# Analyzing Financial and Economic Data with R - Online Version

Marcelo S. Perlin (marcelo.perlin@ufrgs.br)

2023-12-13

**Analyzing Financial and Economic Data with R**

by **Marcelo Scherer Perlin**

**History of editions:**

# Contents

```
## Warning in install.packages :
##    package 'Rbitcoin' is not available for this version of R
##
## A version of this package for your version of R might be available elsewh
## see the ideas at
## https://cran.r-project.org/doc/manuals/r-patched/R-admin.html#Installing-
```

# About New Edition

My plan is to keep updating the content of this book every two years. Its been fun to see how much can change in the R ecosystem in just a couple of years. Here are the main updates in this revision:

**New pipeline operator** A new pipeline operator (`|>`) was introduced in R version 4.1.0. While the old pipeline from **{magrittr}** (Bache and Wickham, 2022) can still be found in the wild, my best bet is that, given its native quality and ease of use, the new pipeline will dominate the scene.

**New packages** Many of the packages used in previous editions have changed over the years. A couple of packages were dropped from CRAN, and others were substituted by upgraded versions.

**New book package** I rewrote all functions in package **{afedR3}** (Perlin, 2023b) towards a modular approach, facilitating the future maintenance of the book content. It also includes a testing framework, which will make sure all content in the book is presented as it should.

I hope you enjoy this new edition. Its been a great pleasure to evolve with the book, and I hope I can keep maintaining it over the next decades.

Marcelo S. Perlin,

Porto Alegre, Brazil, 2023-03-08

# Preface

Since you are reading this book, you are likely a data analyst looking for alternative and more efficient ways to add value to your organization, an undergraduate or graduate student in the first steps of learning data science, or an experienced researcher, looking for new computational tools. In any case, be assured that you are in the right place. **This book will teach you how to use R and RStudio for data analysis in finance and economics**.

The first version of the book originates from the class material I teach my postgraduate students in my university. By observing students learning and using R in the classroom, I frequently see the positive impact of technology on their careers. They spend less time doing repetitive and soul-crushing spreadsheet data chores, and more time thinking about their analysis and learning new tools. This book my humble attempt to go beyond the local classroom and reach an international audience.

Another motivation for writing this book is my personal experience using code from other researchers. Usually, the code is not well-organized, lacks clarity, and, possibly, only works in the computer of its author! After being constantly frustrated, I realized the work required to figure out the code of other researchers would take more time than writing the procedure myself. These cases hurt the development of science, as one of its basic principles is the **reproducibility of experiments**. As researchers are expected to be good writers, it should also be expected that their code is in a proper format and readable by other people. With this book, I will tackle this problem by presenting a code structure focused on scientific reproducibility, organization, and usability.

In this book, we will not work on the advanced uses of R. The content will be limited to simple and practical examples. One challenge I had while writing

this book was defining the boundary between introductory and advanced material. Wherever possible, I gradually dosed the level of complexity. For readers interested in learning advanced features and inner workings of R, I suggest the book Venables et al. (2004), Teetor (2011) and Wickham (2019).

This is what you'll learn from this book:

**Using R and RStudio** In chapter 01 we will discuss the use of R as a programming platform designed to solve data-related problems in finance and economics. In chapter 02 we will explore basic commands and functionalities that will increase your productivity as a data analyst.

**Importing financial and economic data** In chapters 04 and 05 we will learn to import data from local files, such as an Excel spreadsheet, or the internet, using specialized packages that can download financial and economic data such as stock prices, economic indices, the US yield curve, corporate financial statements, and many others.

**Cleaning, structuring and analyzing the data with R** In    chapters 06 and 07 we will concentrate our study on the ecosystem of basic and advanced classes of objects within R. We will learn to manipulate objects such as numeric vectors, dates and whole tables. In chapters 08 and 09 we'll learn to use the programming to solve data-related problems such as cleaning and structuring messy data. In chapter 11 we will learn applications of the most common econometric models used in finance and economics including linear regression, generalized linear model, Arima model and others.

**Creating a visual analysis of data** In chapter 10 we'll learn to use functions from package **{ggplot2}** (Wickham et al., 2023a) to create clever visualizations of our datasets, including the most popular applications in finance and economics, time series and statistical plots.

**Reporting your results** In chapter 12 we will see how to report our data analysis using specialized packages and the *RMarkdown* technology. It includes the topic of presenting and exporting tables, figures and models to a written report.

**Writing better and faster code** In the last chapter of the book we discuss best programming practices with R. We will look at how to profile code and search for bottlenecks and improving execution time with caching strategies using package **{memoise}** (Wickham et al., 2021), C++ code with **{Rcpp}** (Eddelbuettel et al., 2023) and parallel computing with **{furrr}** (Vaughan and Dancho, 2022).

# Conventions

The format of the book was chosen to maximize learnability and memorization. Here are the conventions used throughout the text:

**Packages** Every R package used in the text will have the textual format of **{package}**. The first time a R package shows up in the text, a formal citation will also be available.

**Functions** Functions are formatted as **dplyr::glimpse()** , with the information of which package the function belongs to. This notation is simply a copy of real R code, that is, you can call functions using the same structure. The first time the function is referenced, the package name will be included, except for packages that are pre-loaded in a R session (**{base}**, **{utils}** and others).

**Code** All R code will be presented in boxes, with the code output prefixed by string R>. Inline comments are set with the symbol #. Anything on the right side of # is not evaluated by R. Here's an example, showing the contents of a `list` in R:

```r
# create a list
my_list <- list('xx', 1:5, 'dec')

# print list
print(my_list)
```

```
R> [[1]]
R> [1] "xx"
R>
R> [[2]]
R> [1] 1 2 3 4 5
R>
R> [[3]]
R> [1] "dec"
```

# Supplement Material

All the material used in the book, including code examples separated by chapters, is publicly available on the internet and distributed with an R package called **{afedR3}** (Perlin, 2023b). It includes data files and several functions that can make it easier to run the examples of the book. If you plan to write some code as you read the book, this package will greatly help your journey.

In order to install the book package in your computer, you need to execute a couple of lines of code in R. For that, copy and paste the following commands into RStudio prompt (bottom left of screen, with a ">" sign) and press enter for each command. Be aware you'll need R and RStudio installed in your computer (see section 1.4 for details).

```r
# install devtools dependency
install.packages('devtools')


# install book package
devtools::install_github('msperlin/afedR3')
```

What this code will do is to install package **{devtools}** (Wickham et al., 2022), a required dependency for installing a package from Github, which is where the book bundle is hosted. After that, a call to `devtools::install_github('msperlin/afedR3')` will install the package in your computer. You can safely ignore any warning messages about long paths during installation.

After installing package **{afedR3}** (Perlin, 2023b), you can, but its not necessary, to copy all book files to a local folder by executing the following command in R:

```r
afedR3::bookfiles_get(path_to_copy = '~/afedR3')
```

The previous code will unzip the book file into your "Documents/afedR3" folder, as the tilda (`~`) is a shortcut to your "Documents" directory[1]. If you prefer the old-fashioned way of using an internet page, you can find and download the package zip file from github[2].

A suggestion, before you read the rest of the book: go to the book website and search for the related links page at the bottom. There you will find all internet addresses highlighted in the text, including the links for the installation of R and RStudio.

# Content for Instructors

If you are an R instructor, you'll find plenty of material you can use with your classes. I made sure you get everything you need:

**Over 100 exercises** Every chapter in this book includes exercises that

---

[1]In R, you can type `path.expand('~')` to see exactly where is your "Documents" folder located.

[2]https://github.com/msperlin/afedR3

your students can practice, with solutions available in the web version of the book. Also, all exercises are available in the `exams` format, meaning that you can compile the same exercises in pdf or html. Moreover, you can export the exercises to *e-learning* platforms such as Moodle and Blackboard. See this blog post[3] for instructions on how to use it with your students.

**Web version** The first seven chapters of the book are freely available at link https://www.msperlin.com/afedr, which is more than enough material for an introductory class on R and data analysis.

All of this content is released with the MIT license, so feel free to use and abuse it, as long as you give the credits to the original author. You can find the content within the book package **{afedR3}** (see previous instructions on installation) or directly at the book site[4].

I hope you enjoy this book and find it useful for your work.

Good reading!

Marcelo S. Perlin

---

[3]https://www.msperlin.com/post/2023-03-09-compiling-exercises-afedR3/
[4]https://www.msperlin.com/publication/2020_book-afedr-en/

# Chapter 1

# Introduction

**In the digital era, information is abundant and accessible**. From the ever-changing price of financial contracts to the unstructured data of social media websites, the high volume of information creates a strong need for data analysis in the workplace. A company or organization benefit immensely when it can create a bridge between raw information from its environment and making strategic decisions. Undoubtedly, this is a prolific time for professionals skilled in using the right tools for acquiring, storing, and analyzing data.

In particular, datasets related to Economics and Finance are widely available to the public. International and local institutions, such as central banks, government research agencies, financial exchanges, and many others, provide their data publicly, either by legal obligation or to foment research. Whether you are looking into statistics for a particular country or a company, most information is just two clicks away.

Not surprisingly, it is expected that a graduate student or a data analyst has learned at least one programming language that allows him/her to do his work more efficiently. **Learning how to program is becoming a requisite for the job market**. This is where the role and contribution of R comes into play. In the next sections, I will explain what R is and why you should use it.

## 1.1   What is R

R is a programming language specially designed to resolve statistical problems and display graphical representations of data. R is a modern version

of S, a programming language originally created in Bell Laboratories (formerly AT&T, now Lucent Technologies). The base code of R was developed by two academics, **Ross Ihaka** and **Robert Gentleman**, resulting in the programming platform we have today. For anyone curious about the name, the letter R was chosen due to the common first letter of the name of their creators.

Today, R is almost synonymous with data analysis, with a large user base and consolidated modules. It is likely that researchers from various fields, from economics to biology, find in R significant preexisting code that facilitates their analysis. On the business side, large and established companies, such as *Google* and *Microsoft*, already adopted R as the internal language for data analysis. R is maintained by the **R Foundation**[1] and the **R Consortium**[2], a collective effort to fund projects for extending the programming language.

## 1.2   Why Choose R

Learning a new programming language requires a lot of time and effort. Perhaps you're wondering why you should choose R and invest time in learning it. Here are the main arguments.

First, **R is a mature and stable platform, continuously supported and intensively used in the industry**. When choosing R, you will have the computational background not only for an academic career but also to work as a data analyst in private organizations. Due to its open license, you can use R anywhere. Also, the strong support from the community means it is very unlikely the R platform will ever fade away or be substituted. Depending on your career choices, R might be the only programming language you ever need to learn.

**Learning R is easy**. My experience in teaching R allows me to confidently state that students, even those with no programming experience, have no problem learning the language and using it to create their own code. The language is intuitive and certain rules and functions can be extended to different cases. Once you understand how the software expects you to think, it become easy to traverse over different modules and functionalities.

**The engine of R and the interface of RStudio creates a highly productive environment**. The graphical interface provided by RStudio facilitates the use of R and increases productivity by introducing new features to the platform. By combining both, the user has at his disposal many

---

[1]https://www.r-project.org/foundation/

[2]https://www.r-consortium.org/

tools that facilitate the development of research scripts and other projects.

**R Packages allow the user to do many different things with R**. We will soon learn that R offers several modules that can be installed over the internet whenever necessary. These modules extend the basic language of R and enable the most diverse functionalities. Besides basic data tasks such as reading and writing, you can, for example, use R to build and publish a blog, send emails, create exams, write random jokes and poems (seriously!), and many other features. The existing external modules in R are truly an impressive achievement of the community.

**R is compatible with different operating systems and it can interface with different programming languages**. If you need to execute code from another programming language, such as *C++*, *Python*, *Julia*, it is easy to integrate it with R. Therefore, the user is not restricted to a single programming language and can easily use features and functions from others. For example, the C++ code is well known for its superior speed in numerical tasks. From an R script, you can use package **{Rcpp}** (Eddelbuettel et al., 2023) to write a C++ function and effortlessly use it within your R code.

**R is free!** The main software and all its packages are free. A generous license motivates the adoption of the R language in a business environment, where obtaining individual and collective licenses of commercial software can be costly. This means you can take R anywhere you go, regardless of whether you have a budget for software or not.

# 1.3   What Can You Do With R and RStudio?

R is a fairly complete programming language and any computational problem can be solved based on it. Given the adoption of R for different areas of knowledge, the list is extensive. With finance and economics, I can highlight the following possibilities:

- Substitute and improve data-intensive tasks from spreadsheet-like software;

- Develop routines for managing investment portfolios and executing financial orders;

- Creating tools for calculating and reporting economic indices such as inflation and unemployment;

- Performing empirical data research using statistical techniques, such as econometric models and hypothesis testing;

- Create dynamic *websites* with the **{shiny}** (Chang et al., 2021) package, allowing anyone in the world to use a computational tool created by you;

- Automate the process of writing technical reports with the **RMarkdown** and **Quarto** technology;

Moreover, public access to packages developed by users further expands these capabilities. The CRAN views website[3] offers a *Task Views* panel for the topic of Finance[4] and Econometrics[5]. There you can find the main packages to perform specific operations such as importing financial data from the internet, estimating econometric models, calculation of different risk estimates, among many other possibilities. Reading this page and the knowledge of these packages is essential for those who intend to work in Finance and Economics.

