

Breaking the Ice with R and RStudio

Gaston Sanchez

Contents

About	5
1 Installing R	7
1.1 Interacting with R	7
1.2 Installing R	9
2 Installing RStudio	13
2.1 Download RStudio	13
3 First Contact with R	17
3.1 First Contact with R (via RStudio)	17
3.2 Getting Help	22
3.3 Installing Packages	23
3.4 Exercises	27
4 A Quick Tour Around RStudio	29
4.1 First Contact with RStudio	29
4.2 RStudio Panes in a Nutshell	32
4.3 Exercises	34
5 Session Management	37
5.1 Starting a Session	37
5.2 Working During a Session	39
5.3 Closing a Session	41
6 Intro to R Markdown Files	43
6.1 Get to know Rmd files	43
6.2 Code chunks	45

About



This manuscript provides a brief tutorial for getting started with R and RStudio.

Citation

You can cite this work as:

Sanchez, G. (2022) Breaking the Ice with R and RStudio. <https://www.gastonsanchez.com/R-ice-breaker>

My Series of R Tutorials

This document is part of series of texts that I've written about Programming and Data Analysis in R:

- **Tidy Hurricanes: Analyzing Tropical Storms with Tidyverse Tools**

<https://www.gastonsanchez.com/R-tidy-hurricanes>

- **R Coding Basics: An Introduction to Practical Programming in R**

<https://www.gastonsanchez.com/R-coding-basics>

- **Rolling Dice: Exploring Simulations in Games of Chance with R**

<https://www.gastonsanchez.com/R-rolling-dice>

- **Web Technologies in R: A Short Introduction to Web Data Technologies in R**

<https://www.gastonsanchez.com/R-web-tech>

Donation

As a Data Science and Statistics educator, I love to share the work I do. Each month I spend dozens of hours curating learning materials like this resource. If you find any value and usefulness in it, please consider making a one-time donation—via PayPal—in any amount (e.g. the amount you would spend inviting me a cup of coffee or any other drink). Your support really matters.

License

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Chapter 1

Installing R

To learn how to use R, you obviously need to have access to it. This chapter, and the next one, guide you through the installation process of R and RStudio so that you can have both programs up and running in your computer.

If you already have R and RStudio installed in your machine, you can safely go to chapter First Contact with R.

1.1 Interacting with R

Before I show you how to install R and the integrated development environment RStudio, let me tell you first about the different ways in which users can work with R.

Overall, you can work with R in two major ways:

- 1) Interactive Mode, and
- 2) Non-interactive Mode

Interactive Mode. Most R users work with R in an interactive way, and this is certainly the way I personally interact with R in my daily coding activities. Interactive means that you launch R (i.e. you open a session), having direct access to its console. This is where you type in commands, and then R does its magic reading the commands, parsing them, evaluating them, and—typically—printing an output back into the console, waiting for you to type in the next command(s).

Non-interactive Mode. In contrast to interactive mode, working with R in non-interactive way involves writing all the commands in a text file (for example in an R script file), and then asking your computer—via the command line interface—to pass this file to R so that it runs the commands without you launching R or having direct access to its console.

Think of interactive way as having a direct conversation with R, establishing a dialogue in which you type in a command, R interprets it, gives you an answer, and then you type more commands, continuing the dialogue. In contrast, non-interactive is like writing a letter (or an email) to R. Here you don't have that synchronous conversation, instead R will see the script file, try to execute all the commands “behind the scenes”, and it will disappear when it finishes the computations. You may get some output back, but R is gone in the sense that there is no open session waiting for you to execute the next instructions.

Most of what I discuss in this book is applicable to writing code in R regardless of how you decide to interact with R. However, to learn R and to follow the examples of the book it is definitely much better to do it in an interactive way using: a) R's built-in graphical user interface (R's GUI), b) an integrated development environment (IDE) such as RStudio, or c) R from a command line interface (CLI) commonly referred to as a *terminal*.

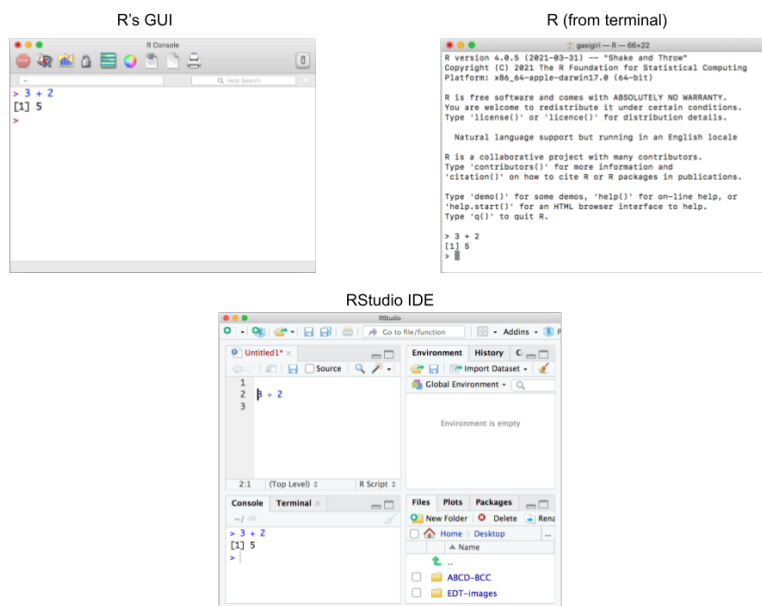


Figure 1.1: Interacting with R in various ways: R's GUI, RStudio, and R from a command-line terminal

While you can interact with R using its built-in graphical user interface (GUI) or launching R from the terminal (command line interface), nowadays I highly recommend that you interact with it using an *Integrated Development Environment* (IDE) such as **RStudio**. Simply put, programs like RStudio provide a nice working space that make your life easier while writing code, creating all sorts of reports, documents, and slides, running analysis, making graphs, generating outputs, creating web apps, etc.



Figure 1.2: Main computational tools: R and RStudio

Keep in mind that R and RStudio are not the same thing. R is like the main “engine” or computational core. RStudio is just a convenient layer that talks directly to R, and gives us a convenient working space to organize our files, to type in code, to run commands, visualize plots, interact with our filesystem, etc. Having said that, everything that happens in RStudio, can be done in R alone. Yes, you may need to write more code and work in a more rudimentary way, but nothing should stop your work in R if one day RStudio disappears from the face of the earth.

By the way, both R and RStudio are free, and available for Mac (OS X), Windows, and Linux (e.g. Ubuntu, Fedora, Debian). More about this in the following sections.

1.2 Installing R

To download and install R in your computer, follow the steps listed below.

Step 1) Go to the **R project** website: <https://r-project.org>

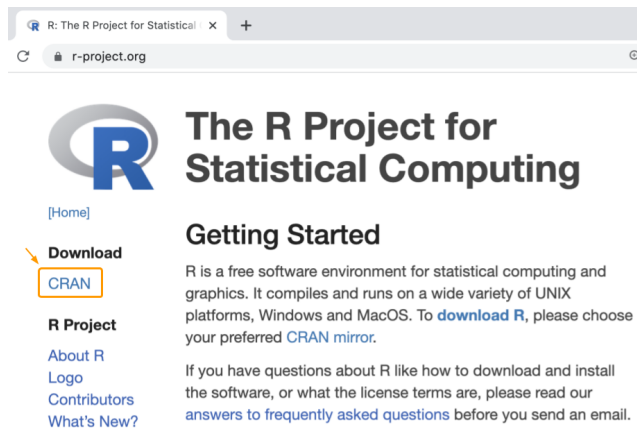


Figure 1.3: R project’s home webpage

Step 2) Click on the CRAN link, located in the navigation bar (on the left side). This will take you to the *Comprehensive R Archive Network* page (see screenshot below).

