 1.3.2 or when you read data saved in spreadsheet files with the `read_csv()` function. These are helpful diagnostic messages and they don't stop your code from working.

  Additionally, you'll see these messages when you install packages using `install.packages()` as discussed in Subsection 1.3.1.

Remember, when you see red text in the console, *don't panic*. It doesn't necessarily mean anything is wrong. Rather:

- If the text starts with "Error", figure out what's causing it. Think of errors as a red traffic light: something is wrong!
- If the text starts with "Warning", figure out if it's something to worry about. For instance, if you get a warning about missing values in a scatterplot and you know there are missing values, you're fine. If that's surprising, look at your data and see what's missing. Think of warnings as a yellow traffic light: everything is working fine, but watch out/pay attention.
- Otherwise, the text is just a message. Read it, wave back at R, and thank it for talking to you. Think of messages as a green traffic light: everything is working fine and keep on going!

1.2.3 Tips on learning to code

Learning to code/program is quite similar to learning a foreign language. It can be daunting and frustrating at first. Such frustrations are common and it is normal to feel discouraged as you learn. However, just as with learning a foreign language, if you put in the effort and are not afraid to make mistakes, anybody can learn and improve.

Here are a few useful tips to keep in mind as you learn to program:

- **Remember that computers are not actually that smart**: You may think your computer or smartphone is "smart," but really people spent a lot of time and energy designing them to appear "smart." In reality, you have to tell a computer everything it needs to do. Furthermore, the instructions you give your computer can't have any mistakes in them, nor can they be ambiguous in any way.
- **Take the "copy, paste, and tweak" approach**: Especially when you learn your first programming language or you need to understand particularly complicated code, it is often much easier to take existing code that you know works and modify it to suit your ends. This is as opposed to trying to type out the code from scratch. We call this the *"copy, paste, and tweak"* approach. So early on, we suggest not trying to write code from memory, but rather take existing examples we have provided you, then copy, paste, and tweak them to suit your goals. After you start feeling more confident, you can slowly move away from this approach and write code from scratch. Think of the "copy, paste, and tweak" approach as training wheels for a child learning to ride a bike. After getting comfortable, they won't need them anymore.
- **The best way to learn to code is by doing**: Rather than learning to code for its own sake, we find that learning to code goes much smoother when you have a goal in mind or when you are working on a particular project, like analyzing data that you are interested in and that is important to you.
- **Practice is key**: Just as the only method to improve your foreign language skills is through lots of practice and speaking, the only method to improving your coding skills is through lots of practice. Don't worry, however, we'll give you plenty of opportunities to do so!

1.3 What are R packages?

Another point of confusion with many new R users is the idea of an R package. R packages extend the functionality of R by providing additional functions, data, and documentation.

They are written by a worldwide community of R users and can be downloaded for free from the internet.

A good analogy for R packages is they are like apps you can download onto a mobile phone:

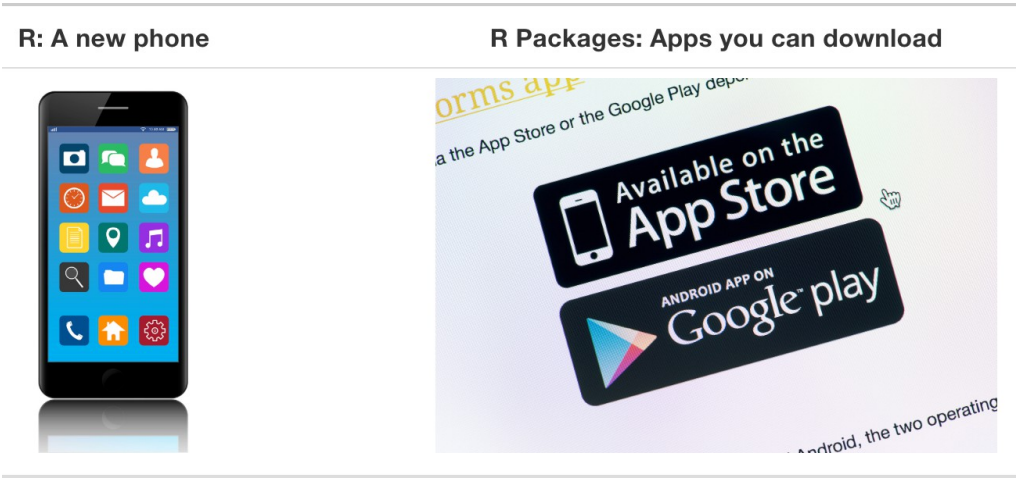


FIGURE 1.4: Analogy of R versus R packages.

So, R is like a new mobile phone: while it has a certain number of features when you use it for the first time, it doesn't have everything. R packages are like the apps you can download onto your phone from Apple's App Store or Android's Google Play.