## 1.4   Installing R

Before going any further, let's install the required software on your computer. The most direct and practical way to install R is to direct your favourite internet browser to R website[6] and click the *Download* link in the left side of the page, as shown in Figure 1.1.

The next screen gives you a choice of the mirror to download the installation files. The CRAN repository (*R Comprehensive Archive network*) is mirrored in various parts of the world. You can choose one of the links from the nearest location to you. If undecided, just select the mirror *0-Cloud* (see Figure 1.2), which will automatically take you to the nearest location.

The next step involves selecting your operating system, likely to be *Windows*. From now on, due to the greater popularity of this platform, we will focus on installing R in Windows. The instructions for installing R in other operating systems can be easily found online. Regardless of the underlying platform, using R is about the same. There are a few exceptions, especially when R interacts with the file system. In the content of the book, special care was taken to choose functions that work the same way in different operating

---

[3]https://cran.r-project.org/web/views

[4]https://cran.r-project.org/web/views/Finance.html

[5]https://cran.r-project.org/web/views/Econometrics.html

[6]http://www.r-project.org/
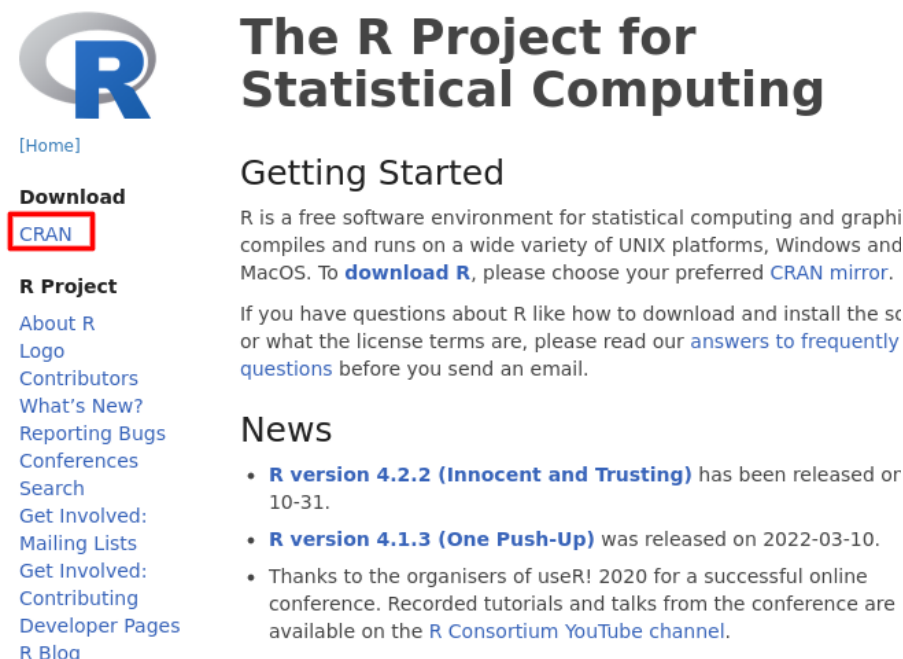
Figure 1.1: Initial page for downloading R

systems. A few exceptions are highlighted throughout the book. So, even if you are using a Mac or a flavor of Linux, you can take full advantage of the material presented here.

After clicking the link *Download R for Windows*, as in Figure 1.3, the next screen will show the following download options: *base*, *contrib*, *old.contrib* and *RTools*. The first (*base*), should be selected. It contains the download link to the executable installation file of R in *Windows*.

Some R packages requires local compilation of the files. For that, you need *RTools*, a bundle of compilers and utilities. So, you can safely install *RTools* from CRAN website.

After clicking the link *base*, the next screen will show the link to the *download* of the R installation file. After downloading the file, open it and follow the steps in the installation screen. At this time, no special configuration is required. I suggest keeping all the default choices and simply hit *accept* in the displayed dialogue screens. After the installation of R, it is strongly recommended to install RStudio, which will be addressed next.

```
                              CRAN Mirrors
```

The Comprehensive R Archive Network is available at the following URLs, please cho
statistics on the status of the mirrors can be found here: main page, windows release

If you want to host a new mirror at your institution, please have a look at the CRAN N

0-Cloud
    https://cloud.r-project.org/                    Automatic redirection to servers
                                                    sponsored by Rstudio
Argentina
    http://mirror.fcaglp.unlp.edu.ar/CRAN/          Universidad Nacional de La Plata
Australia
    https://cran.csiro.au/                          CSIRO
    https://mirror.aarnet.edu.au/pub/CRAN/          AARNET
    https://cran.ms.unimelb.edu.au/                 School of Mathematics and Statis
                                                    Melbourne
    https://cran.curtin.edu.au/                     Curtin University
Austria
    https://cran.wu.ac.at/                          Wirtschaftsuniversität Wien
Belgium
    https://www.freestatistics.org/cran/            Patrick Wessa
    https://ftp.belnet.be/mirror/CRAN/              Belnet, the Belgian research and
Brazil

Figure 1.2: Choosing the CRAN mirror

```
Download and Install R

Precompiled binary distributions of the base system and contributed
packages, Windows and Mac users most likely want one of these
versions of R:

   • Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
   • Download R for macOS
   • Download R for Windows

R is part of many Linux distributions, you should check with your Linux
package management system in addition to the link above.
```

Figure 1.3: Choosing the operating system

R for Windows

Subdirectories:

| | |
|---|---|
| base | Binaries for base distribution. This is what you want |
| contrib | Binaries of contributed CRAN packages (for R >= 3 |
| old contrib | Binaries of contributed CRAN packages for outdate |
| Rtools | Tools to build R and R packages. This is what you w Windows, or to build R itself. |

Figure 1.4: Installation options

> Be aware that R has a consistent release schedule. Every four months a new version of R is released, fixing *bugs* and implementing new solutions. There are two main types of releases, *major* and *minor*. For example, today, 2023-02-23, the latest version of R is 4.2.2. The first digit ("4") indicates the *major* release while all others are of the *minor* type. Generally, the *minor* changes are very specific and, possibly, will have little impact on your work. However, unlike *minor* releases, **major releases are fully reflected in the R package ecosystem**. Every time you install a new *major* version of R, you will have to reinstall all packages. Particularly, the problem here is that it is not uncommon that a new major release comes with package incompatibility issues. My advice is: every time a new *major* release of R comes out, **wait a few months** before installing it on your machine. Thus, the authors of the packages will have more time to update their codes, minimizing the possibility of compatibility problems.

# 1.5  Installing RStudio

The base installation of R includes its own *GUI* (graphical user interface), where we can write and execute code. However, this native interface has several limitations. RStudio Desktop substitutes the original GUI and makes access to R more practical and efficient. One way to understand this relationship is with an analogy with cars. While R is the engine of the programming language, RStudio is the body and instrument panel, which significantly improves the user experience. With RStudio you'll have code highlight, creation of projects, and much more.

The installation of RStudio is simpler than that of R. Direct your favourite

browser to Posit (formerly RStudio) website[7] and click in *Download RStudio* and then *Download RStudio Desktop*. After that, just select the installation file relative to the operating system on which you will work. This option is probably *WINDOWS Vista 7/8/10*. Note that RStudio is also available for Mac and Linux.

I emphasize that using RStudio is not essential to develop programs in R. Other interfaces are available and can be used. However, in my experience, RStudio is the interface that offers a vast range of features for the language and is widely used, which justifies its choice. If you want to explore other programming interfaces for R, one that I really enjoy and use is Microsoft's VSCode[8].

## 1.6   Resources in the Web

**The R community is vivid and engaging**. There are many authors, such as myself[9], that constantly release material about R in their blogs. It includes the announcement of new packages, analysis of real world datasets, curiosities, rants, and tutorials. R-Bloggers[10] is a website that aggregates these blogs, making it easier for anyone to access and participate. I strongly recommend to sign up for the R-Bloggers feed in RSS[11], Facebook[12] or Twitter[13]. Not only you'll be informed of what is happening in the R community, but also learn a lot by reading other people's code and articles.

Learning and using R can be a social experience. Several conferences and user groups are available in many countries. You can find the complete list in this link[14]. I also suggest looking in social platforms for local R groups in your region.

## 1.7   Structure and Organization

This book presents a practical approach to using R in finance and economics. To get the most out of it, I suggest you first try to understand what the code does and, after that, use it on your own computer. Whenever you find a piece

---

[7]https://posit.co/

[8]https://code.visualstudio.com/

[9]https://www.msperlin.com

[10]https://www.r-bloggers.com/

[11]https://feeds.feedburner.com/RBloggers

[12]https://www.facebook.com/rbloggers/?fref=ts

[13]https://twitter.com/Rbloggers

[14]https://jumpingrivers.github.io/meetingsR/index.html

of code that you do not understand, go on and study it. At first, it might seem like a daunting task but, with time, be confident that the learning process will get a lot easier as the code blocks will start to make sense and connect to each other.

Learning to program in a new platform is like learning a foreign spoken language: the use in day-to-day problems is imperative to create fluency. All the code and data used in this book is available with the installation of package **{afedR3}** (Perlin, 2023b) (see the preface for instructions on how to install it). I suggest you test the code on your computer and *play* with it, modifying the examples and checking the effect of changes in the outputs. Whenever you have a computational problem, try using R to solve it. You'll stumble and make mistakes at first. But I guarantee that, soon enough, you'll be able to write complex data tasks effortlessly.

Throughout the book, every demonstration of code will have two parts: the R code and its output. The output is nothing more than the textual result of the commands on the screen. All inputs and outputs code will be marked in the text with a special format. See the following example:

```r
# create a list
this_list <- list('abc', 1:5, 'dec')

# print list
print(this_list)
```

```
R> [[1]]
R> [1] "abc"
R>
R> [[2]]
R> [1] 1 2 3 4 5
R>
R> [[3]]
R> [1] "dec"
```

For the previous chunk of code, lines `this_list <- list('abc', 1:5, 'dec')` and `print(this_list)` are actual commands given to R. The output of this simple piece of code is the on-screen presentation of the contents of object `this_list`.

The code can also be spatially organized using newlines. This is a common strategy around arguments of functions. The next chunk of code is equivalent to the previous and will run the exact same way. Notice how we used a new line to vertically align the arguments of function `list`. You'll soon see that, throughout the book, this type of vertical alignment is constantly used.

```r
# create a list
this_list <- list('abc',
                  1:5,
                  'dec')

# print list
print(this_list)
```

```
R> [[1]]
R> [1] "abc"
R>
R> [[2]]
R> [1] 1 2 3 4 5
R>
R> [[3]]
R> [1] "dec"
```

The code also follows a well-defined structure. One decision in writing computer code is how to name objects and how to structure it. It is recommended to follow a clear pattern, so it is easy to maintain over time and be used and understood by others. For this book, a mixture of the author's personal choices with the coding style suggested by Google[15] was used. The reader, however, may choose the structure he finds more efficient and aesthetically pleasing. Like many things in life, this is a choice. We will get back at discussing code structure in chapter 13.

## 1.8  Exercises

---

01 - The R language was developed based on what other programming language?

---

02 - What are the names of the two authors of R?

---

03 - Why is R special when comparing to other programming languages, such as Python, C++, javascript and others?

---

[15]https://google.github.io/styleguide/Rguide.xml

04 - What was the reason the programming language was named R?

_____

05 - Consider the following alternatives about R and RStudio:

I - R is a mature and stable programming platform;

II - RStudio is a modern interface to R, increasing productivity;

III - R has compatibility with different programming languages;

Which alternatives are correct?

_____

06 - Once you have R and RStudio installed, head over to the CRAN package website[16] and look for technologies you use in your work. For example, if you use Google Sheets[17] ostensibly in your work, you will soon discover that there is a package in CRAN called `googlesheets4` that interacts with spreadsheets in the cloud.

_____

07 - On the CRAN site you can also install the Rtools application. What is it for?

_____

08 - Use Google to search for R groups in your region. Check if the meetings are frequent and, if you don't have a major impediment, go to one of these meetings and make new friends.

_____

09 - Go to the RBloggers website[18] and look for a topic of interest to you, such as football (*soccer*) or investments (*investments*). Read at least three of the found blog posts.

_____

10 - If you work in an institution with data infrastructure, talk to the person in charge of the IT department and verify what technologies are used. Check if, through R, it is possible to access all tables in the databases. For now there is no need to write code, yet. Just check if this possibility exists.