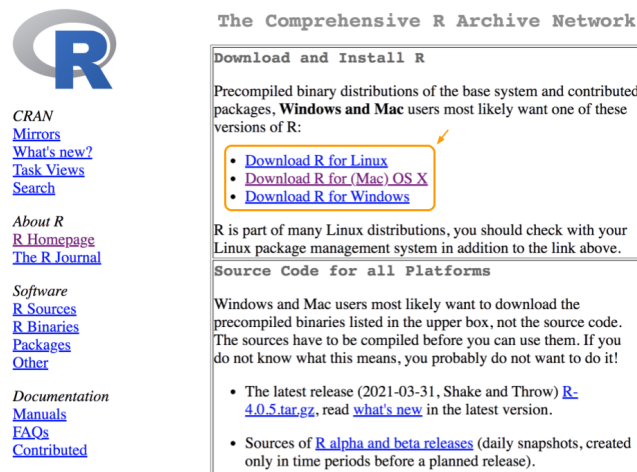


Figure 1.4: R is available for MacOS, Windows, and Linux

Step 3) Click on the download option that corresponds to your operating system (e.g. Linux, Mac, or Windows). In my case, I have a Mac computer, which explains why the Mac OS-X link is highlighted in the above screenshot.

For most users, you will want to install the **Latest release**, which in the screenshot above happens to be R 4.0.5 "Shake and Throw". Keep in mind that by the time you read this book, R will very likely have a more recent version.


Step 4) Click on the package link, which in the screenshot corresponds to R-4.0.5.pkg. This is the link of a compressed file that contains the binary code. Before installing a given version of R, read the description of the release to make sure the operating system in your computer is compatible with a specific version of R.

After clicking on the R-4.0.5.pkg link, the compressed file will be downloaded to your computer.

Step 5. Click on the downloaded file. An installation wizard will open automatically, ready to guide you through the installation process, step by step (see image below).

In most cases, you will want to use the default settings. Personally, I've been using the default settings for several years without having the need to customize anything.

At the end of the installation, if everything went well, you should be able to see a successful message (see figure below):



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) [here](#). Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the [old](#) directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

Package binaries for R versions older than 3.2.0 are only available from the [CRAN archive](#) so users of such versions should adjust the CRAN mirror setting (<https://cran-archive.r-project.org>) accordingly.

R 4.0.5 "Shake and Throw" released on 2021/03/31

Please check the SHA1 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type `openssl sha1 R-4.0.5.pkg` in the *Terminal* application to print the SHA1 checksum for the R-4.0.5.pkg image. On Mac OS X 10.7 and later you can also validate the signature using `pkgutil --check-signature R-4.0.5.pkg`

Latest release:

R-4.0.5.pkg (notarized and signed)

SHA1:
hash: 2f683b3c10f1a9aad927236636abef02285b6132
(ca. 85MB)

R 4.0.5 binary for macOS 10.13 (High Sierra) and higher, signed and notarized package. Contains R 4.0.5 framework, R.app GUI 1.74

Figure 1.5: CRAN download for Mac

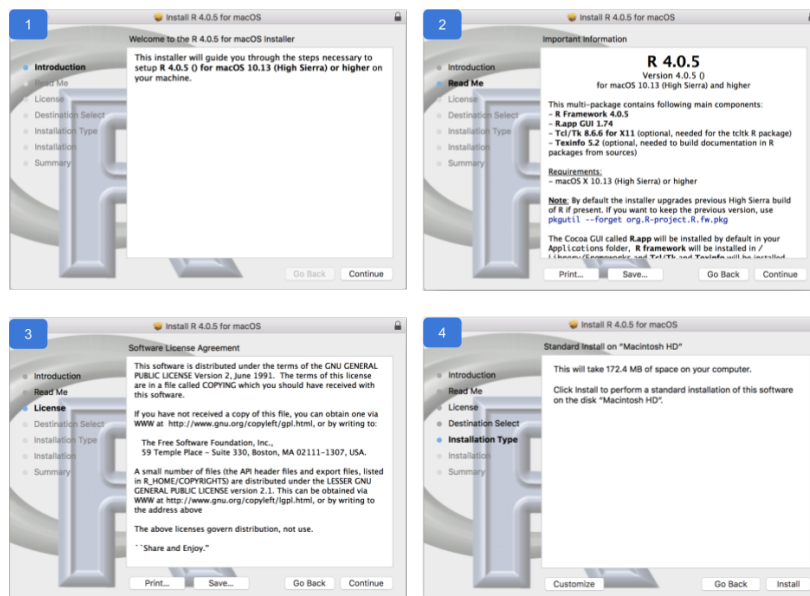


Figure 1.6: R Installation wizard for Mac

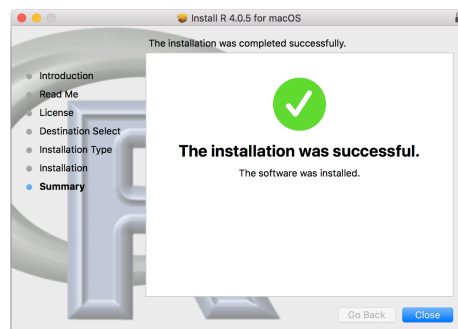


Figure 1.7: R Installation wizard for Mac

Chapter 2

Installing RStudio

In addition to R, the other program you will need to have installed in your machine is RStudio.

Technically speaking, RStudio is an *Integrated Development Environment* (IDE) developed by Posit. An IDE is just the fancy term that is used for a program that provides a nice working space that make your life easier while writing code, running analysis, and making graphs. In addition, you can also use RStudio to create all sorts of reports, documents, slides, and web apps.

2.1 Download RStudio

To download the Desktop version of RStudio follow the steps listed below.

Step 1) Go to **Posit’s** download webpage:

<https://posit.co/downloads/>

At the time of this writing, there are two options of RStudio Desktop: 1) the Free version, and 2) the Pro version.

Step 2) Choose the **Free** version of RStudio Desktop (see image below), and click on the “DOWNLOAD” button.

Step 3) Select the version that matches your operating system (e.g. Windows, macOS, linux). Double check that the operating system in your computer is compatible with a specific version of RStudio.

Once the installation of RStudio is completed, you should be able to open a new session in RStudio, and start interacting with R.

