A Minimal Book Example

Yihui Xie

2021-08-02

Contents

Chapter 1

Prologue

Dear student,

in the following, you will receive a gentle introduction to R and how you can use it to work with data. This tutorial was heavily inspired by Richard Cotton's "Learning R" (**?**) and Hadley Wickham's and Garrett Grolemund's "R for Data Science" (abbreviated with R4DS). The latter can be found online (**?**). We will not immediately start out with the packages from the `tidyverse` (although some strong points have been made in favor of doing so right from the start). I will rather try to build some sort of foundation from where we can proceed to the tidy packages in the following lessons. Hence, one can also understand this tutorial as an introduction to the `tidyverse` (**?**) (or `hadleyverse`, as it was named originally), even though I will not introduce it in this very first part. When it comes to what I would refer to as the "daily workflow" with R and RStudio, Jennifer Bryan's blog articles have been a big inspiration for me. When looking at RMarkdown, I will mainly build on "R Markdown: The Definitive Guide" – which is also freely available online (**?**).

The coming course will be structured as follows: In the beginning, I will introduce you to R and the Integrated Developer Environment (IDE) RStudio (session #1). Thereafter, the basic principles of R (such as underlying principles and data types) are taught (session #2). The process of performing data science looks like this:

Session #3 will be when you do your first steps with data. In order to analyze them, data need to be read in. Then, more often than not, they need to be reshaped properly to become tidy. Therefore, first, the theoretical principles of tidy data need to be clarified. Those data sets need to be prepared for subsequent analyses – they need to become tidy. You are introduced to techniques for achieving this. Moreover, certain analyses require transformed variables. Session #4 teaches you how to further manipulate variables and data sets in general. Session #5 will demonstrate how you can work with categorical vari-

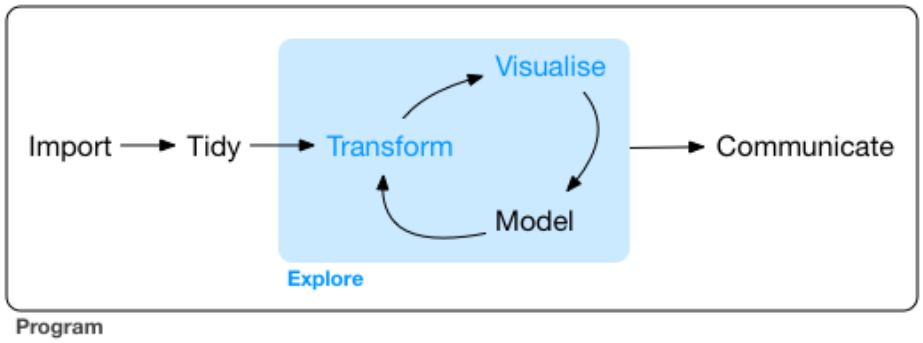


Figure 1.1: Data science as a process

ables and strings. Session #6 will demonstrate how to summarize descriptives. In this course, modeling will not be on the menu. This will, however, be covered in further depth in courses on statistical inference. In a similar vein, visualization with `ggplot2` is to be covered in session #7. Finally, you need to be able to communicate your results. This encompasses exporting tables and visualization. However, you can also write entire papers in R using RMarkdown. This is to be exemplified in session #8.

Some sessions will be longer and are split up into multiple, more digestible chunks. Each session will come with exercises that you can do at home. Those are not mandatory, yet it is highly advisable to do them thoroughly. If questions arise, I am always only one email away.

1.1 References

Chapter 2

Introduction to R

2.1 Installing R

For downloading R, just visit the website of the Comprehensive R Archive Network (CRAN). CRAN is simply a network of ftp and web servers all around the world. Here, things related to R (code, documentation, etc.) are stored and can be downloaded. On top of the page, you will find a box with three links that refer to different versions depending on your operating system. Choose the one that applies.