---

[16]https: //cloud.r- project.org/web/packages/available_packages_by_date.html
[17]https://www.google.com/sheets/about/
[18]https://www.r-bloggers.com/

# Chapter 2

# Basic Operations in R

When working with R and RStudio, there are fundamental tasks (or basic operations) that you will be repeating many times over. In this chapter we will look at these basic operations with RStudio, including:

- Rstudio interface and shortcuts
- basic R commands
- working with files
- working with related file extensions
- the autocomplete feature of RStudio.

Here, we will go through the initial steps from the viewpoint of someone who has never worked with R and possibly never had contact with other programming language. Those already familiar with the software may not find novel information here and, therefore, I suggest skipping to the next section. However, I recommended that you at least check the discussed topics so you can confirm your knowledge about the features of the software and how to use them for working smarter, and not harder. This is especially true for RStudio, which offers several tools to increase your productivity.

## 2.1   Working With R

The greatest hurdle a new user faces when developing routines in R is the format of work – the so-called development cycle. Our interaction with computers has been simplified over the years and we are currently comfortable with the *point&click* format. That is, if you want to perform an operation

on the computer, just point the *mouse* to a specific location on the screen
and click a button. Visual cues in a series of steps allow the execution of
complex tasks. Be aware, however, that this form of interaction is just one
layer above what actually happens. Behind all these *clicks*, there is a com-
mand being executed on your computer. Any common task such as opening
a *pdf* file, a spreadsheet document, directing a *browser* to a web page has an
underlying call to a code.

The *point&click* format of visual and motor interaction has its benefits in
facilitating and popularizing the use of computers. However, it is not flexi-
ble and effective when working with computational procedures such as data
analysis. A better approach would be to create a file containing several in-
structions in sequence and, in the future, simply request that the computer
**execute** this file using the recorded procedures. There is no need to do a
"scripted" point and click operation. You spend some time studying com-
mands and writing the program but, in the future, it will always execute the
recorded procedure in the same way.

Using *scripts* provides a significant gain in productivity when comparing to
a *point&click* type of interface. Going further, the risk of human error in
executing the procedure is almost nil, because the commands and their re-
quired sequence of execution are recorded in the text file and will always
be executed in the same way. This is one of the main reasons why pro-
gramming languages are popular in science. All the steps of a data-based
research, including results, can be replicated by different people, in different
computers.

While programming in R, the ideal format of work is to merge the mouse
movement with commands. R and RStudio have some functionality with the
*mouse*, but their capacity is optimized when we perform operations using
code. When a group of commands is performed in a smart way, we have an
R script that should preferably produce something important to us at the
end of its execution. In finance and economics, this can be the current price
of a stock, the value of an economic index such as inflation, the result of
academic research, among many other possibilities.

Like other software, R allows us to import data and export files. We can use
code to import a dataset stored in a local file – or the web–, analyze it and
paste the results into a technical report. Going further, we can use RStudio
and the *RMarkdown* technology to write a dynamic report, where code and
content are integrated. Needless to say that, by using the capabilities of R
and RStudio, you will work smarter and faster.

The final product of working with R and RStudio will be an R script that

produces digital elements for a data report. A good example of a simple and polished R script can be found at this link[1]. Open it and you'll see the content of a file with extension *.R* that will download stock prices of two companies and create a plot and a table. By the end of the book, you will understand what is going on in the code and how it gets the job done. Even better, you'll be able to improve it. Soon, you'll learn to execute the code on your own computer. If impatient, simply copy the text content of the link to a new RStudio R script, save it, and press `control + shift + enter`.

## 2.2 Objects in R

**In R, everything is an object, and each type of object has its properties**. For example, the daily closing prices of the IBM stock over 2023 can be represented as a numerical vector, where each element is a price recorded at the end of a trading day. Dates related to these prices can be represented as text (*string*) or as a unique `Date` class. Finally, we can represent the price data and the dates together by storing them in a single object of type `dataframe`, which is nothing more than a table with rows and columns.

While we represent data as objects in R, a special type is a `function`. It stores a pre-established manipulation of other objects available to the user. R has an extremely large number of functions, which enable the user to perform a wide range of operations. For example, the basic commands of R, available in the package **{base}** (R Core Team, 2023b), adds up to a total of 1268 functions.

Each function has its own name and a programmer can write their own functions. For example, the **sort()** function is a procedure that sorts elements within a vector. If we wanted to sort the elements of `2, 1, 4, 3, 1`, simply insert the following command in the *prompt* (left bottom side of RStudio's screen) and press *enter*:

```
my_vec <- c(2, 1, 4, 3, 1)

sorted_vec <- sort(
    x = my_vec,
    decreasing = TRUE
    )
```

---

[1]https://github.com/msperlin/afedR3/blob/main/inst/extdata/others/S_Example_Script.R

```
print(sorted_vec)
```

```
R> [1] 4 3 2 1 1
```

The **sort()** function is used with start and end parentheses. These parentheses serve to highlight the entries (*inputs*), that is, the information sent to the function to produce something that will be saved in object `sorted_vec`. Note that each entry is separated by a comma, as in `my_fct(input1, input2, input3, ...)`. We also set option `decreasing = TRUE`. This is a specific directive for the **sort()** function to order the value from highest to lowest.

Be aware that **functions are at the heart of R** and we will dedicate a large part of this book to them. You can use the available functions or write your own. You can also publish functions as a package and let other people use your code. We will discuss more about functions in chapter 8.

## 2.3   International and Local Formats

Before explaining the use of R and RStudio, it is important to highlight some rules of formatting numbers, latin characters and dates.

**decimal:** Following an international notation, the decimal point in R is defined by the period symbol (.), as in `2.5` and not a comma, as in `2,5`. If this is not the standard format in your country, you'll have issues when importing local data from text files. Sometimes, such as with storing data in Microsoft Excel files, the reading function already takes care of the conversion. This, however, is generally an exception. As a general rule of using R, only use commas to separate the inputs of a function. Under no circumstances should the comma symbol be used as the decimal point separator. Always give priority to the international format because it will be compatible with the vast majority of data.

**Latin characters:** Due to its international standard, R has problems understanding Latin characters, such as the cedilla and accents. If you can, avoid using Latin characters in the names of your variables or files. In the content of character objects (text), you can use them without problems as long as the encoding of the script is correctly specified (e.g. UTF-8, Latin1). I strongly recommend the use of the English language for writing code and defining object names. This automatically eliminates the use of Latin characters and facilitates the usability of the code by people outside of your country.

**date format:** Dates in R are structured according to the ISO 8601[2] format.

---

[2]https://www.iso.org/iso-8601-date-and-time-format.html

It follows the `YYYY-MM-DD` pattern, where `YYYY` is the year in four numbers (e.g. 2023), `MM` is the month as a number and `DD` is the day. An example of date is 2023-12-13. This, however, may not be the case in your country. When importing local data, make sure the dates are in this format. If necessary, you can convert any date to the ISO format. Again, while you can work with your local format of dates in R, it is best advised to use the international notation. The conversion between one format and another is quite easy and will be presented in chapter 7.

If you want to learn more about your local format in R, use the following command by typing it in the prompt and pressing enter:

```
Sys.localeconv()
```

```
R>     decimal_point      thousands_sep          grouping
R>              "."                 ""                ""
R>   int_curr_symbol    currency_symbol mon_decimal_point
R>           "BRL "               "R$"               ","
R> mon_thousands_sep       mon_grouping      positive_sign
R>              "."          "\003\003"                ""
R>     negative_sign     int_frac_digits        frac_digits
R>              "-"                "2"               "2"
R>     p_cs_precedes      p_sep_by_space      n_cs_precedes
R>              "1"                "1"               "1"
R>     n_sep_by_space        p_sign_posn        n_sign_posn
R>              "1"                "1"               "1"
```

The output of **Sys.localeconv()** shows how R interprets decimal points and the thousands separator, among other things. As you can see from the previous output, this book was compiled using the Brazilian notation for the currency symbol, but uses the dot point for decimals.

> Be careful when modifying the format that R interprets symbols. As a rule of thumb, if you need to use a specific format, do it separately within the context of the code. Avoid permanent changes as you never know where such formats are being used. That way, you can avoid unpleasant surprises in the future.

## 2.4 Types of Files in R

Like any other programming platform, R has a ecosystem of file extensions, where each has a different purpose. In the vast majority of cases, however, the work will focus mostly on a couple of types. Next, I describe the various

file extensions you'll find in a day to day basis. The items in the list are ordered by importance. Note that I omitted graphic files such as *.png*, *.jpg*, *.gif* and data storage/spreadsheet files (*.csv*, *.xlsx*, ..) among others, as they are not exclusive to R.

**Files with extension *.R***: text files containing R code. These are the files which you will spend most of your time. They contain the sequence of commands that configures the main script and computational routines of the data research. Examples: *Script-stock-research.R*, *R-fcts.R*.

**Files with extension *.RData* or *.rds***: files that store data in the native format. These files are used to save/write objects created in different sessions into your hard drive. For example, you can use a *.rds* file to save a table after processing and cleaning up the raw database. By *freezing* the data into a local file, we can later load it for subsequent analysis. Examples: *cleaned-inflation-data.rds*, *model-results.RData*.

**Files with extension *.Rmd* and *.quarto***: files used for editing dynamic documents in the *RMarkdown* and *Quarto* format. Using these files allows the creation of documents where text and code output are integrated into the same document. While *RMarkdown* is mostly related to R, the *quarto* format allows for a more flexible integration of text and code for other programming languages such as Python. In chapter 12 we have a dedicated section for RMarkdown, which will explore this functionality in detail. Example: *investment-report.Rmd*.

**Files with extension *.Rproj***: contain files for editing projects in RStudio, such as a new R package, a *shiny* application or a book. While you can use the functionalities of RStudio projects to write R scripts, it is not a hard requirement. For those interested in learning more about this functionality, I suggest the Posit manual[3]. Example: *project-retirement.Rproj*.

## 2.5   Explaining the RStudio Screen

After installing the R and RStudio, open RStudio by double-clicking its icon. **Be aware that R also has its own interface and this often causes confusion**. In Windows, you can search for RStudio link using the *Start* button and typing *Rstudio*.

After opening RStudio, the resulting window should look like Figure 2.1.

Note that RStudio automatically detected the installation of R and initialized a session on the left side of the interface.

---

[3]https://support.posit.co/hc/en-us/articles/200526207-Using-Projects

Figure 2.1: The RStudio screen

As a first exercise, click *File*, *New File*, and *R Script*. A text editor should appear on the left side of the screen. It is there that we will spend most of our time developing code. Commands are executed sequentially, from top to bottom. A side note, all *.R* files created in RStudio are just text files and can be edited anywhere. As an exercise, use Windows's Notepad to open an R file. You'll see that the raw code is the same, but without the syntax highlighting.



Figure 2.2: Explaining the RStudio screen

When using RStudio, a common suggestion is to change the color scheme to a **dark mode** setting. It is not just an aesthetic issue, but also a strategy for preventing eye strain. Since you will be spending a lot of time in front of the computer, it is smart to change the colors of the interface to relieve your eyes of the constant brightness of the screen. That way, you'll be able to work longer, without straining your vision. You can configure the color scheme of RStudio by going to the option *Tools*, *Global Options* and then *Appearance*. A dark color scheme that I personally like and suggest is *Ambience*.

After the previous steps in RStudio, the resulting screen should look like the image in Figure 2.2. The main items/panels are:

**Script Editor:** located on the left side and above the screen. This panel is used to write scripts and functions, mostly on files with the *.R* extension;

**R prompt:** on the left side and below the script editor. It displays the *prompt*, which can also be used to give commands to R. The main purpose of the *prompt* is to test code and display the results of the commands entered in the script editor;

**Environment:** located on the top-right of the screen. Shows all objects, including variables and functions, currently available to the user. Also note a *History* panel, which shows the history of commands previously executed by the user;

**Panel Packages:** shows the packages installed and loaded by R. Here you have four tabs: *Files*, to load and view system files; *Plots*, to view statistical figures created in R; *Help* to access the help system and *Viewer* to display dynamic and interactive results, such as a web page.

As an introductory exercise, let's initialize two objects in R. Inside the prompt (lower left side), insert these commands and press *enter* at the end of each. The <- symbol is nothing more than the result of joining < (less than) with the - (minus sign). The ' symbol represents a single quotation mark and, in most computer keyboards, it is found under the escape (*esc*) key (top left).

```
# set x
x <- 1

# set y
y <- 'My humble text'
```

If done correctly, notice that two objects are available in the *environment* panel, one called x with a value of 1, and another called y with the text content "My humble text". Also noticed how we used specific symbols to define objects x and y. The use of double quotes (" ") or single quotes (' ') defines objects of the class `character`. Numbers are defined by the value itself. As will be discussed later, understanding R object classes are important as each has a different behavior within the R code. After executing the previous commands, notice that the *history tab* has been updated.