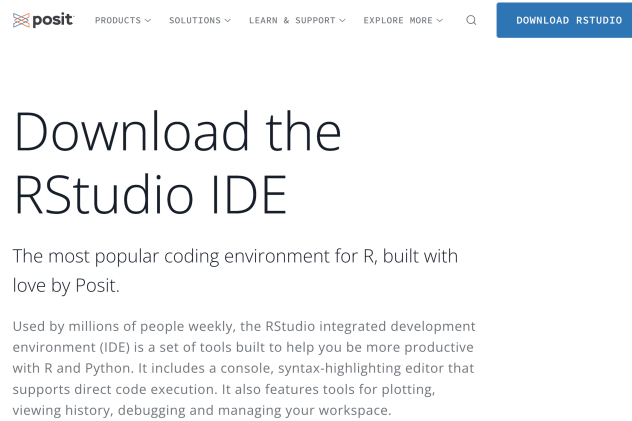


Figure 2.1: RStudio download options

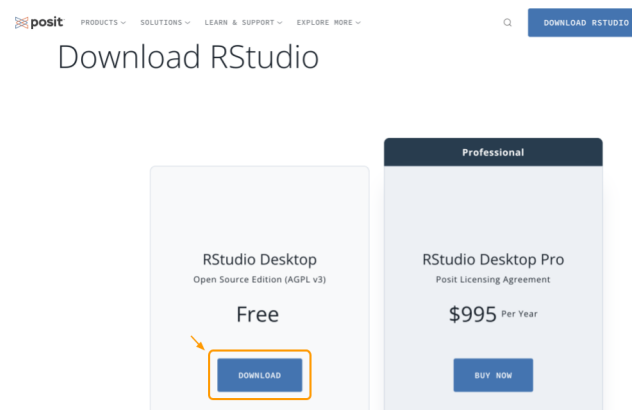
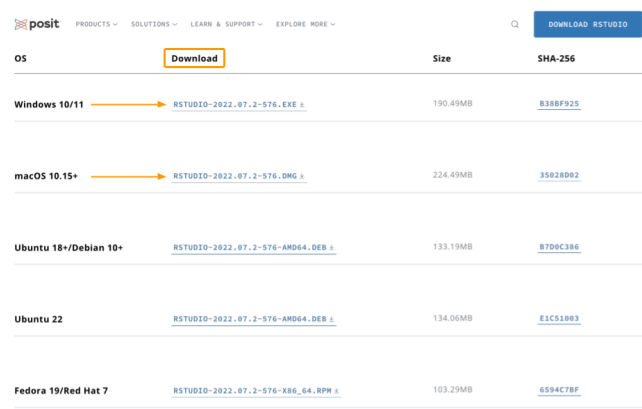


Figure 2.2: Choose RStudio free desktop



OS	Download	Size	SHA-256
Windows 10/11	RSTUDIO-2022.07.2-576.EXE	190.49MB	8388F925
macOS 10.15+	RSTUDIO-2022.07.2-576.DMG	224.49MB	35028002
Ubuntu 18+/Debian 10+	RSTUDIO-2022.07.2-576-AMD64.DEB	133.19MB	B700C386
Ubuntu 22	RSTUDIO-2022.07.2-576-AMD64.DEB	134.06MB	E1C51003
Fedora 19/Red Hat 7	RSTUDIO-2022.07.2-576-X86_64.RPM	103.29MB	6594C7BF

Figure 2.3: RStudio Desktop versions

Chapter 3

First Contact with R

If you are new to R and don't have any programming experience, then you should read this chapter in its entirety. If you already have some previous experience working with R and/or have some programming background, then you may want to skim over most of the introductory chapters of part I.

This chapter, and the rest of the book, assumes that you have installed both R and RStudio in your computer. If this is not the case, then go to chapters Installing R and Installing RStudio to follow the steps for downloading and installing these programs.

R comes with a simple built-in graphical user interface (GUI), and you can certainly start working with it right out of the box. That is actually the way I got my first contact with R back in 2001 during my senior year in college. Nowadays, instead of using R's GUI, it is more convenient to interact with R using a third party software such as RStudio.

I describe more introductory details about RStudio in the next chapter A Quick Tour Around RStudio. For now, go ahead and launch RStudio in your computer.

3.1 First Contact with R (via RStudio)

When you open RStudio, you should be able to see its layout organized into quadrants officially called *panes*. The very first time you launch RStudio you will only see three panes, like in the screenshot below.

To help you break the ice with R, it's better if we start working directly on the **Console**.

As you can tell from the following screenshot, the console is located in the left-hand side quadrant of RStudio. Keep in mind that your RStudio's console pane may be located in a different quadrant.

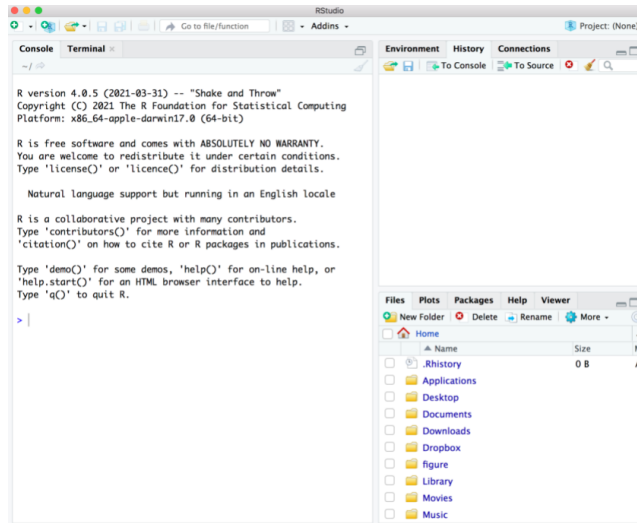


Figure 3.1: Screenshot of RStudio when launched for the first time.

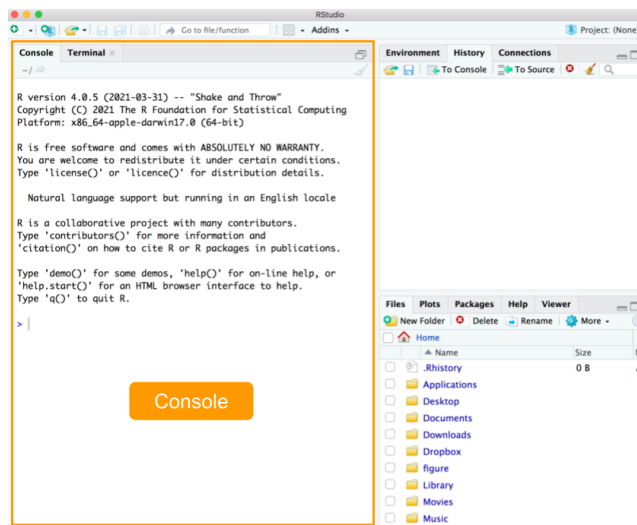


Figure 3.2: Console quadrant in RStudio.

Technically speaking, the console is a terminal where a user inputs commands and views output. Simply put, this is where you can directly interact with R by typing commands, and getting the output from the execution of the commands.

3.1.1 R as a scientific calculator

This first activity is dedicated for readers with little or no programming experience, especially those of you who have never used software in which you have to type commands. The idea is to start typing simple things in the **console**, basically using R as a scientific calculator.

Here's a toy example. Consider the monthly bills of an undergraduate student:

- cell phone \$80
- transportation \$20
- groceries \$527
- gym \$10
- rent \$1500
- other \$83

You can use R to find the student's total expenses by typing these commands in the console:

```
80 + 20 + 527 + 10 + 1500 + 83
```

There is nothing surprising or fancy about this piece of code. In fact, it has all the numbers and all the + symbols that you would use if you had to obtain the total expenses by using the calculator in your cellphone.

3.1.2 Assigning values to objects

Often, it will be more convenient to create **objects**, sometimes also called **variables**, that store one or more values. To do this, type the name of the object, followed by the assignment or “arrow” operator `<-`, followed by the assigned value. By the way, the arrow operator consists of a left-angle bracket `<` (or “less than” symbol) and a dash or hyphen symbol `-`.