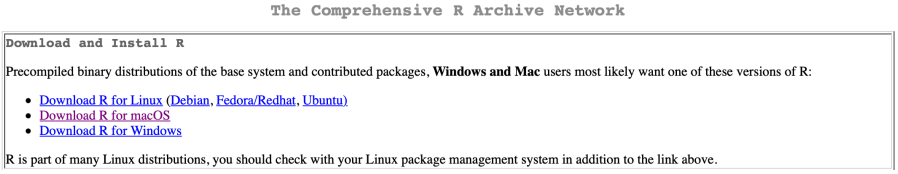
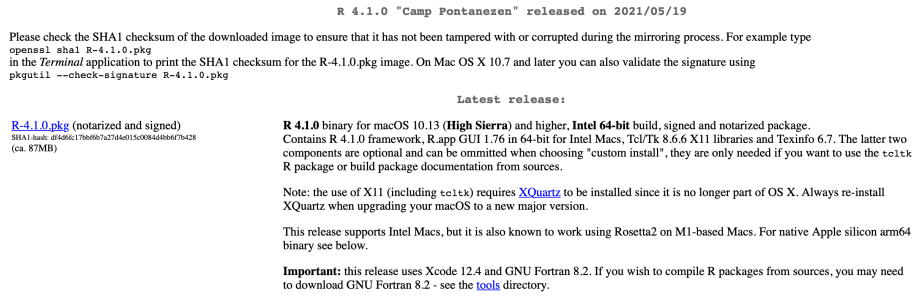


Figure 2.1: The CRAN webpage – choose the right operating system

Thereafter, just click the link for downloading the latest version (we will work with R 4.1.0). If you are on a Mac (as I am), it will look like this:

After the download is finished, just execute the installer and, once it is done, you can proceed with the installation of RStudio.

New versions of R are released multiple times per year. If you want to update your R, the process is the same as installing it from scratch: go to https://cran.r-project.org, download the latest version, install it, and that is basically it.



R 4.1.0 "Camp Pontanezen" released on 2021/05/19

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-4.1.0.pkg`
in the *Terminal* application to print the SHA1 checksum for the R-4.1.0.pkg image. On Mac OS X 10.7 and later you can also validate the signature using
`pkgutil --check-signature R-4.1.0.pkg`

Latest release:

R-4.1.0.pkg (notarized and signed)
SHA1-hash: df4d6fc17bb89b7a27d4e015c0084d4bb6f7b428
(ca. 87MB)

R 4.1.0 binary for macOS 10.13 (**High Sierra**) and higher, **Intel 64-bit** build, signed and notarized package.
Contains R 4.1.0 framework, R.app GUI 1.76 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 6.7. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires XQuartz to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your macOS to a new major version.

This release supports Intel Macs, but it is also known to work using Rosetta2 on M1-based Macs. For native Apple silicon arm64 binary see below.

Important: this release uses Xcode 12.4 and GNU Fortran 8.2. If you wish to compile R packages from sources, you may need to download GNU Fortran 8.2 - see the tools directory.

Figure 2.2: The CRAN webpage – download the latest version

2.2 Installing RStudio

When you are on a Windows or Mac machine and you click the R icon, a window (the so-called Mac or Windows GUI – graphic user interface) that looks pretty much like your machine's terminal will appear. You could now just type R code in there and execute it – and, in fact, that is how the users did it the ancient way. In my opinion, this is fairly inconvenient. Luckily, we have progressed a lot from that and come up with IDEs (integrated development environments) for R. The most popular among them is RStudio which we will use as well.

To install RStudio, just click on this link, choose the right version (i.e., RStudio Desktop – Open Source License), and hit the download button. After downloading it, you simply install it, and then you are good to go.

2.3 Setting up RStudio

After installing RStudio, you can open it just like every other application on your machine. When you open it for the very first time, a window will appear that looks like this:

As you can see, there are three panes. When you open a script (for instance, by clicking File » New File » R Script), a fourth one will appear in the upper part on the left side. If you open multiple scripts, they will be organized in tabs as well[1].