Now, lets show the values of x on the screen. To do this, type the following command:

```
# print contents of x
print(x)
```

```
R> [1] 1
```

The **print()** function is one of the main commands for displaying values in the *prompt* of R. The text displayed as [1] indicates the index of the first line number. To verify this, enter the following command, which will show a lengthy sequence of numbers on the screen:

```
# print a sequence
print(50:100)
```

```
R>  [1]  50  51  52  53  54  55  56  57  58  59  60  61  62  63
R> [15]  64  65  66  67  68  69  70  71  72  73  74  75  76  77
R> [29]  78  79  80  81  82  83  84  85  86  87  88  89  90  91
R> [43]  92  93  94  95  96  97  98  99 100
```

Here, we use the colon symbol in `50:100` to create a sequence starting at 50 and ending at 100. Note that, on the left side of each line, we have the values [1], [13], and [25]. These represent the index of the first element presented in the line. For example, the fifteenth element of `50:100` is 64.

## 2.6  R Packages

One of the greatest benefits of using R is its package collection. A package is nothing more than a group of procedures aimed at solving a particular computational problem. R has at its core a collaborative philosophy. Users provide their codes for others to use. And, most importantly, **all packages are free**. For example, consider the scenario where a user is interested in accessing data about historical inflation in the USA. He can install and use an R module that is specifically designed for importing data from central banks and research agencies.

Every function in R belongs to a package.  When R initializes, packages `stats`, `graphics`, `grDevices`, `utils`, `datasets`, `methods` and `base` are loaded by default. This is way why we can use some functions in R without the explicit call to a library. For example, function `print` is from the `base` package and is available whenever you start R.

R packages can be accessed and installed from different sources. The main being **CRAN** (*The Comprehensive R Archive network*), and **Github**. It's worth knowing that the quantity and diversity of R packages increases every day. **CRAN is the official repository of R and it is built and maintained by the community**. One of the reasons for the success of CRAN is the quality of code. While anyone can send a package, there is an evaluation process to ensure that certain strict rules about code format and safety are respected. For those interested in creating and distributing packages in CRAN, a clear roadmap on is available on the R packages site[4]. The official (and complete) rules are available on the CRAN website[5].

> So far, I have eight package published in CRAN. In my experience, sending and publishing a package on CRAN can demand a significant amount of work, especially in the first submission. After that, it becomes a lot easier. Don't be angry if your package is rejected. My own packages were rejected several times before entering CRAN. Listen to what the maintainers tell you and try fixing all problems before resubmitting. If you're having issues that you cannot solve or find a solution on the Internet, look for help in the R-packages mailing list[6]. You'll probably be surprised at how accessible and helpful the R community can be.

The complete list of packages available on CRAN, along with a brief description of each, can be accessed at the packages section of the R site[7]. A practical way to check if there is a package that does a specific procedure is to load the previous page and search in your *browser* for a keyword of interest (e.g. "SEC data"). If there is a package that does what you want, it is very likely that the keyword is used in the description field.

Another important source for finding packages is the CRAN Task Views[8]. There you can find the collection of noteworthy packages for a given area of expertise. See the *Task Views* screen in Figure 2.3.

---

[4]https://r-pkgs.org/

[5]https://cran.r-project.org/web/packages/policies.html

[7]https://cran.r-project.org/web/packages/available_packages_by_date.html

[8]https://cran.r-project.org/web/views/

CRAN Task Views

| | |
|---|---|
| Bayesian | Bayesian Inference |
| ChemPhys | Chemometrics and Computational Physics |
| ClinicalTrials | Clinical Trial Design, Monitoring, and Analysis |
| Cluster | Cluster Analysis & Finite Mixture Models |
| DifferentialEquations | Differential Equations |
| Distributions | Probability Distributions |
| Econometrics | Econometrics |
| Environmetrics | Analysis of Ecological and Environmental Data |
| ExperimentalDesign | Design of Experiments (DoE) & Analysis of Experimental Data |
| ExtremeValue | Extreme Value Analysis |
| Finance | Empirical Finance |
| Genetics | Statistical Genetics |
| Graphics | Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization |
| HighPerformanceComputing | High-Performance and Parallel Computing with R |
| MachineLearning | Machine Learning & Statistical Learning |
| MedicalImaging | Medical Image Analysis |
| MetaAnalysis | Meta-Analysis |
| Multivariate | Multivariate Statistics |
| NaturalLanguageProcessing | Natural Language Processing |
| NumericalMathematics | Numerical Mathematics |
| OfficialStatistics | Official Statistics & Survey Methodology |
| Optimization | Optimization and Mathematical Programming |
| Pharmacokinetics | Analysis of Pharmacokinetic Data |
| Phylogenetics | Phylogenetics, Especially Comparative Methods |
| Psychometrics | Psychometric Models and Methods |
| ReproducibleResearch | Reproducible Research |
| Robust | Robust Statistical Methods |

Figure 2.3: Task View screen

A popular alternative to CRAN is Github[9]. Unlike the former, Github imposes no restrictions on the submitted code and, because of this flexibility and its version control system, it is a popular choice by developers. In practice, it is very common for developers to maintain a development version of a module on Github and the official version in CRAN. When the development version reaches a certain stage of maturity, it is then sent to CRAN.

The most interesting part about R packages is that they can be installed directly from the prompt using the internet. To find out the current amount of packages on CRAN, type and execute the following commands in the prompt:

```r
# get a matrix with available packages
df_cran_pkgs <- available.packages()

# find the number of packages
n_cran_packages <- nrow(df_cran_pkgs)

# print it
print(n_cran_packages)
```

```
R> [1] 20186
```

Currently, 2023-12-13 10:32:04.071736, there are 20186 packages available on the CRAN servers, a very impressive mark for the community of developers as a whole.

---

[9]https://github.com/

You can also check the amount of **locally installed packages** in R with
the **installed.packages()** command:

```
# find number of packages currently installed
n_local_packages <- nrow(installed.packages())

# print it
print(n_local_packages)
```

```
R> [1] 546
```

In this case, the computer in which the book was compiled has 546 packages
currently installed. Notice that, even as an experienced R programmer, I'm
only using a small fraction of all packages available in CRAN!

## 2.6.1   Installing Packages from CRAN

Use command **install.packages()** to install a package locally. You only
need to do it once for each new package. As an example, we will install a
package called **{readr}** (Wickham et al., 2023c), that will be used in future
chapters. Note that we defined the package name in the installation as if it
were text with the use of quotation marks (" ").

```
# install package readr
install.packages("readr")
```

That's it! After executing this simple command, package **{readr}** (Wick-
ham et al., 2023c) will be installed and the functions related to the package
will be ready for use once the package is loaded in a script. If the installed
package is dependent on another package, R detects this dependency and
automatically installs the missing packages. Thus, all the requirements for
using the installed package are satisfied and everything will work perfectly.
Be aware, however, that some modules can require external software in the
level of the operating system. These cases are usually announced in the de-
scription of the package and an error informs that a requirement is missing.
External dependencies for R packages are not common, but they do happen.

## 2.6.2   Installing Packages from Github

To install a package hosted in Github, you must first install the **{devtools}**
(Wickham et al., 2022) package, available on CRAN:

```
# install devtools
install.packages('devtools')
```

After that, use function **devtools**::**install_github()** to install a package directly from Github. In the following example, we will install the development version of package **{dplyr}** (Wickham et al., 2023b):

```r
# install ggplot2 from github
devtools::install_github("hadley/dplyr")
```

Note that the username of the developer is included in the input string. In this case, the *hadley* name belongs to the developer of **{dplyr}** (Wickham et al., 2023b), Hadley Wickham. Throughout the book, you will notice that this name appears several times. Hadley is a prolific and competent developer of several popular R packages and currently works for RStudio.

> Be aware that **github packages are not moderated**. Anyone can send code there and the content is not independently checked. Never install github packages without some confidence of the author's work. Although unlikely - it never happened to me, for example - it is possible that they have malicious code.

### 2.6.3  Loading Packages

Within a script, use function **library()** to load a package, as in the following example.

```r
# load package readr
library(readr)
```

After running this command, all functions of the package will be available in the **current** R session. Whenever you close RStudio or start a new session, you'll lose all loaded packages. This is the reason why packages are usually loaded in the top of the script: starting from a clean memory, required packages are sequentially loaded before the actual R code is executed.

If the package you want to use is not available, R will throw an error message. See an example next, where we try to load a non-existing package called `unicorn`.

```r
library(unicorn)
```

```
R> Error in library(unicorn): There is no package called "unicorn"
```

Remember this error message. It will appear every time a package is not found. If you got the same message when running code from this book, you need to check what are the required packages of the example and install them using **install.packages()** .

Alternatively, if you want to use a specific function and do not want to load all functions from the same package, you can do it with the use of double colons (::), as in the following example.

```r
# example of using a function without loading package
fortunes::fortune(10)
```

```
R>
R> Overall, SAS is about 11 years behind R and S-Plus in
R> statistical capabilities (last year it was about 10 years
R> behind) in my estimation.
R>    -- Frank Harrell (SAS User, 1969-1991)
R>       R-help (September 2003)
```

Here, we use function **fortunes**::**fortune()** from the package **{fortunes}** (**?**), which shows on screen a potentially funny phrase chosen from the R mailing list. For our example, we selected message number 10. One interesting use of package **{fortunes}** (**?**) is to display a random joke every time R starts and, perhaps, lighten up your day. As mentioned before, R is fully customizable. You can find many tutorials on how to achieve this effect by searching on the web for "customizing R startup".

Another way of loading a package is by using the **require()** function. A call to **require()** has a different behavior than a call to **library()** . Whenever you try to load an uninstalled package with the **library()** function, it returns an error. This means that the script stops and no further code are evaluated. As for **require()** , if a package is not found, it returns an object with value `FALSE` and the rest of the code is evaluated. So, in order to avoid code being executed without its explicit dependencies, it is best advised to always use **library()** for loading packages in R scripts.

The use of **require()** is left for loading up packages inside of functions. If you create a custom function that requires procedures from a particular package, you must load the package within the scope of the function. For example, see the following code, where we create a new function called `fct_example` that depends on the package **{quantmod}** (Ryan and Ulrich, 2023):

```r
fct_example <- function(x){

  require(quantmod)

    df <- getSymbols(x, auto.assign = F)
    return(df)
}
```

In this case, the first time that `fct_example` is called, it loads up the package **{quantmod}** (Ryan and Ulrich, 2023) and all of its functions. Using **require()** inside a function is a good programming policy because the function becomes self-contained, making it easier to use it in the future. This was the first time where the complete definition of a user-created function in R is presented. Do not worry about it for now. We will explain it further in chapter 8.

> Be aware that loading a package can cause a **conflict of functions**. For example, there is a function called `filter` in the `dplyr` package and also in the `stats` package. If we load both packages and call the `filter` function within the scope of the code, which one will R use?
>
> Well, the **preference is always for the last loaded package**. This is a type of problem that can be very confusing. Fortunately, note that R itself tests for conflicts when loading a package. Try it out: start a new R session and load the `dplyr` package. You'll see that a message indicates that there are two conflicts with the `stats` package – functions `filter` and `lag` – and four with the `base` package.
>
> A simple strategy to avoid bugs due to conflict of function is to call a function using the actual package name. For example, if I'm calling `lag` from `dplyr`, I can write the call as `dplyr::lag`. As you can see, the package name is explicit, avoiding any possible conflict.

### 2.6.4 Upgrading Packages

Over time, it is natural that packages available on CRAN are upgraded to accommodate new features and fix bugs. Thus, it is recommended that users update their installed packages to a new version. In R, a simple and direct way of upgrading packages is to click the button *Update* located in the package panel, lower right corner of RStudio, as shown in Figure 2.4.

The user can also update packages using commands. Simply type command **update.packages()** and hit *enter*, as shown below.

```r
# update all installed packages
update.packages()
```

The command **update.packages()** compares the version of the installed packages with the versions available in CRAN. If it finds any difference, the new version is downloaded and installed. After running the command, all
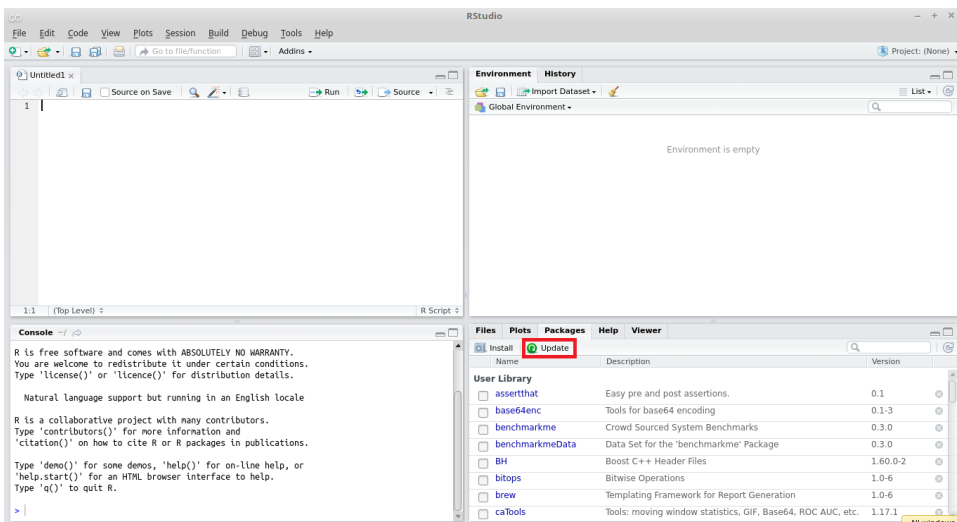
Figure 2.4: Updating R packages

packages will be synchronized with the versions available in CRAN.