For example, you can create an object **phone** to store the value of the monthly cell phone bill, and then inspect the object by typing its name:

```
phone <- 80
phone
> [1] 80
```

All R statements where you create objects are known as **assignments**, and they have this form:

```
object <- value
```

this means you assign a **value** to a given **object**; one easy way to read the previous assignment is “phone gets 80”.

Alternatively, you can also use the equals sign = for assignments:

```
transportation = 20
transportation
> [1] 20
```

As you will see in the rest of the book, I've written most assignments with the arrow operator <-. But you can perfectly replace them with the equals sign =. The opposite is not necessarily true. There are some especial cases in which an equals sign cannot be replaced with the arrow, but we'll talk about this later.

Pro tip. RStudio has a keyboard shortcut for the arrow operator<-:

- Windows & Linux users: **Alt + -**
- Mac users: **Option + -**

In fact, there is a large set of keyboard shortcuts. In the menu bar, go to the *Help* tab, and then click on the option *Keyboard Shortcuts Help* to find information about all the available shortcuts.

3.1.3 Object Names

There are certain rules you have to follow when creating objects and variables. Object names cannot start with a digit and cannot contain certain other characters such as a comma or a space.

The following are invalid names (and invalid assignments)

```
# cannot start with a number
5variable <- 5

# cannot start with an underscore
_invalid <- 10

# cannot contain comma
my,variable <- 3

# cannot contain spaces
my variable <- 1
```

People use different naming styles, and at some point you should also adopt a convention for naming things. Some of the common styles are:

```
snake_case

camelCase

period.case
```

Pretty much all the objects and variables that I create in this book follow the “snake_case” style. It is certainly possible that you may end up working with a team that has a style-guide with a specific naming convention. Feel free to try various styles, and once you feel comfortable with one of them, then stick to it.

3.1.4 Case Sensitive

R is case sensitive. This means that `phone` is not the same as `Phone` or `PHONE`

```
# case sensitive
phone <- 80
Phone <- -80
PHONE <- 8000

phone + Phone
> [1] 0

PHONE - phone
> [1] 7920
```

Again, this is one more reason why adopting a naming convention early on in a data analysis or programming project is very important. Being consistent with your notation may save you from some headaches down the road.

3.1.5 Calling Functions

Like any other programming language, R has many functions. To use a function just type its name followed by parenthesis. Inside the parenthesis you typically pass one or more inputs. Most functions will produce some type of output:

```
# absolute value
abs(10)
abs(-4)

# square root
sqrt(9)

# natural logarithm
log(2)
```

In the above examples, the functions are taking a single input. But often you will be working with functions that accept several inputs. The `log()` function is one them. By default, `log()` computes the natural logarithm. But it also has the `base` argument that allows you to specify the base of the logarithm, say to `base = 10`

```
log(10, base = 10)
> [1] 1
```

3.1.6 Comments in R

All programming languages use a set of characters to indicate that a specific part or lines of code are **comments**, that is, things that are not to be executed. R uses the hash or pound symbol `#` to specify comments. Any code to the right of `#` will not be executed by R.

```
# this is a comment
# this is another comment
2 * 9

4 + 5 # you can place comments like this
```

You will notice that I have included comments in almost all of the code snippets shown in the book. To be honest, some examples may have too many comments, but I've done that to be very explicit, and so that those of you who lack coding experience understand what's going on. In real life, programmers use comments, but not so much as I do in the book. The main purpose of writing comments is to describe—conceptually—what is happening with certain lines of code. Some would even argue that comments should only be used to express not the what but the **why** a developer is doing something. In case of doubt, especially if you don't have a lot of programming experience, I think it's better to err on the side of caution by adding more comments than including no comments whatsoever.

3.2 Getting Help

Because we work with functions all the time, it's important to know certain details about how to use them, what input(s) is required, and what is the returned output.

So how do you find all this information technically known as a function's **documentation**? There are several ways to access this type of information.

If you know the name of a function you are interested in knowing more about, you can use the function `help()` and pass it the name of the function you are looking for:

```
# documentation about the 'abs' function
help(abs)

# documentation about the 'mean' function
help(mean)
```

Alternatively, you can use a shortcut using the question mark `?` followed by the name of the function:

```
# documentation about the 'abs' function
?abs
```

```
# documentation about the 'mean' function
?mean
```

`help()` and `?` only work if you know the name of the function you are looking for. Sometimes, however, you don't know the name of the function but you may know some keyword(s). To look for related functions associated to a keyword, use `help.search()` or simply type double question marks `??`

```
# search for 'absolute'
help.search("absolute")
```

```
# alternatively you can also search like this:
??absolute
```

Notice the use of quotes surrounding the input name inside `help.search()`

Often overlooked by beginners but extremely helpful is to understand the anatomy of the information displayed in the technical documentation. The content is typically organized into seven sections listed below (although sometimes there may be less or more sections)

- Title
- Description
- Usage of function
- Arguments
- Details
- See Also
- Examples

The three screenshots below show the “Help” or technical documentation of the `log()` function. This information is in RStudio's **Help** tab, located in the pane that contains other tabs such as **Files**, **Plots**, **Packages**.

3.3 Installing Packages

R comes with a large set of functions and packages. A package is a collection of functions that have been designed for a specific purpose. One of the great advantages of R is that many analysts, scientists, programmers, and users can create their own packages and make them available so that everybody can use them. R packages can be shared in different ways. The most common way to share a package is to submit it to what is known as **CRAN**, the *Comprehensive R Archive Network*.

You can install a package using the `install.packages()` function. To do this, I recommend that you **run this command directly on the console**. In other words, do not include this command in a source file (e.g. R script file, `Rmd` file). The reason for running this command directly on the console is to avoid getting

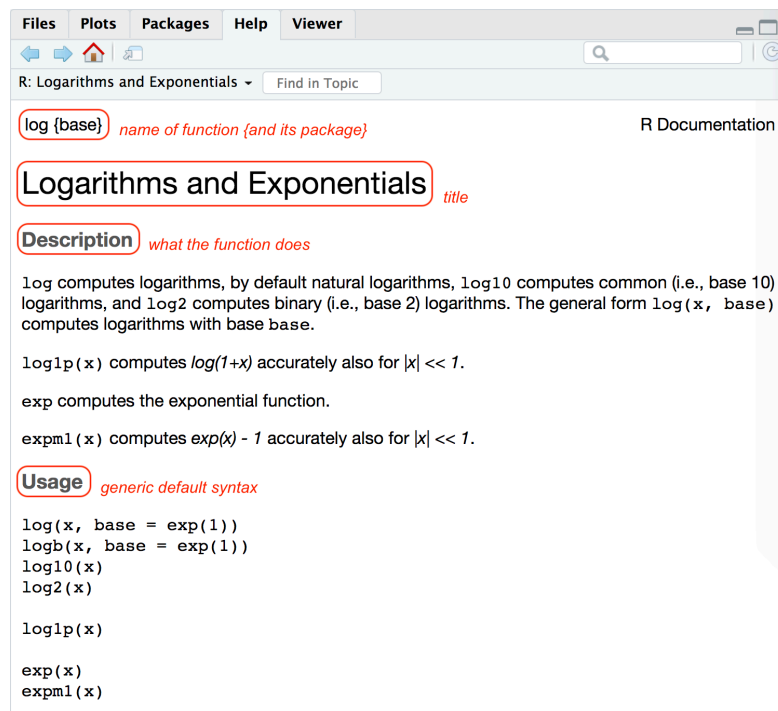


Figure 3.3: Help documentation for the log function (part 1)