Every pane contains different things:

- By default, the console can be found in the lower left pane. This is basically how the GUI would look like. You can either manually enter commands, and execute them by pressing return, or write code in the

[1] more on scripts at the end of session 1.2.

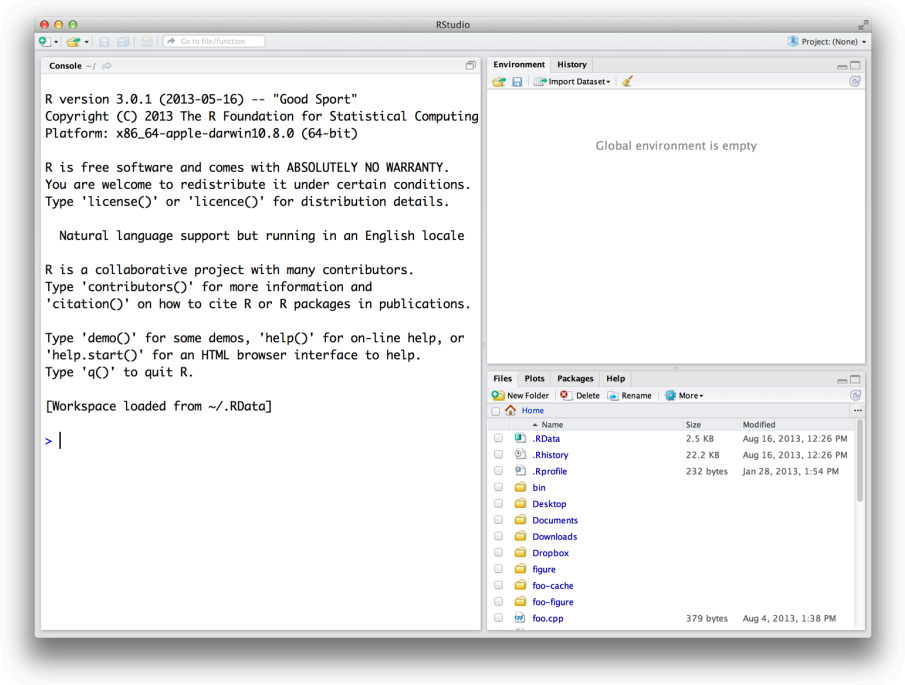


Figure 2.3: RStudio; obtained from **?**

script (upper left pane), and run it by either clicking "Run" or hitting cmd/ctrl+return (Mac/Windows). Then, the console will show, (1), the code you ran and, (2), depending on what you executed, the output.

- In the upper right pane, there are two tabs: the environment and the history. The former will show you all the objects you have defined, the latter all the commands you have executed.

- The lower left pane now consists of four tabs: "Files" simply shows you the files that are stored in the working direction you are in. "Plots" is only used if you have plotted something – and is then called automatically, so you will probably never click on it. In "Packages", as the name hints on, all the packages that are installed are listed. In theory, you could call them from there as well – but I strongly recommend not to. And finally, the "Help" tab provides you with documentations of packages etc. This sounds handy, but I hardly use it – simple googling or calling the `?[name_of_function]` has worked out best for me so far.

In the following, I will tell you more about what has appeared to work best *FOR ME*:

Disclaimer: Every useR has their own preferences when it comes to their setup. Hence, you should see the following paragraphs only as recommendations that originate from my experience.

As you may have noticed, there are a lot of different things RStudio provides you with. However, in every-day use, you will mostly use these five: the script you are actually working on. The console for seeing what your code has produced. The environment for a quick overview of the objects you are actually working with. The "Files" section for seeing the files in your working directory (this is where R projects will come in extremely handy). The "Plots" section for seeing your visualizations.
As the latter opens by itself as soon as you plot something, I mainly use the former four. However, if you use RStudio projects – as you definitely should – you will only occasionally need the "Files" section. Furthermore, your screen is wider than it is high. Hence, vertical space is scarcer than horizontal. At the heart of your coding lies your script, you should therefore give it the utmost space possible. On the right then there is space for two panes you will always have to give quick glimpses: the console and the environment. I put the former to the bottom and the latter on top. When the "Plots" section opens up, you will manually have to return to your former tab. Therefore, I put it to the lower left side where it does not bother me and I can minimize it with one click when I do not need it anymore.