> Package versioning is an extremely important topic for keeping your code reproducible. Although it is uncommon to happen, a new package version can change the results of your analysis. I have a particularly unpleasant experience when a scientific article returned from a journal review and, due to the update of one of the R packages I used, I was unable to reproduce the results presented in the article. In the end everything went well, but the trauma remains.
>
> One solution to this problem is to freeze the package versions for each project using RStudio's `renv` tool. In summary, `renv` makes local copies of the packages used in the project, which have preference over system packages. Thus, if a package is updated in the system, but not in the project, the R code will continue to use the older version and the R code will always run under the same conditions.

## 2.7   Running Scripts from RStudio

Now, let's copy some code into the editor's screen (upper left side). The result should look like Figure 2.5.

```
# a complete script
x <- 1
y <- "my humble text"
print(x)
```

```
R> [1] 1
```

```
print(y)
```

```
R> [1] "my humble text"
```
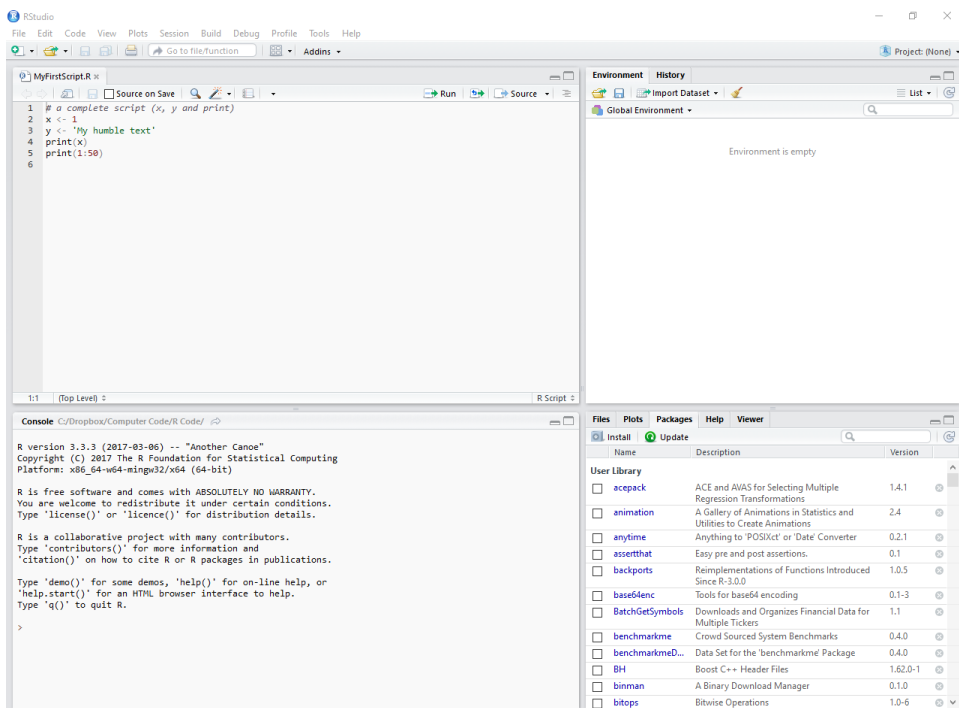


Figure 2.5: Example of an R script

After pasting all the commands in the editor, save the *.R* file to a personal folder where you have read and write permissions. In Windows, one possibility is to save it in the `Documents` folder with a name like `'my_first_script.R'`. This saved file, which at the moment does nothing special, records the steps of a simple algorithm that creates several objects and shows their content.

## 2.8   Using the help files

There is no shame in seeking help. Advanced R users often seek instructions on specific tasks, whether it is to better understand the execution details of some functions or simply to study a new procedure. The use of the R help system is part of the work and you should master it as soon as possible.

Within RStudio, you can get help by using the *help* panel in RStudio or directly from the *prompt*. Simply enter the question mark next to the object on which you want help, as in `?mean`. In this case, object `means` is a function and the use of the **help()** command will open a panel on the right side of RStudio. Another way of opening the help page of a function is the select the name of the function and press F1 in the keyboard.

In R, the help screen of a function is the same as shown in Figure 2.6. It presents a general description of the function, explains its input arguments and the format of the output. The help screen follows with references and suggestions for other related functions. More importantly, examples of usage are given last and can be copied to the prompt or script in order to accelerate the learning process.

mean {base}                                                       R Documentation

## Arithmetic Mean

**Description**

Generic function for the (trimmed) arithmetic mean.

**Usage**

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

**Arguments**

x      An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

trim   the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.

Figure 2.6: Help screen for function mean

If we are looking for help for a given text and not a function name, we can use double question marks as in `??"standard deviation"`. This operation will search for the occurrence of the term in all packages of R and it is very useful to learn how to perform a particular task. In this case, we looked for

the available functions to calculate the standard deviation of a vector.

As a suggestion, the easiest and most direct way to learn a new function is by trying out the examples in the manual. This way, you can see which types of inputs the function expects and what type of output it provides back. Once you have it working in your code, read the help screen to understand if it does exactly what you expected and what are the options for its use. If the function performs the desired procedure, you can copy and paste the example for your own *script*, adjusting where necessary.

Another very important source of help is the Internet itself. Sites like stackoverflow[10] and specific *mailing lists* and blogs, whose content is also on the Internet, are a valuable source of information. If you find a problem that could not be solved by reading the standard help files, the next logical step is to seek a solution using your error message or the description of the problem in search engines. In many cases, your problem, **no matter how specific it is, has already occurred and has been solved by other users**. In fact, it is more surprising **not** to find the solution for a programming problem on the internet, than the other way around.

> Whenever you ask for help on the internet, always try to 1) describe your problem clearly and 2) add a reproducible code of your problem. Thus, the reader can easily verify what is happening by running the example on his computer, and solving the problem.

## 2.8.1 RStudio shortcuts

In RStudio, there are some predefined and time-saving shortcuts for running code from the editor. To execute an entire script, simply press **control + shift + s**. This is the *source* command. With RStudio open, I suggest testing this key combination and checking how the code saved in a *.R* file is executed. The output of the script is shown in the prompt of R. The result in RStudio should look like Figure 2.7.

Another way of executing code is with the shortcut **control + enter**, which will only execute the line where the cursor is located. This shortcut is very useful in developing scripts because it allows each line of the code to be tested. As an example of usage, point the cursor to the `print(x)` line and press *control + enter*. As you will notice, only the line `print(x)` was executed and the cursor moves to the next line. Therefore, before running the whole script, you can test it line by line and check for possible errors.

---

[10]http://stackoverflow.com/
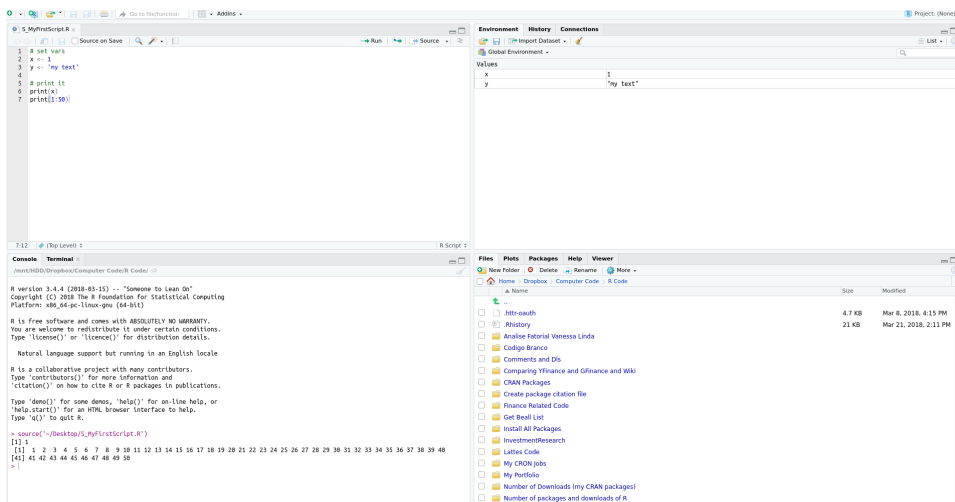
Figure 2.7: Example of a R script after execution

Next, I highlight these and other RStudio shortcuts, which are also very useful.

**control + shift + s** executes (source) the current RStudio file;

**control + shift + enter** executes the current file with echo, showing the commands on the prompt;

**control + enter** executes the selected line, showing on-screen commands;

**control + shift + b** executes the codes from the beginning of the file to the cursor's location;

**control + shift + e** executes the codes of the lines where the cursor is until the end of the file.

I suggest you try to create a healthy habit by using these shortcuts from day one. Those who like to use the *mouse*, an alternate way to execute code is to click the *source* button in the upper-right corner of the code editor.

If you want to run code in a *.R* file within another *.R* file, you can use the **source()** command. For example, imagine that you have a main R script with your data analysis and another two scripts that performs some support operation such as importing data to R. These operations have been dismembered as a way of organizing the code.

To run the support *scripts*, just call it with function `source` in the main script, as in the following code:

```r
# execute import script
source('01-import-data.R')

# execute analysis
source('02-build-tables')
```
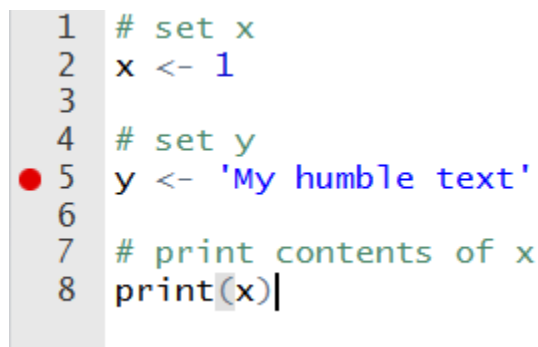
Here, all code in `01-import-data.R` and `02-build-tables.R` will be executed sequentially. This equals manually opening each file and hitting *control + shift + s*.

## 2.9 Testing and Debugging Code

Developing code follows an observable cycle. At first, you will write a command line on a script, try it using *control + enter*, and check the result on the prompt or the content of objects. A new line of code is written once the previous line worked as expected. A moving cycle is clear, writing code is followed by line execution, followed by result checking, modify and repeat if necessary. This is a normal and expected process. You need to make sure that every line of code is correctly specified before moving to the next one. Whenever the code asks for something that is not expected (or possible), an error occurs.

When trying to find an error in a preexisting script, R offers debugging tools for controlling and assessing its execution. This is especially useful when you have a long and complicated script. The simplest and easiest tool that R and RStudio offer is **code breakpoint**. In RStudio, you can click on the left side of the script editor and a red circle will appear, as in Figure 2.8.

```r
1  # set x
2  x <- 1
3
4  # set y
5  y <- 'My humble text'
6
7  # print contents of x
8  print(x)
```

Figure 2.8: Example of breakpoint in an R script

This red circle indicates a flag that will force the code to stop at that line. You can use it to test existing code and check its objects at a certain part

of the execution. Pausing the code at a certain point might seem strange for a starting programmer but, for large scripts, with many functions and complex code, it is a necessity. When the execution hits the breakpoint, the prompt will change to `Browse[1]>` and you'll be able to try new code and verify the content of all current objects. From the Console, you have the option to continue the execution to the next breakpoint or stop it by pressing *shift+f8*. The same result can be achieved using a function called `browser`. Have a look:

```r
# set x
x <- 1

# set y
y <- 'My humble text'

browser()

# print contents of x
print(x)
```

The practical result is the same as using RStudio's red circle, but it gives you more control for the case of several commands in the same line.

## 2.10   Creating Simple Objects

One of the most basic and most used commands in R is the creation of objects. As shown in previous sections, you can define an object using the `<-` command, which is verbally translated to *assign*. For example, consider the following code:

```r
# set x
x <- 123

# set my_x, my_y and my_z in one line
my_x <- 1; my_y <- 2; my_z <- 3
```

We can read this code as *the value 123 is assigned to x*. The direction of the arrow defines where the value is stored. For example, using `123 ->` `x` also works, although this is not recommended as the code becomes less readable. Moreover, notice that you can create objects within the same line by separating the commands using a semi-colon.