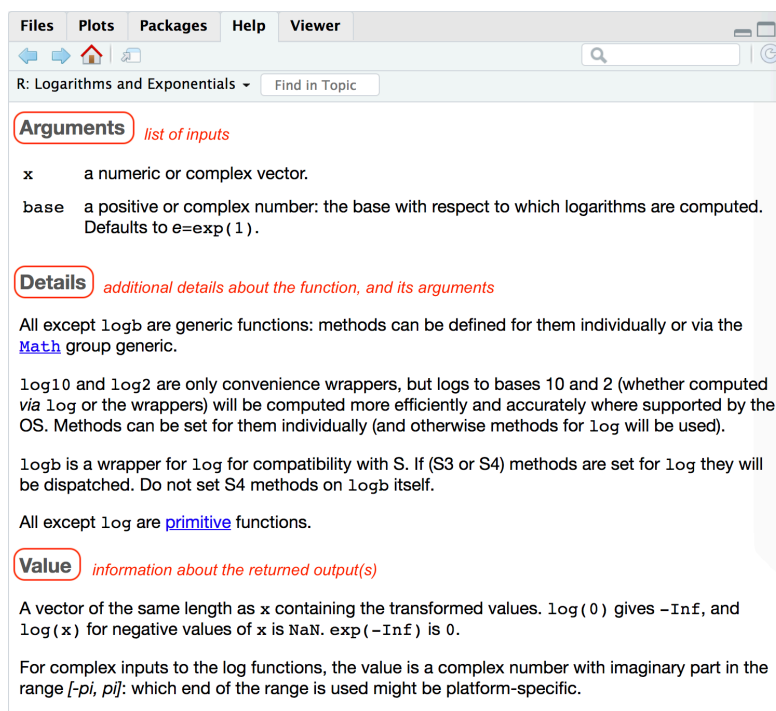


Figure 3.4: Help documentation for the log function (part 2)

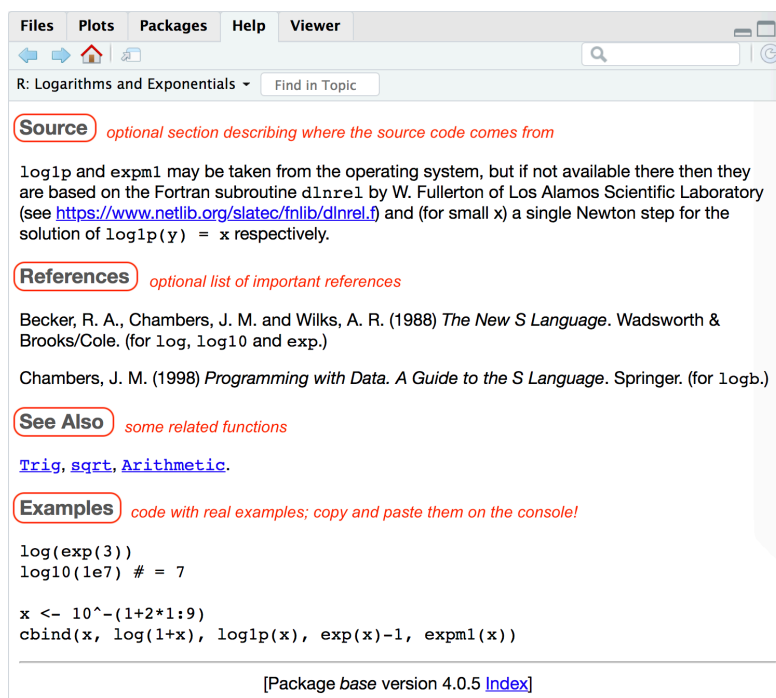


Figure 3.5: Help documentation for the log function (part 3)

an error message when running code from a source file.

To use `install.packages()` just give it the name of a package, surrounded by quotes, and R will look for it in CRAN, and if it finds it, R will download it to your computer.

```
# installing (run this on the console!)  
install.packages("knitr")
```

You can also install a bunch of packages at once by placing their names, each name separated by a comma, inside the `c()` function:

```
# run this command on the console!  
install.packages(c("readr", "ggplot2"))
```

Once you installed a package, you can start using its functions by *loading* the package with the function `library()`. For better or worse, `library()` allows you to specify the name of the package with or without quotes. Unlike `install.packages()` you can only specify the name of one package in `library()`

```
# (this command can be included in an Rmd file)  
library(knitr)      # without quotes  
library("ggplot2") # with quotes
```

By the way, you only need to install a package once. After a package has been installed in your computer, the only command that you need to invoke in order to use its functions is the `library()` function.

3.4 Exercises

1) Here's the list of monthly expenses for a hypothetical undergraduate student

- cell phone \$80
 - transportation \$20
 - groceries \$550
 - gym \$15
 - rent \$1500
 - other \$83
- a) Using the **console** pane of RStudio, create objects (i.e. variables) for each of these expenses listed above, and then create an object **total** with the sum of the expenses.
- b) Assuming that the student has the same expenses every month, how much would she spend during a school “semester”? (assume the semester involves five months). Write code in R to find this value.

- c) Using the same assumption about the monthly expenses, how much would she spend during a school “year”? (assume the academic year is 10 months). Write code in R to find this value.
- 2) Use the function `install.packages()` to install packages `"stringr"`, `"RColorBrewer"`, and `"bookdown"`
- 3) Write code in the console to calculate: $3x^2 + 4x + 8$ when $x = 2$
- 4) Calculate: $3x^2 + 4x + 8$ but now with a numeric sequence for x using `x <- -3:3`
- 5) Find out how to look for information about math binary operators like `+` or `^` (without using `?Arithmetic`). *Tip*: quotes are your friend.

Chapter 4

A Quick Tour Around RStudio

As I mentioned in the previous chapter, R comes with a simple built-in graphical user interface, or *GUI* for short. While you can use this interface to work with R, it is more convenient if you interact with R using a third party software such as RStudio.

Technically speaking, RStudio is an IDE which is the acronym for *Integrated Development Environment*. This is just the fancy name for any software application that provides comprehensive facilities to programmers for making their lives easier when writing code and developing programs.

Simply put, you can think of RStudio as a “workbench” that gives you an organized working space for interacting with R, while taking care of many of the little tasks that can be a hassle.

4.1 First Contact with RStudio

When you open RStudio, you should be able to see its layout organized into quadrants officially called *panes* (or panels).

The very first time you launch RStudio you will only see three panes, like in the screenshot below.

As you can tell from the previous screenshot, the left-hand side shows the Console pane which is what we used in the previous chapter to write a handful of simple commands, execute them, and inspect the output provided by R.

If RStudio only displays three panes, why do I call them “quadrants”? Where is the fourth pane? Well, to see the extra pane you need to open a file. One way to do this is by clicking on the icon of a blank file with a green plus sign.