My RStudio layout looks like this:

How you accomplish this? `Preferences >> Pane Layout`.

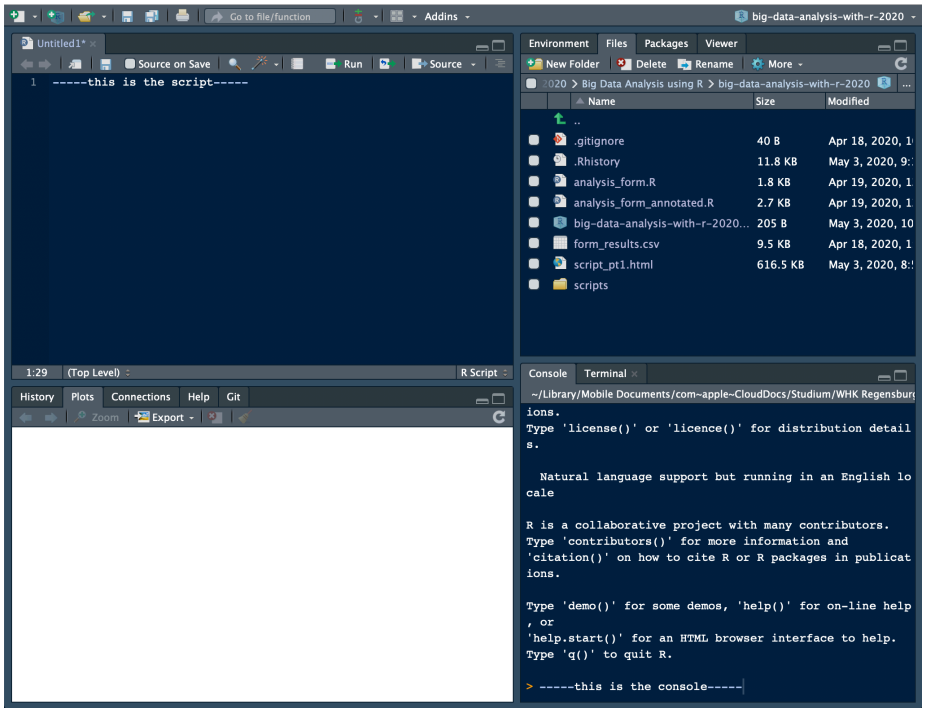


Figure 2.4: my RStudio layout

Other strongly recommended settings (ordered by Options section):

- `General`: Never save your work space, this makes you lazy (read more here)
- `Code`: some "Display" settings make your life easier; also set default encoding to UTF-8 in the "Save" section

- `Appearance`: check out some themes (I use: RStudio theme "Modern", Editor font "Courier", Font size 11, and Editor theme "Cobalt")
- `Panes layout`: feel free to set them up the way I did

More inspiration can be found here.

2.4 Some remarks on your daily workflow

As some of you are beginners, it might be hard for you to see the point in setting up projects and a GitHub account already. The intermediate and advanced users among you, who are not familiar with projects and GitHub yet though, might also wonder what they would need it for: working with R has gone pretty well in the past, so why should you change this running system?

I start out making points on why using Projects is useful. Then, I will provide step-by-step guidance on how to set them up. Since using GitHub is not that straight-forward, I will motivate why to use it and then link to a bigger tutorial covering the setup process (again by Jennifer Bryan, a statistic professor who also works at RStudio).

2.4.1 RStudio Projects

2.4.1.1 Motivation

Disclaimer: those things might not be entirely clear right away. However, I am deeply convinced that it is important that you use R and RStudio properly from the start. Otherwise it won't be as easy to re-build the right habits.