Using an arrow symbol `<-` for object definition is a simple way to identify R code. The reason for this choice was that, at the time of conception of the *S* language, keyboards had a specific key that directly defined the arrow symbol. This means that the programmer only had to hit one key in the keyboard to create new variables. Modern keyboards, however, are different. If you find it troublesome to type this symbol, you can use a shortcut as well. In *Windows*, the shortcut for the symbol `<-` is *alt* plus "-".

Most programming languages uses a equality symbol (=) to define objects and, often, this creates confusion. When using R, you can also define objects with =, as in `x = 123`, however, no one should ever recommend it. The equality symbol has a special use within an R code as it defines function arguments, as in `my_l <- fct(arg1 = 1, arg2 = 3)`. For now, just remember to use `<-` for defining objects. We will learn more about functions and using the equality symbol in a future chapter.

The name of the object is important in R. With the exception of very specific cases, you can name objects as you please. This freedom, however, can work against you. It is desirable to set short object names that make sense to the content of the script and which are simple to understand. This facilitates the understanding of the code by other users and is part of the suggested set of rules for structuring code. Note that all objects created in this book have nomenclature in English and specific format, where the white space between nouns are replaced by an underscore, as in `my_x <- 1` and `name_of_file <- 'my_data_file.csv'`. We will get back at code structure in chapter 13.

R executes code by looking for objects available in the environment, including functions. You also need to be aware that R is case sensitive. That is, object `m` is not the same as `M`. If we try to access an object that does not exist, R will return an error message and halt the execution of the rest of the code. Have a look:

```
print(z)
```

```
R> Error in eval(expr, envir, enclos): object 'z' not found
```

The error occurred because the object `z` does not exist in the current environment. If we create a variable `z` as `z <- 321` and repeat the command `print(z)`, we will not have the same error message.

## 2.11   Creating Vectors

In the previous examples, we created simple objects such as `x <- 1` and `x <- 'ABC'`. While this is sufficient to demonstrate the basic commands in R, in practice, such commands are very limited. A real problem of data analysis will certainly have a larger volume of information.

When we gather many elements of the same class, such as `numeric`, into a single object, the result is an atomic vector. An example would be the representation of a series of daily stock prices as an atomic vector of the class `numeric`. Once you have a vector, you can manipulate it any way you want.

Atomic vectors are created in R using the **c()** command, which comes from the verb *combine*. For example, if we want to combine the values 1, 2 and 3 in a single object, we can do it with the following command:

```
# create numeric atomic vector
x <- c(1, 2, 3)

# print it
print(x)
```

```
R> [1] 1 2 3
```

The **c()** command works the same way for any other class of object, such as *character*:

```
# create character atomic vector
y <- c('text 1', 'text 2', 'text 3', 'text 4')

# print it
print(y)
```

```
R> [1] "text 1" "text 2" "text 3" "text 4"
```

The only restriction when creating vectors is that all elements must have the same class. If we insert data from different classes in a call to **c()** , R will try to mutate all elements into the same class following its own logic. If the conversion of all elements to a single class is not possible, an error message is returned. Note the following example, where numeric values are set in the first and second element of `x` and a character in the last element.

```
# a mixed vector
x <- c(1, 2, '3')
```

```
# print result of forced conversion
print(x)
```

R> [1] "1" "2" "3"

Notice that all values in x are converted to type **character**. The use of **class()** command confirms this result:

```
# print class of x
class(x)
```

R> [1] "character"

## 2.12  Knowing Your Environment and Objects

After using various commands, further development of an R script requires you to understand what objects are available and if their content are correct. You can find this information simply by looking at the *environment* tab in the upper right corner of RStudio. However, there is a command that shows the same information in the prompt. In order to know what objects are currently available in R's memory, you can use the command **ls()** . Note the following example:

```
# set some objects
x <- 1
y <- 2
z <- 3

# print all objects in the environment
print(ls())
```

R> [1] "x" "y" "z"

Objects x, y and z were created and are available in the current working environment. If we had other objects, they would also appear in the output to **ls()** .

To display the content of each object, just enter the names of objects and press **enter** in the *prompt*:

```
# print objects by their name
x
```

R> [1] 1

```
y
```

```
R> [1] 2
z
```

```
R> [1] 3
```

Typing the object name on the screen has the same effect as using the **print()** command. In fact, when executing the sole name of a variable in the prompt or script, R internally passes the object to the **print()** function.

In R, all objects belong to a class. As previously mentioned, to find the class of an object, simply use the **class()** function. In the following example, x is an object of the class `numeric`, y is a text (`character`) object and `fct_example` is a function object.

```
# set objects
x <- 1
y <- 'a'
fct_example <- function(){}

# print their classes
print(class(x))
```

```
R> [1] "numeric"
```

```
print(class(y))
```

```
R> [1] "character"
```

```
print(class(fct_example))
```

```
R> [1] "function"
```

Another way to learn more about an object is to check its textual representation. Every object in R has this property and we can find it with function **str()** :

```
# set vec
x <- 1:10
# print the textual representation of a vector
print(str(x))
```

```
R>  int [1:10] 1 2 3 4 5 6 7 8 9 10
R> NULL
```

We find that object x is a vector of class `int` (integer). Function **str()** is

particularly useful when trying to understand the details of a more complex object, such as a `dataframe` or a `list`, which will be introduced in future chapter.

## 2.13   Finding the Size of Objects

In R, an object size can mean different things but most likely it is defined as the number of individual elements that constitute the object. This information serves not only to assist the programmer in checking possible code errors but also to know the length of iteration procedures such as *loops*, which will be treated in a later chapter of this book.

In R, the size of an object can be checked with the use of four main functions: **length()** , **nrow()** , **ncol()** and **dim()** .

Function **length()** is intended for objects with a single dimension, such as atomic vectors:

```r
# create atomic vector
x <- c(2, 3, 3, 4, 2,1)

# get length of x
n <- length(x)

# display message
message('The length of x is ', n)
```

```
R> The length of x is 6
```

For objects with more than one dimension, such as a matrix, use functions **nrow()** , **ncol()** and **dim()** (dimension) to find the number of rows (first dimension) and the number of columns (second dimension). See the difference in usage below.

```r
# create a matrix
M <- matrix(1:20, nrow = 4, ncol = 5)

# print matrix
print(M)
```

```
R>      [,1] [,2] [,3] [,4] [,5]
R> [1,]    1    5    9   13   17
R> [2,]    2    6   10   14   18
R> [3,]    3    7   11   15   19
R> [4,]    4    8   12   16   20
```

```r
# calculate size in different ways
my_nrow <- nrow(M)
my_ncol <- ncol(M)
my_n_elements <- length(M)

# display messages
message('The number of lines in M is ', my_nrow)
```

```
R> The number of lines in M is 4
```

```r
message('The number of columns in M is ', my_ncol)
```

```
R> The number of columns in M is 5
```

```r
message('The number of elements in M is ', my_n_elements)
```

```
R> The number of elements in M is 20
```

The **dim()** function shows the dimension of the object, resulting in a numeric
vector as output. This function should be used when the object has more
than two dimensions. In practice, however, such cases are rare as most data-
related problems can be solved with a bi-dimensional representation. An
example is given next:

```r
# get dimension of M
my_dim <- dim(M)

# print it
print(my_dim)
```

```
R> [1] 4 5
```

In the case of objects with more than two dimensions, we can use the **array()**
function to create the object and **dim()** to find its size. Have a look at the
next example:

```r
# create an array with three dimensions
my_array <- array(1:9, dim = c(3, 3, 3))

# print it
print(my_array)
```

```
R> , , 1
R>
R>      [,1] [,2] [,3]
R> [1,]    1    4    7
```

```
R> [2,]    2    5    8
R> [3,]    3    6    9
R>
R> , , 2
R>
R>      [,1] [,2] [,3]
R> [1,]    1    4    7
R> [2,]    2    5    8
R> [3,]    3    6    9
R>
R> , , 3
R>
R>      [,1] [,2] [,3]
R> [1,]    1    4    7
R> [2,]    2    5    8
R> [3,]    3    6    9
```

```
# display its dimensions
print(dim(my_array))
```

```
R> [1] 3 3 3
```

## 2.14 Selecting Elements from an Atomic Vector

After creating an atomic vector, you can access and select portions of it. For example, suppose that a month ago you invested 1.000 USD in Apple shares. You download stock price data for previous month and want to see how much your portfolio is currently worth. In that case, you are interested only in latest price of the stocks. All price values from other dates are not relevant to our analysis and therefore could be safely ignored.

The selection of *pieces* of an atomic vector is called **indexing** and it is accomplished with the use of square brackets ([ ]). Consider the following example:

```
# set x
my_x <- c(1, 5, 4, 3, 2, 7, 3.5, 4.3)
```

If we wanted only the third element of `my_x`, we use the bracket operator as follows:

```r
# get the third element of x
elem_x <- my_x[3]

# print it
print(elem_x)
```

```
R> [1] 4
```

Indexing also works using vectors containing the desired locations. If we are only interested in the last and penultimate values of `my_x`, we use the following code:

```r
# set vector with indices
my_idx <-  (length(my_x)-1):length(my_x)

# get last and penultimate value of my_x
piece_x_1 <- my_x[my_idx]

# print it
print(piece_x_1)
```

```
R> [1] 3.5 4.3
```

A cautionary note: **a unique property of the R language is that if a non-existing element of an object is accessed, the program returns the value `NA` (*not available*)**. See the next example code, where we attempt to obtain the fourth value of a vector with only three components.

```r
# set object
my_vec <- c(1, 2, 3)

# print non-existing fourth element
print(my_vec[4])
```

```
R> [1] NA
```

This is important because `NA` elements are contagious. That is, anything that interacts with `NA` will also become `NA`. The lack of treatment of these errors can lead to problems that are difficult to identify. In other programming languages, attempting to access non-existing elements generally returns an error and cancels the execution of the rest of the code.

> ❗ In most cases, the occurrence of `NA` (*Not Available*) values suggests that a problem exists within the code. Always remember that `NA` values indicates lack of data and are contagious: anything that interacts with a `NA` value will turn into another `NA`. **You should become suspicious about your code quality every time that `NA` values are found unexpectedly**.

The use of indices is very useful when you are looking for items of a vector that satisfy some condition. For example, if we wanted to find out all values in `my_x` that are greater than 3, we could use the following command:

```
# find all values in my_x that is greater than 3
piece_x_2 <- my_x[my_x>3]

# print it
print(piece_x_2)
```

```
R> [1] 5.0 4.0 7.0 3.5 4.3
```

It is also possible to index elements by more than one condition using the logical operators `&` and `|` (*or*). For example, if we wanted the values of `my_x` greater than 2 **and** lower than 4, we could use the following command:

```
# find all values of my_x that are greater than 2 and lower then 4
piece_x_3 <- my_x[ (my_x > 2) & (my_x < 4) ]
print(piece_x_3)
```

```
R> [1] 3.0 3.5
```

Likewise, if we wanted all items that are lower than 3 **or** greater than 6, we use:

```
# find all values of my_x that are lower than 3 or higher than 6
piece_x_4 <- my_x[ (my_x < 3) | (my_x > 6) ]

# print it
print(piece_x_4)
```

```
R> [1] 1 2 7
```

Moreover, logic indexing also works with the interaction of different objects. That is, we can use a logical condition in one object to select items from another:

```
# set my_x and my.y
my_x <- c(1, 4, 6, 8, 12)
```

```
my_y <- c(-2, -3, 4, 10, 14)

# find all elements of my_x where my.y is higher than 0
my_piece_x <- my_x[my_y > 0 ]

# print it
print(my_piece_x)
```

```
R> [1]  6  8 12
```

Looking more closely at the indexing process, notice that we are creating a
variable of the `logical` type. This object takes only two values: `TRUE` and
`FALSE`. Have a look in the code presented next, where we create a `logical`
object, print it and present its class.

```
# create a logical object
my_logical <- my_y > 0

# print it
print(my_logical)
```

```
R> [1] FALSE FALSE  TRUE   TRUE   TRUE
```

```
# find its class
class(my_logical)
```

```
R> [1] "logical"
```

Logical objects are very useful whenever we are testing a particular condition
on a data set. We will learn more about this and other basic classes in chapter
7.

## 2.15   Removing Objects from the Memory

After creating several variables, the R environment can become full of used
and disposable content. In this case, it is desirable to clear the memory to
erase objects that are no longer needed. Generally, this is accomplished at
the beginning of a script, so that every time the script runs, the memory
will be cleared before any calculation. In addition to cleaning the computer's
memory, it also helps to avoid possible errors in the code.