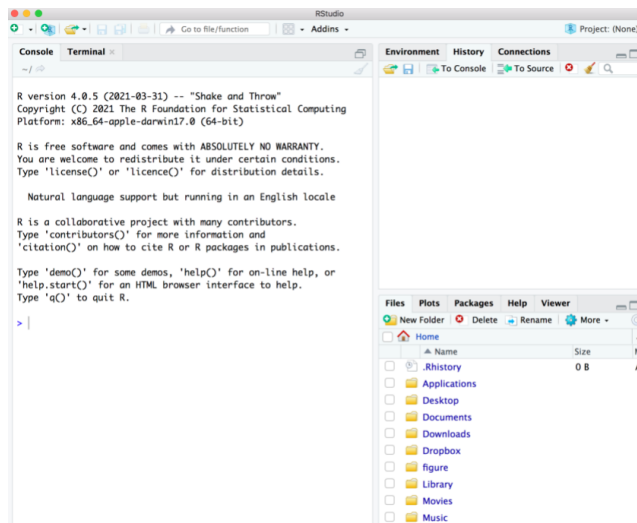


Figure 4.1: Screenshot of RStudio when launched for the first time.

This button is located in the top-left corner of the icons menu bar of RStudio. A drop-down menu with a long list of available file formats will be displayed, the first option being an “R Script” file (see image below).

Once you open a (text) file, the layout of RStudio will show the Editor quadrant, officially called the *Source* pane, like in the following screenshot.

The visual appearance of RStudio’s quadrants can be a bit intimidating for beginners. But fear not. In the above screenshot, the panes are:

- Source or editor pane (top left quadrant)
- Console pane (bottom left quadrant)
- Environment/History/Connections pane (top right quadrant)
- Files/Plots/Packages/Help pane (bottom right quadrant)

Pro tip: among many other things, you can change the default location of the panes. If you are interested in knowing what customizing options are available in RStudio, visit the following link:

<https://support.rstudio.com/hc/en-us/articles/200549016-Customizing-RStudio>

If you have no previous programming experience, you don’t have to customize anything right now. It’s better if you wait some days until you get a better feeling of the working environment. You will probably be experimenting (trial and error) some time with the customizing options until you find what works for you.

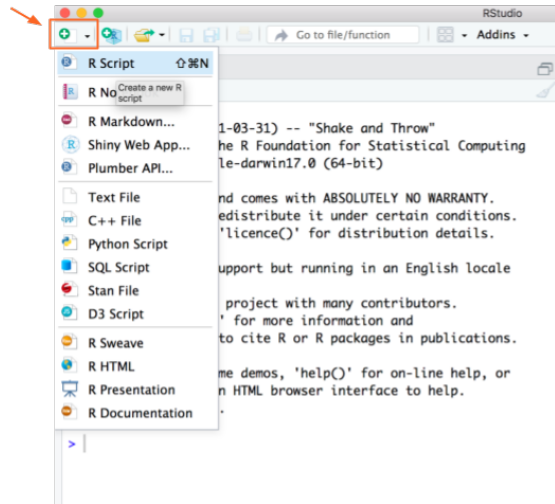


Figure 4.2: Opening a new (text) file in RStudio.

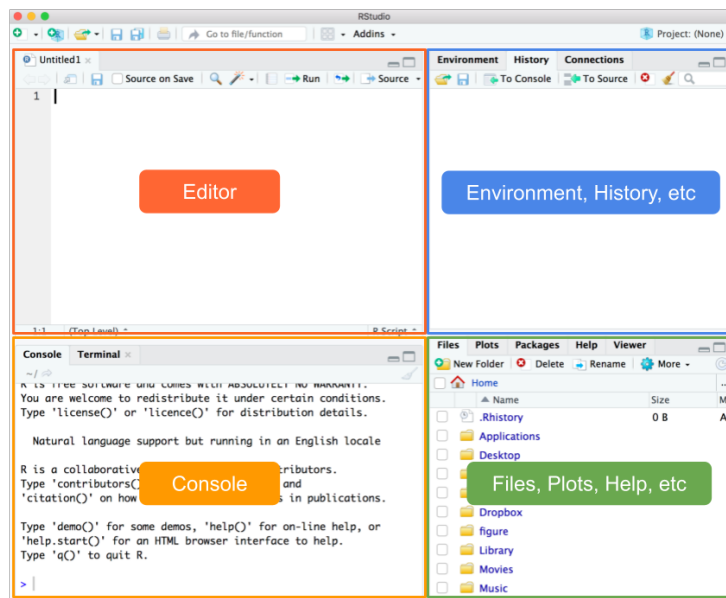


Figure 4.3: RStudio layout organized into quadrants.

4.2 RStudio Panes in a Nutshell

Sooner or later you will be using all four panes in RStudio. Most programming activities will require working with both the **Source** and **Console** panes. Certain operations will involve using the **Files** tab. Occasionally you will also use one of the tabs in the **Environment/History/etc** pane. The set of specific tabs that you have to use really depends on the type of work you plan to carry out. You will have time to learn the basics—and not so basics—of every pane throughout the book. The more time you spend in RStudio, and the more you use it, the more features you will discover about it.

4.2.1 Console

The **Console** is supposed to be the terminal—or the place—where you type in commands, which R then executes, and where the output of those commands is typically displayed. The truth is that most programmers don't write commands *directly* in the console. Instead, what we use is the **Source** pane to write commands in a text file (e.g. an R-Script file, an R-Markdown file), and then execute the commands from that pane.

The reason for writing commands in a text file and not directly in the console, is because of convenience and organization. *Convenience* because, as you will see, many commands involve writing several lines of code which can be tricky to write them correctly just by typing in the console. *Organization* in the sense that having all your commands in a text file makes it easy to store your code, keep track of all the work you do, build upon it, and share it with others.

So, knowing that programmers rarely make direct use of the **Console**, when do you actually use this pane? I don't know about the rest of programmers but I can tell you how I personally use it.

One common use of the **Console** is when I want to calculate basic things like the monthly balance in my credit cards, or the overall score for one of the students in the classes I teach, or some other quick computation. These are types of calculations that I could perfectly perform with any scientific calculator like the one in my smartphone. But more often than not I prefer to do them in R, typically when that's the tool I have at hand (which happens almost every day).

The other typical situation in which I use the console is when I'm trying out some simple idea or testing if a certain command could work. I like to explore the feasibility of my code with a small example in the console, and then refine it or generalize it by writing code in a text file—using the **Source** pane.

4.2.2 Files, Plots, Packages, Help

The **Files** quadrant contains multiple tabs.

- **Files:** this tab lets you navigate your file system without the need of leaving RStudio. You can move to any directory or folder in your home

directory, inspect the contents of a given folder, create a new folder, and perform standard operations on files such as opening, renaming, moving, copying, and deleting a file. In addition, you can also see the working directory, or change to a different directory if you want to.

- **Plots:** this tab is used by R Graphics Devices to display any graphic or image produced by an R plotting function.
- **Packages:** this tab allows you to install and update R packages. Often, you will want to use functions from external R packages, and to do this you must first install those packages in your system. While it is possible to write commands for doing this, the **Packages** tab gives you a richer interface to see what packages are already available in your computer, what their versions are, update them if necessary, or uninstall them in case you no longer need them.
- **Help:** this is the tab that gives you access to the “help” or manual documentation of functions, objects, tutorials, and demos of a given R package. The preceding chapter contains an example of the manual documentation for the `log()` function, showing the main anatomy of the so-called *R Documentation* files.