Let's continue this analogy by considering the Instagram app for editing and sharing pictures. Say you have purchased a new phone and you would like to share a photo you have just taken with friends on Instagram. You need to:

1. *Install the app*: Since your phone is new and does not include the Instagram app, you need to download the app from either the App Store or Google Play. You do this once and you're set for the time being. You might need to do this again in the future when there is an update to the app.
2. *Open the app*: After you've installed Instagram, you need to open it.

Once Instagram is open on your phone, you can then proceed to share your photo with your friends and family. The process is very similar for using an R package. You need to:

1. ***Install the package***: This is like installing an app on your phone. Most packages are not installed by default when you install R and RStudio. Thus, if you want to use a package for the first time, you need to install it first. Once you've installed a package, you likely won't install it again unless you want to update it to a newer version.
2. ***"Load" the package***: "Loading" a package is like opening an app on your phone. Packages are not "loaded" by default when you start RStudio on your computer; you need to "load" each package you want to use every time you start RStudio.

Let's perform these two steps for the `ggplot2` package for data visualization.

1.3.1 Package installation

There are two ways to install an R package: an easy way and a more advanced way. Let's install the `ggplot2` package the easy way first as shown in Figure 1.5. In the Files pane of RStudio:

a. Click on the "Packages" tab.
b. Click on "Install" next to Update.
c. Type the name of the package under "Packages (separate multiple with space or comma):" In this case, type `ggplot2` .
d. Click "Install."

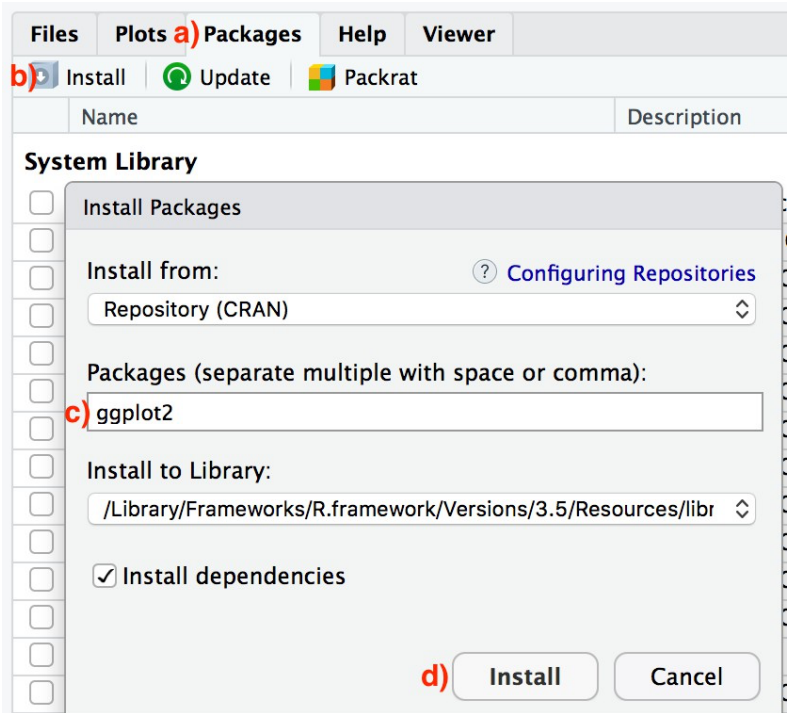


Figure 1.5: Installing packages in R the easy way.

An alternative but slightly less convenient way to install a package is by typing `install.packages("ggplot2")` in the console pane of RStudio and pressing Return/Enter on your keyboard. Note you must include the quotation marks around the name of the package.

Much like an app on your phone, you only have to install a package once. However, if you want to update a previously installed package to a newer version, you need to reinstall it by repeating the earlier steps.

1.3.2 Package loading

Recall that after you've installed a package, you need to "load it." In other words, you need to "open it." We do this by using the `library()` command.

For example, to load the `ggplot2` package, run the following code in the console pane. What do we mean by "run the following code"? Either type or copy-and-paste the following code into the console pane and then hit the Enter key.

```
library(ggplot2)
```

If after running the earlier code, a blinking cursor returns next to the `>` "prompt" sign, it means you were successful and the `ggplot2` package is now loaded and ready to use. If, however, you get a red "error message" that reads `...`

```
Error in library(ggplot2) : there is no package called 'ggplot2'
```

`...` it means that you didn't successfully install it. This is an example of an "error message" we discussed in Subsection 1.2.2. If you get this error message, go back to Subsection 1.3.1 on R package installation and make sure to install the `ggplot2` package before proceeding.

1.3.3 Package use

One very common mistake new R users make when wanting to use particular packages is they forget to "load" them first by using the `library()` command we just saw. **Remember**: *you have to load each package you want to use every time you start RStudio.* If you don't first "load" a package, but attempt to use one of its features, you'll see an error message similar to:

```
Error: could not find function
```

This is a different error message than the one you just saw on a package not having been installed yet. R is telling you that you are trying to use a function in a package that has not yet been "loaded." R doesn't know where to find the function you are using. Almost all new users forget to do this when starting out, and it is a little annoying to get used to doing it.

However, you’ll remember with practice and after some time it will become second nature for you.