For example, given an object x, we can delete it from memory with the
command **rm()** , as shown next:

```
# set x
x <- 1

# remove x
rm('x')
```

After executing the command `rm('x')`, the value of `x` is no longer available in the R session. In practical situations, however, it is desirable to clean up all the memory used by all objects created in R. We can achieve this goal with the following code:

```
rm(list = ls())
```

The term `list` in `rm(list = ls())` is a function argument of **rm()** that defines which objects will be deleted. The **ls()** command shows all the currently available objects. Therefore, by chaining together both commands, we erase all current objects available in the environment. As mentioned before, it is a good programming policy to clear the memory before running the script. However, you should only wipe out all of R's memory if you have already saved the results of interest or if you can replicate them.

> Clearing memory in *scripts* is a controversial topic. Some authors argue that it is better not to clear the memory as this can erase important results. My opinion is that it is important to clear the memory at the top of the script, as long as all results are reproducible. When you start a code in a clean state – no variables or functions – it becomes easier to understand and solve possible bugs.

## 2.16 Displaying and Setting the Working Directory

Like other programming platforms, **R always executes code in a working directory**. If no directory is set, a default value is used when R starts up. It is based on the working directory that R searches for files to load data or other R scripts. It is in this directory that R saves any output if we do not explicitly define a path on the computer. This output can be a graphic file, text or a spreadsheet. That said ,**it is good policy to change the working directory to the same place where the *script* is located**.

The simplest way of checking the current working directory is looking at RStudio's prompt panel. At the top, in a small font and just below the word

"Console", you'll see the working path. Using code, we can check the current working directory with function **getwd()** :

```
# get current dir
my_dir <- getwd()

# display it
print(my_dir)
```

```
R> C:/my-books/afedR-ed4/book-content
```

The result of the previous code shows the folder in which this book was written and compiled.

The change of the working directory is performed with the **setwd()** command. For example, if we wanted to change our working directory to *C:/My Research/*, simply type in the *prompt*:

```
# set where to change directory
my_d <- 'C:/My Research/'

# change it
setwd(my_d)
```

After changing the directory, importing and saving files in the *C:/My Research/* folder will be a lot easier.

Additionally, the easiest and most straightforward way to ensure that the working directory is the same as the R script is using **RStudio projects**. To do so, click in "File" and "New Project". Doing so will create a .Rproj file in the chosen directory of the project. The trick here is to create the R scripts in the root folder of the project. Every time you open the project, it will automatically change the working directory to where the .Rproj file is located.

Once you are working on the same path as the script, using relative paths is preferable. For example, if you are working in a folder that contains a subdirectory called `data`, you can enter this sub-folder with the code:

```
# change to subfolder
setwd('data')
```

Another possibility is to go to a previous level of the directory using `..`, as in:

```
# change to the previous level
setwd('..')
```

So, if you are working in directory `C:/My Research/` and execute the command `setwd('..')`, the current folder becomes `C:/`, which is one level below `C:/My Research/`.

## 2.17 Canceling Code Execution

Whenever R is running some code, a visual cue in the shape of a small red circle with the word *stop* in the right corner of the *prompt* will appear. This button is not only an indicator that R is busy running code but also a shortcut for canceling its execution. Another way to cancel an execution is to point the mouse to the *prompt* and press the *escape* (*esc*) button from the keyboard.

To try it out, run the next chunk of code in RStudio and cancel its execution using *esc*.

```
for (i in 1:100) {
  message('\nRunning code (please make it stop by hitting esc!)')
  Sys.sleep(1)
}
```

In the previous code, we used a `for` loop and function `Sys.sleep` to display the message `'\nRunning code (please make it stop by hitting ESC!)'` one hundred times, every second. For now, do not worry about the code and functions used in the example. We will discuss the use of loops in chapter 8.

> **!** Another very useful trick for defining working directories in R is to use the `~` symbol. The tilda defines the "Documents" folder in *Windows*, which is unique for each user. For Linux and Mac users, the tilda defines the "home" folder (e.g. "/home/USERNAME"). Therefore, by running `setwd('~')`, you will direct R to a personal folder in any operating system.

## 2.18 Code Comments

In R, comments are set using the hashtag symbol `#`. Any text after this symbol will not be processed. This gives you the freedom to write whatever you want within the script. An example:

```
# this is a comment (R will not parse it)
# this is another comment (R will again not parse it)
```

```r
x <- 'abc' # this is an inline comment
```

Comments are an effective way to communicate any important information
that cannot be directly inferred from the code. In general, you should avoid
using comments that are too obvious or too generic:

```r
# read CSV file
df <- read.csv('data/data_file.csv')
```

As you can see, it is quite obvious from the line `df <- read.csv('..')` that
the code is reading a .csv file. The name of the function already states that.
So, the comment was not a good one as it did not add any new information
to the user. A better approach at commenting would be to set the author,
description of the script and better explain the origin and last update of the
data file. Have a look:

```r
# Script for reproducing the results of JOHN (2019)
# Author: Mr data analyst (dontspamme@emailprovider.com)
# Last script update: 2020-01-10
#
# File downloaded from www.site.com/data-files/data_file.csv
# The description of the data goes here
# Last file update: 2020-01-10


df <- read.csv('data/data_file.csv')
```

So, by reading the comments, the user will know the purpose of the script,
who wrote it and the date of the last edit. It also includes the origin of the
data file and the date of the latest update. If the user wants to update the
data, all he has to do is go to the referred website and download the new
file.

Another productive use of comments is to set sections in the code, such as
in:

```r
# Script for reproducing the results of JOHN (2019)
# Author: Mr data analyst (dontspamme@emailprovider.com)
# Last script update: 2020-01-10
#
# File downloaded from www.site.com/data-files/data_file.csv
# The description of the data goes here
# Last file update: 2020-01-10


# Clean data ------------------------
```

```
# - remove outliers
# - remove unnecessary columns


# Create descriptive tables ----------



# Estimate models --------------------



# Report results --------------------
```

The use of a long line of dashes (-) at each section of the code is intentional. It causes RStudio to identify and bookmark the sections, with a link to them at the bottom of the script editor. Test it yourself, copy and paste the above code into a new R script, save it, and you'll see that the sections appear on a button between the editor and the prompt. Such a shortcut can save plenty of time in lengthy scripts.

> When you start to share code with other people, you'll soon realize that comments are essential and expected. They help transmit information that is not available from the code. This is one way of a discerning beginners from experienced programmers. Contrary to popular belief, it is likely that someone with experience in programming will be very communicative in its comments (sometimes too much!). A note here, throughout the book you'll see that the code comments are, most of the time, a bit obvious. This was intentional as clear and direct messages are important for new users, which is part of the audience of this book.

## 2.19  Using Code Completion with *tab*

A very useful feature of RStudio is *code completion*, an editing tool that facilitates the search of object names, packages, function arguments, and files. Its usage is very simple. After you type any first letter in the keyboard, just press *tab* (left side of the keyboard, above *capslock*) and a number of options will appear. See Figure 2.9 where, after entering the *f* letter and pressing *tab*, a small window appears with a list of object names that begin with that letter.

The autocomplete feature is self-aware and will work differently depending on where it is called. As such, it works perfectly for searching for packages. For

```
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.

> f|
```

| | | | |
|---|---|---|---|
| 📄 for | {snippet} | for (${1:variable} in ${2:vector}) { | |
| 📄 fun | {snippet} | ${0} | |
| ◆ factanal | {stats} | } | |
| ◆ factor | {base} | | |
| ◆ factor.scope | {stats} | | |
| ◆ factorial | {base} | | |

Figure 2.9: Usage of autocomplete for object name

that, type `library()` in the prompt or editor, place the cursor in between
the parentheses and press *tab*. The result should look something like Figure
2.10, shown next.

```
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.

> library(r)
```

| | |
|---|---|
| 📦 rappdirs | **Application directories: determine where to save data, caches and logs.** |
| 📦 rbcb | rappdirs solves the problem of where to save persistent data |
| 📦 rcmdcheck | outside of the R library or the R per-session `tempdir`. |
| 📦 rdrop2 | Press F1 for additional help |
| 📦 readODS | |
| 📦 readr | |

Figure 2.10: Usage of autocomplete for packages

Note that a description of the package or object is also offered by the code
completion tool. This greatly facilitates the day to day work as the memo-
rization of package names and R objects can be challenging. The use of the
*tab* decreases the time to look up names, also avoiding possible errors.

The use of this tool becomes even more beneficial when objects and functions
are named with some sort of pattern. In the rest of the book, you will notice
that objects tend to be named with the prefix *my*, as `my_x`, `my_num`, `my_df`.
Using this naming rule (or any other) facilitates the lookup for the names of
objects created by the user. You can just type `my_`, press *tab*, and a list of
objects will appear.

As mentioned in the previous section, you can also find files and folders on your computer using *tab*. To try it, write the command `my_file <-` `""` in the prompt or a script, point the cursor to the middle of the quotes and press the *tab* key. A screen with the files and folders from the current working directory should appear, as shown in Figure 2.11.

```
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.

> my_file <- '|'
        📁 01-CRAN Packages
        📁 02-Finance Code
        📁 03-Bibliometric Code
        📁 04-Personal Code
        📁 05-About R and others
        📁 06-Unused and Old
```
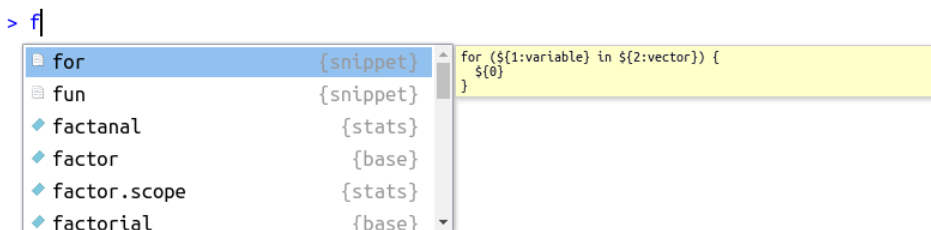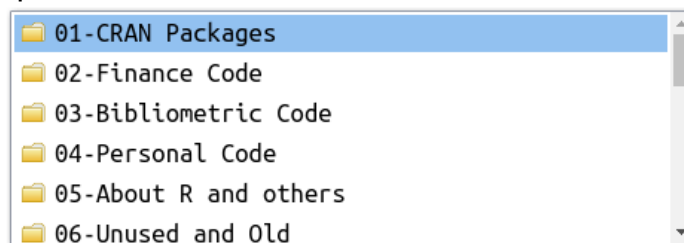
Figure 2.11: Usage of autocomplete for files and folders

The use of autocomplete is also possible for finding the name and description of function arguments. To try it out, write `message()` and place the mouse cursor inside the parentheses. After that, press *tab*. The result should be similar to Figure 2.12. By using *tab* inside of a function, we have the names of all arguments and their description – a mirror of the information found in the help files.

Likewise, you can also search for a function within a package with *tab*. For that, simply type the name of the package followed by two commas, as in `readr::`, and press *tab*. The result should be similar to Figure 2.13

Summing up, using code completion will make you more productive. You'll find names of files, objects, arguments, and packages much faster. Use it as much as you can.

> ❗ *Autocomplete* is one of the most important tools of RStudio, help-ing users to find object names, locations on the hard disk, pack-ages and functions. Get used to using the *tab* key and, soon enough, you'll see how much the *autocomplete* tool can help you write code quickly, and without typos.

```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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

  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.

> sort()
```
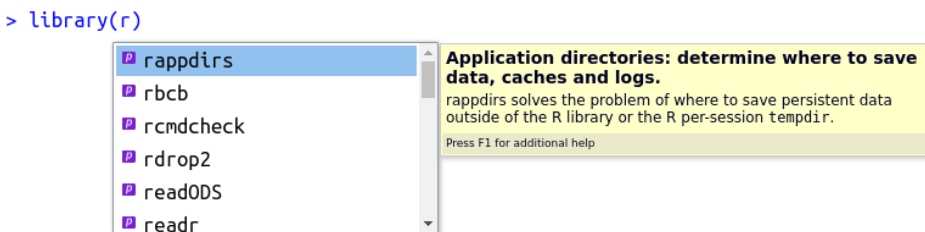
| x = | **x** |
| decreasing = | for sort an R object with a class or a numeric, complex, character or logical vector. For sort.int, a numeric, complex, character or logical vector, or a factor. |
| ... = | Press F1 for additional help |
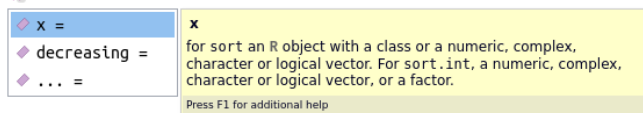
Figure 2.12: Usage of autocomplete for function arguments

```
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.

> readr::
```

| AccumulateCallback | | AccumulateCallback |
| as.col_spec | {readr} | These classes are used to define callback behaviors. |
| ChunkCallback | | Press F1 for additional help |
| clipboard | {readr} | |
| col_character | {readr} | |
| col_date | {readr} | |
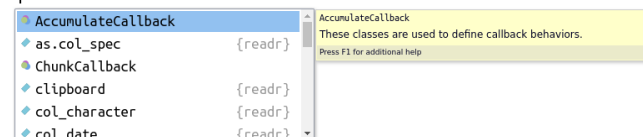
Figure 2.13: Usage of autocomplete for finding functions within a package

## 2.20 Interacting with Files and the Operating System

As you are learning R, soon enough you'll find a data-related problem that requires some interaction with files on the computer, either by creating new folders, decompressing and compressing files, listing and removing files from the hard drive of the computer or any other type of operation. R has full support for such type of operations. You can automate any type of computer task within an R script, if so needed.