4.2.3 Source or Editor

The **Source** pane is basically the text editor of RStudio. This is the quadrant you use to edit any text file, again, without the need to leave RStudio. The reason why is called “source” is because the text files edited in this pane are, for the most part, files that contain the commands that R will run. In other words, these files are the **source** of the commands to be executed.

4.2.4 Environment, History, Connections

The last quadrant is the pane that contains, at least, the following three tabs: **Environment**, **History**, and **Connections**.

I provide a deeper explanation of the **Environment** and **History** tabs in the following chapter Session Management. In the meantime, what you need to know about **Environment** is that this tab is used to list the objects that have been created, or that are available, in a given R session.

In turn, the **History** tab is a very useful resource that lists **all** the R commands that you have executed so far. In theory, R will track all the invoked commands since the first time you used it, unless you’ve removed the auxiliary **.Rhistory** file linked to your working directory, or unless you’ve modified the history mechanism used by your R console.

As for the **Connections** tab, this plays a more advanced (and somewhat obscure) role that I briefly discuss in part IV of the book.

4.3 Exercises

1) In RStudio, one of the panes has tabs **Files**, **Plots**, **Packages**, **Help**, **Viewer**.

- a) In the tab **Files**, what happens when you click the button with a House icon?
- b) Go to the **Help** tab and search for the documentation of the function `mean`.
- c) In the tab **Help**, what happens when you click the button with a House icon?

2) In RStudio, one of the panes has the tabs **Environment**, **History**, **Connections**.

- a) If you click on tab **History**, what do see?
- b) Find what the buttons of the menu bar in tab **History** are for.
- c) Likewise, what can you say about the tab **Environment**?

3) When you start a new R session in Rstudio, a message with similar content to the text below appears on the console (the exact content will depend on your R version):

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
```

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

```
Natural language support but running in an English locale
```

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

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

- a) What happens when you type (in the console): `license()`?
- b) What happens when you type (in the console): `contributors()`?
- c) What happens when you type (in the console): `citation()`?

- d) What happens when you type (in the console): `demo()`?

Chapter 5

Session Management

In this chapter I review some important aspects about managing your interactive session with R using RStudio.

Here's what you should always keep in mind. From the point of view of a session, all the work, activities, and actions you do with R can be classified into three categories:

- when starting a session
- during the session
- when closing a session

Because there are several things going on behind the scenes in each of the categories listed above, it is important that we talk about them—at least briefly.

5.1 Starting a Session

Starting a session can be done in two primary ways:

- launching the R application program, which will give you access to its graphical user interface; or via RStudio or any other IDE that has the ability to open an R session.
- by clicking on a file that your computer associates with R (or RStudio). For example, R-script files (with file extension `.R` or `.r`), R-Markdown files (`.Rmd` extension), R-Noweb files (`.Rnw` extension), RStudio project files (`.Rproj` extension), etc.

5.1.1 What happens when you open an R session via RStudio?

This is an important question that many users never stop to think about. However, it's worth reviewing what happens when a session is started. So let's talk about this.

- Every time you open RStudio, the console pane will display R's welcome message.
- The console is always linked to a working directory.
- Typically, the working directory of the console will be your home directory, unless you specified a different location when you installed R in your computer.
- You can change the working directory to a different directory if you want. This change can be permanent (for future sessions), or temporary (for a current session).
- To permanently modify the working directory (when a session is opened), go to the menu bar, select "RStudio" tab, click on "Preferences", and modify the "R General" options.
- To temporarily change the working directory, go to the menu bar, select the "Session" tab, and click on "Set Working Directory", or simply specify a working directory with the `setwd()` function executed from R's console.

5.1.2 Opening a session for the first time

If you are opening a session in RStudio for the very first time:

- the "Editor" pane will be collapsed, and
- all the tabs in the "Environment, History, Connections" pane will be empty

In general, when you open a session (**not** for the very first time):

- the "Environment" tab may display some objects, which means you have some existing objects in your global environment. You can also invoke the list function `ls()` to list any available objects in your current session:

```
# what objects are in my global environment?  
ls()
```

- the "History" tab may contain lines of previously used commands; if this is the case it means that there is an associated text file called `.Rhistory` in your working directory. You can also use the `history()` function to display the *Commands History*, that is, all the commands invoked in your interactive sessions:

```
# is my history of commands being tracked?  
history()
```

5.1.3 Working Directory

If you open R via an application program (e.g. launching RStudio) that lets you interact with R's console, your session will have an associated working directory, sometimes also referred to as the *current* working directory.

When you install R in your computer, during the installation process a default working directory is assigned to R. By default, this directory is your home directory. You can check whether this is the case if you run the *get working directory* function `getwd()` in your console.

```
# run this command in the console to find out  
# the working directory of your session  
getwd()
```

In RStudio, you can also look at the console pane, and inspect the text right below the **Console** tab that always displays the working directory of your session. If you see the symbols `~/` it means that the home directory (represented by the tilde) is your working directory.

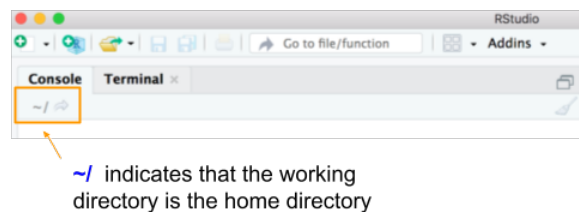


Figure 5.1: The working directory is indicated right below the Console tab.

5.2 Working During a Session

Working with R involves a series of common actions:

- writing commands (in a source document or in the console)
- executing commands (from a source doc or from the console)
- looking or examining outputs
- reading or importing files (e.g. data files, script files)
- writing or exporting/saving output to files (e.g. data files, images, results)

From the logistical point of view, it all boils down to executing commands, taking into account the following:

- where a command is being executed from
- if the command requires an input, where does that input come from?
- if the command produces an output, where does that output go to?

This is why we need to describe the following:

1. Working Directory
2. Workspace and Global Environment
3. History of commands

5.2.1 Session's Working Directory

You will be writing commands either directly in the console or in a source document (e.g. R script file, `Rmd` file, etc.)

The console is always associated to a working directory (usually your home directory).

A source document, once it has been saved, will live in some directory. **Ideally**, a source document's directory would be used as its working directory, using relative file-paths to handle all input and output resources required in the code of the source document. Unfortunately, this ideal is far from what happens in practice.

- By default, when you execute code chunks in a **saved Rmd** file, the working directory is the directory where the `Rmd` file resides in.
- By default, when you execute code from an R file, the working directory is that of the console.

5.2.2 Workspace and Global Environment

Regardless of where commands are being executed from, R will carry out all the necessary computations, and objects will be created along the way.

The collection of objects that are being created (*and kept alive*) during a session are part of what is considered to be your **workspace**.

At a more technical level, all the objects in your workspace are part of an R **environment**. To be more precise, the workspace is the **Global Environment**.

On the console, if you type `ls()`, R will display all the available objects in your workspace.