If you analyze data with R, one of the first things you do is to load in the data that you want to perform your analyses on. Then, you perform your analyses on them, and save the results in the (probably) same directory.
When you load a data set into R, you might use the `readr` package and do `read_csv(absolute_file_path.csv)`. This becomes fairly painful if you need to read in more than one data set. Then, relative paths (i.e., where you start from a certain point in your file structure, e.g., your file folder) become more useful. How you CAN go across this is to use the `setwd(absolute_file_path_to_your_directory)` function. Here, `set` stands

for set and `wd` stands for working directory. If you are not sure about what the current working directory actually is, you can use `getwd()` which is the equivalent to `setwd(file_path)`. This enables you to read in a data set – if the file is in the working directory – by only using `read_csv(file_name.csv)`. However, if you have ever worked on an R project with other people in a group and exchanged scripts regularly, you may have encountered one of the big problems with this `setwd(file_path)` approach: as it only takes absolute paths like this one: "/Users/felixlennert/Library/Mobile Documents/com$_{\text{apple}}$CloudDocs/phd/teaching/hhs-stockholm/fall2021/scripts/", no other person will be able to run this script without making any changes[2]. Just to be clear: there are no two machines which have the exact same file structure.

This is where RStudio Projects come into play: they make every file path relative. The Project file (ends with .Rproj) basically sets the working directory to the folder it is in. Hence, if you want to send your work to a peer or a teacher, just send a folder which also contains the .Rproj file and they will be able to work on your project without the hassle of pasting file paths into `setwd()` commands.

2.4.1.2 How to create an RStudio Project?

I strongly suggest that you set up a project which is dedicated to this course.

1. In RStudio, click File » New Project…
2. A windows pops up which lets you select between "New Directory", "Existing Directory", and "Version Control." The first option creates a new folder which is named after your project, the second one "associates a project with an existing working directory," and the third one only applies to version control (like, for instance, GitHub) users. I suggest that you click "New Directory".
3. Now you need to specify the type of the project (Empty project, R package, or Shiny Web Application). In our case, you will need a "new project." Hit it!

4. The final step is to choose the folder the project will live in. If you have already created a folder which is dedicated to this course, choose this one, and let the project live in there as a sub-directory.
5. When you write code for our course in the future, you *first* open the R project – by double-clicking the .Rproj file – and then create either a new script or open a former one (e.g., by going through the "Files" tab in the respective pane which will show the right directory already.)

[2]This becomes especially painful if you teach R to your students and have to grade 20 submissions and, hence, have to paste your personal directory's file path into each of these submissions.

2.5 Further links

- Hadley Wickham and Garrett Grolemund wrote an entire chapter in R4DS on Projects.

- Need more motivation? Jennifer Bryan tells you under which circumstances she would set your machine on fire: Project-oriented workflow.
- If you have created your project folder and are now unsure how to structure it, read Chris von Csefalvay's blog post on how to do it.

Chapter 3

Before you start: R scripts and RMarkdown

In this course, you will work with two sorts of documents to store your code in: R scripts (suffix `.R`) and RMarkdown documents (suffix `.Rmd`). In the following, I will briefly introduce you to both of them.

3.1 R scripts

The console, where you can only execute your code, is great for experimenting with R. If you want to store it – e.g., for sharing – you need something different. This is where R scripts come in handy. When you are in RStudio, you create a new script by either clicking `File >> New File >> R Script` or ctrl/cmd+shift+n. There are multiple ways to run code in the script:
* cmd/ctrl+return (Mac/Windows) – execute entire expression and jump to next line
* option/alt+return (Mac/Windows) – execute entire expression and remain in line
* cmd/ctrl+shift+return (Mac/Windows) – execute entire script from the beginning to the end (rule: every script you hand in or send to somebody else should run smoothly from the beginning to the end)

If you want to make annotations to your code (which you should do because it makes everything easier to read and understand), just insert '#' into your code. Every expression that stands to the right of the '#' sign will not be executed when you run the code.