```
# available objects in your workspace
ls()
```


You can also go to the pane “Environment, History, ...” and click on the **Environment** tab to see the objects in your workspace which are displayed by default under the option “Global Environment”.

5.2.3 Commands History

When you start a session, R will track all the commands that you execute during that session. As you execute commands, they will become part of what is called the **Commands History**.

You can find the list of all used commands in the **History** tab, located in the *Environment, History, Connections* pane of RStudio. You can also access the commands history with the function `history()`.

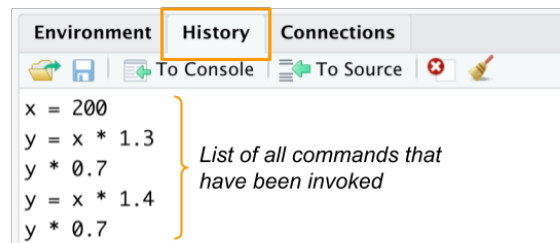


Figure 5.2: The comands history is availalble in the History tab.

By default, the history of commands are stored in a text file called `.Rhistory` that is saved in your session’s working directory.

5.3 Closing a Session

At some point, the work that you’ve done in a session will come to an end, and consequently you will **close** the session.

When closing a session, what should you do?

This is a somewhat “very personal” type of question, because it is up to you to decide what should happen to all the work that you’ve done in R.

Having said that, you can always decide whether or not to:

- save changes in your source document(s)
- save the commands history in a text file
- save the objects in your workspace (i.e. objects in Global Environment) in a binary file (native to R)

It turns out that RStudio comes with default actions that take place when you close a session:

- it will ask you if you want to save changes in your source documents
- it will ask you if you want to save the workspace in an `.RData` file (this is a file that uses R's native binary format; this is saved in your session's working directory)
- it will automatically save your commands history in a text file called `.Rhistory` (saved in your session's working directory)

Also, because of RStudio's default settings, the next time you open a new session it will:

- restore the previously open source document
- restore objects in the `.RData` file into your workspace
- give you access to all the commands stored in the `.Rhistory` file

Of course, you can change and customize the default settings of RStudio so that R/RStudio do certain things when closing a session.

Chapter 6

Intro to R Markdown Files

Most of the times you won't be working directly on the console. Instead, you will be typing your commands in some **source file**. The most basic type of source files are known as *R script files*. But there are more flavors of source files. A very convenient type of source file that allow you to mix R code with narrative is an **R markdown file** commonly referred to as **Rmd** file.

6.1 Get to know Rmd files

In the menu bar of RStudio, click on **File**, then **New File**, and choose **R Markdown**. Select the default option (Document), and click **Ok**.

Rmd files are a special type of file, referred to as a *dynamic document*, that allows to combine narrative (text) with R code. Because you will be turning in most homework assignments as **Rmd** files, it is important that you quickly become familiar with this resource.

Locate the button **Knit HTML** (the one with a knitting icon) and click on it so you can see how **Rmd** files are rendered and displayed as HTML documents.

6.1.1 What is an Rmd file?

Rmd files are a special type of file, referred to as a *dynamic document*. This is the fancy term we use to describe a document that allows us to combine narrative (text) with R code in one single file.

R markdown files use a special syntax called **markdown**. To be more precise, **Rmd** files let you type text using either: 1) R syntax for code that needs to be executed; 2) markdown syntax to write your *narrative*, and 3) latex syntax for math equations and symbols.

Rmd files are plain text files. This means that you can open an Rmd file with any text editor (not just RStudio) and being able to see and edit its contents.

The main idea behind dynamic documents is simple yet very powerful: instead of working with two separate files, one that contains the R code, and another one that contains the narrative, you use an `.Rmd` file to include both the commands and the narrative.

One of the main advantages of this paradigm, is that you avoid having to copy results from your computations and paste them into a report file. In fact, there are more complex ways to work with dynamic documents and source files. But the core idea is the same: combine narrative and code in a way that you let the computer do the manual, repetitive, and time consuming job.

Rmd files is just one type of dynamic document that you will find in RStudio. In fact, RStudio provides other file formats that can be used as dynamic documents: e.g. `.Rnw`, `.Rpres`, `.Rhtml`, etc.

6.1.2 Anatomy of an Rmd file

The structure of an `.Rmd` file can be divided in two parts: 1) a **YAML header**, and 2) the **body** of the document. In addition to this structure, you should know that `.Rmd` files use three types of syntaxes: YAML, Markdown, and R.

The *YAML header* consists of the first few lines at the top of the file. This header is established by a set of three dashes `---` as delimiters (one starting set, and one ending set). This part of the file requires you to use YAML syntax (Yet Another Markup Language.) Within the delimiter sets of dashes, you specify settings (or metadata) that will apply to the entire document. Some of the common options are things like:

- `title`
- `author`
- `date`
- `output`

The *body* of the document is everything below the YAML header. It consists of a mix of narrative and R code. All the text that is narrative is written in a markup syntax called **Markdown** (although you can also use LaTeX math notation). In turn, all the text that is code is written in R syntax inside *blocks of code*.

There are two types of blocks of code: 1) **code chunks**, and 2) **inline code**. Code chunks are lines of text separated from any lines of narrative text. Inline code is code inserted within a line of narrative text .

6.1.3 How does an Rmd file work?

Rmd files are plain text files. All that matters is the syntax of its content. The content is basically divided in the header, and the body.

- The header uses YAML syntax.
- The narrative in the body uses Markdown syntax.
- The code and commands use R syntax.

The process to generate a nice rendered document from an Rmd file is known as **knitting**. When you *knit* an Rmd file, various R packages and programs run behind the scenes. But the process can be broken down in three main phases: 1) Parsing, 2) Execution, and 3) Rendering.

- 1) Parsing: the content of the file is parsed (examined line by line) and each component is identified as yaml header, or as markdown text, or as R code.

Each component receives a special treatment and formatting.

The most interesting part is in the pieces of text that are R code. Those are separated and executed if necessary. The commands may be included in the final document. Also, the output may be included in the final document. Sometimes, nothing is executed nor included.

Depending on the specified output format (e.g. HTML, pdf, word), all the components are assembled, and one single document is generated.

6.1.4 Yet Another Syntax to Learn

R markdown (Rmd) files use markdown as the main syntax to write content. Markdown is a very lightweight type of markup language, and it is relatively easy to learn.

One of the most common sources of confusion when learning about R and Rmd files has to do with the hash symbol `#`. As you know, `#` is the character used by R to indicate comments. The issue is that the `#` character has a different meaning in markdown syntax. Hashes in markdown are used to define levels of headings.

In an Rmd file, a hash `#` that is inside a code chunk will be treated as an R comment. A hash outside a code chunk, will be treated as markdown syntax, making its associated text a given type of heading.

6.2 Code chunks

There are dozens of options available to control the execution of the code, the formatting and display of both the commands and the output, the display of images, graphs, and tables, and other fancy things. Here's a list of the basic options you should become familiar with:

- **eval**: whether the code should be evaluated
 - TRUE
 - FALSE
- **echo**: whether the code should be displayed
 - TRUE
 - FALSE
 - numbers indicating lines in a chunk
- **error**: whether to stop execution if there is an error
 - TRUE
 - FALSE
- **results**: how to display the output
 - `markup`
 - `asis`
 - `hold`
 - `hide`
- **comment**: character used to indicate output lines
 - the default is a double hash `##`
 - `"` empty character (to have a cleaner display)

6.2.1 Resources for Markdown

In RStudio's menu bar select the **Help** tab. Then click on the option **Markdown Quick Reference**.

Work through the markdown tutorial: www.markdown-tutorial.com

Your turn: After lab discussion, find some time to go through this additional markdown tutorial www.markdowntutorial.com

RStudio has a very comprehensive R Markdown tutorial: [Rstudio markdown tutorial](#)