In this section we will give preference to functions from package **{fs}** (Hester et al., 2023), which provides several routines for interacting with files and folders. Be aware, however, that the `base` package also offers similar functions.

### 2.20.1 Listing Files and Folders

To list files from your computer, use function **fs**::**dir\_ls()** , where the `path` argument sets the directory to list the files from. For the compilation of the book, I've created a directory called *data*. This folder contains all the data needed to recreate the book's examples. You can check the files in the sub-folder `data` with the following code:

```r
# list files in data folder
my_files <- fs::dir_ls(path = "data")
print(my_files)
```

```
R> data/FileWithLatinChar_ISO-8859-9.txt
R> data/FileWithLatinChar_UTF-8.txt
R> data/Financial Sample.xlsx
R> data/MySQLiteDatabase.SQLITE
R> data/SP500-Stocks-WithRet.rds
R> data/SP500-Stocks_long.csv
R> data/SP500-Stocks_wide.csv
R> data/SP500_long_yearly_2010-01-01_2019-11-04.rds
R> data/SQLite_db.SQLITE
R> data/UCI_Credit_Card.csv
R> data/price-data.csv
R> data/pride_and_prejudice.txt
R> data/temp.txt
R> data/temp_file.xlsx
R> data/top25babynames-by-sex-2005-2017.csv
```

There are several files with different extensions in this directory. These files

contain data that will be used in future chapters.

You can also list the files recursively over inner folders, that is, list all files from all sub-folders contained in the original path. To check it, try using the following code in your computer:

```
# list all files for all subfolders (IT MAY TAKE SOME TIME...)
fs::dir_ls(path = getwd(), recurse = TRUE)
```

The previous command will list all files in the current folder and sub-folders. Depending on the current working directory, it may take some time to run it all (you can cancel it anytime by pressing *esc* in your keyboard).

You can also set what type of output you need. For example, if you want only the available folders, and not files, use input `type = "directory"`:

```
# store names of directories
my_dirs <- fs::dir_ls(
    path = getwd(),
    type = "directory")

# print it
print(my_dirs)
```

```
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/_book
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/blocks
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/data
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/ebook files
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/eqs
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/mem_cache
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/quandl_cache
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/resources
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/tabs
R> /tmp/RtmpCpmYn0/compile-book-site--4f56560405e0/text-to-reuse
```

The output shows the directories used to write the book. It includes the directory with the data ("./data"), resources ("./resources") among others. In this same directory, you can find the chapters of the book, organized by files and based on the *RMarkdown* language (`.Rmd` file extension):

```
# list all files with the extension .Rmd
fs::dir_ls(glob = "*.Rmd")
```

```
R> 00a-About-new-edition.Rmd
R> 00b-Preface.Rmd
R> 01-Introduction.Rmd
```

```
R> 02-Basic-operations.Rmd
R> 03-Research-scripts.Rmd
R> 04-Importing-exporting-local.Rmd
R> 05-Importing-internet.Rmd
R> 06-Data-structure-objects--ONLINE.Rmd
R> 07-Basic-objects--ONLINE.Rmd
R> 08-Programming--ONLINE.Rmd
R> 09-Cleaning-data--ONLINE.Rmd
R> 10-Figures--ONLINE.Rmd
R> 11-Models--ONLINE.Rmd
R> 12-Reporting-results--ONLINE.Rmd
R> 13-Optimizing-code--ONLINE.Rmd
R> 14-References.Rmd
R> _Welcome.Rmd
R> afedR_ed03_ONLINE.Rmd
R> index.Rmd
```

The files presented above contain all the contents of this book, including this
specific paragraph, located in file `02-Basic-operations.Rmd`!

## 2.20.2 Deleting Files and Directories

You can also use an R session to delete files and directories from your com-
puter. This might come in handy when dealing with disposable data files.
Use these commands with responsibility. If not careful, you can easily break
the operating system of your computer.

You can delete files with command **fs::file_delete()** :

```
# create temporary file
my_file <- 'data/tempfile.csv'
write.csv(x = data.frame(x=1:10),
          file = my_file)

# delete it
fs::file_delete(my_file)
```

To delete directories and all their elements, we use **fs::dir_delete()** :

```
# create temp dir
fs::dir_create('temp')

# create a file inside of temp
my_file <- 'temp/tempfile.csv'
```

```
write.csv(x = data.frame(x=1:10),
          file = my_file)


fs::dir_delete('temp')
```

If needed, we can check if the deletion of a directory was successful with command **fs**::**dir_exists()** :

```
fs::dir_exists('temp')
```

```
R>  temp
R> FALSE
```

As expected, the directory was not found.

### 2.20.3   Downloading Files from the Internet

We can also use R to download files from the Internet with function **download.file()** . See the following example, where we download an Excel spreadsheet from Microsoft's website:

```
# set link
link_dl <- 'go.microsoft.com/fwlink/?LinkID=521962'
local_file <- 'data/temp_file.xlsx' # name of local file


download.file(url = link_dl,
              destfile = local_file)
```

Using **download.file()** is quite handy when you are working with Internet data that is constantly being updated. In the script, we can download new data in every execution, making sure that our analysis is based on current information.

One trick worth knowing is that you can also download files from cloud services such as Dropbox[11] and Google Drive[12]. So, if you need to send a data file to a large group of people and update it frequently, just pass the file link from the cloud service. This way, any local change in the data file in your computer will be reflected for all users with the file link.

---

[11]https://www.dropbox.com/
[12]https://drive.google.com/

Needless to say, **be very careful** with commands `fs::file_delete` and `fs::dir_delete`, especially when using recursion (`recurse = TRUE`). One simple mistake and important parts of your hard drive can be erased, breaking your operating system. Be aware that R **permanently deletes** files and folder, without any simple way to restore them.

### 2.20.4   Using Temporary Files and Directories

Every time a user starts an R session, a new temporary folder is created in your hard drive. This folder is used to store any disposable files and folders generated by R. The location of this directory is available with **fs**::**path_temp()** :

```
windows_tempdir <- fs::path_temp()
print(windows_tempdir)
```

R> C:\Users\msperlin\AppData\Local\Temp\RtmpEGt3Q7

The name of the temporary directory, in this case `'RtmpEGt3Q7'`, is unique and randomly defined at the start of every new R session. This means that every R session is linked to an unique temporary folder. If two R sessions are created, as in starting two RStudio sessions, two temporary folders will be available, each with an unique name. **When the computer is rebooted, all temporary directories are deleted**.

The same dynamic is found for file names. If you want to use a temporary random name for some reason, use **fs**::**file_temp()** :

```
windows_tempfile <- fs::file_temp()
message(windows_tempfile)
```

R> C:\Users\msperlin\AppData\Local\Temp\RtmpEGt3Q7\file19917e5fbc7e

You can also set its extension and name:

```
windows_tempfile <- fs::file_temp(
  pattern = 'temp_',
  ext = '.csv'
  )

message(windows_tempfile)
```

R> C:\Users\msperlin\AppData\Local\Temp\RtmpEGt3Q7\temp_19913e1dafe5.csv

As a practical case of using temporary files and folders, let's *download* the

Excel worksheet from Microsoft into a temporary folder and read its content
for the first five rows:

```r
# set link
link_dl <- 'go.microsoft.com/fwlink/?LinkID=521962'
local_file <- fs::file_temp(ext = '.xlsx')

download.file(url = link_dl,
              destfile = local_file)

df_msft <- readxl::read_excel(local_file)

print(head(df_msft))
```

```
R> # A tibble: 6 x 16
R>   Segment     Country Product   `Discount Band` `Units Sold`
R>   <chr>       <chr>   <chr>     <chr>                  <dbl>
R> 1 Government Canada   Carretera None                  1618.
R> 2 Government Germany  Carretera None                  1321
R> 3 Midmarket  France   Carretera None                  2178
R> 4 Midmarket  Germany  Carretera None                   888
R> 5 Midmarket  Mexico   Carretera None                  2470
R> 6 Government Germany  Carretera None                  1513
R> # i 11 more variables: `Manufacturing Price` <dbl>,
R> #   `Sale Price` <dbl>, `Gross Sales` <dbl>,
R> #   Discounts <dbl>, Sales <dbl>, COGS <dbl>, Profit <dbl>,
R> #   Date <dttm>, `Month Number` <dbl>, `Month Name` <chr>,
R> #   Year <chr>
```

The example Excel file contains the sales report of a company.  Do notice
that the imported file becomes a `dataframe` in our R session, a table like an
object with rows and columns.

By using **fs**::**file_temp()** , we do not need to delete (or worry) about the
downloaded file because it will be removed from the computer's hard disk
when the system is rebooted.

## 2.21   Exercises

---

01 - In RStudio, create a new *script* and save it in a personal folder. Now,
write R commands in the script that define two objects: one holding a
sequence between 1 and 100 and the other with the text of your name (ex.

'Richard'). Execute the whole script with the keyboard shortcuts.

---

02 - In the previously created *script*, use function `message` to display the following phrase in R's *prompt*:`"My name is ...."`.

---

03 - Within the same script, show the current working directory (see function `getwd`, as in `print(getwd())`). Now, change your working directory to *Desktop* (*Desktop*) and show the following message on the *prompt* screen: `'My desktop address is ....'`. Tip: use and abuse of RStudio's *autocomplete* tool to quickly find the *desktop* folder.

---

04 - Use R to download the compressed zip file with the book material, available at this link[13]. Save it as a file in the temporary session folder (see function `fs::file_temp()`).

---

05 - Use the `unzip` function to unzip the downloaded file from previous question to a directory called `'afedR-files'` inside the "Desktop" folder. How many files are available in the resulting folder? Tip: use the `recursive = TRUE` argument with `fs::dir_ls` to also search for all available subdirectories.

---

06 - Every time the user installs an R package, all package files are stored locally in a specific directory of the hard disk. Using command `Sys.getenv('R_LIBS_USER')` and `fs::dir_ls`, list all the directories in this folder. How many packages are available in this folder on your computer?

---

07 - In the same topic as previous exercise, list all files in all subfolders in the directory containing the files for the different packages (see command `Sys.getenv('R_LIBS_USER')`). On average, how many files are needed for each package?

---

[13]https://www.msperlin.com/files/afedr-files/afedR-code-and-data.zip

08 - Use the `install.packages` function to install the `yfR` package on your computer. After installation, use function `yf_get()` to download price data for the IBM stock in the last 15 days. Tip: use function `Sys.Date()` to find out the current date and `Sys.Date()- 15` to calculate the date located 15 days in the past.

---

09 - The `cranlogs` package allows access to downloads statistics of CRAN packages. After installing **cranlogs** on your computer, use the `cranlogs::cran_top_downloads` function to check which are the 10 most installed packages by the global community in the last month. Which package comes first? Tip: Set the `cran_top_downloads` function input to `when = 'last-month'`. Also, be aware that the answer here may not be the same as you got because it depends on the day the R code was executed.

---

10 - Using the `devtools` package, install the development version of the `ggplot2` package, available in the Hadley Hickman repository. Load the package using `library` and create a simple figure with the code `qplot(y = rnorm(10), x = 1:10)`.

---

11 - Using your programming ability check on your computer which folder, from the "Documents" directory (shortcut = ~), has the largest number of files. Display the five folders with the largest number of files on R's prompt.

# Chapter 3

# Writing Research Scripts

So far we learned how to use R for basic tasks such as interacting with the computer, creating simple vectors and downloading files from the internet. At this point, it is important to discuss the structure of a research script and, more specifically, how to organize our work in a efficient manner. As the R code base becomes larger and more complex, organization is a necessity. In this chapter, I will suggest a way to organize files and folders. So, I recommend that you follow these guidelines – or at least your own version of them – in every project you work on.

## 3.1   Stages of Research

Unlike other software designs, every script in data analysis follows through clear and consecutive steps to achieve its goal.

1. **Importation of data**: Raw (original) data is imported from a local file or the internet. At this stage, no manual data manipulation should happen. The raw data must be imported "as it is".

2. **Cleaning and structuring the data**: The raw data imported in the previous step is cleaned and structured within the needs of the analysis. Abnormal records and errors in observations can be removed or treated. The structure of the data can also be manipulated, binding (merging) different datasets and calculating variables of interest. Preferably, at the end of this stage, there should be a final collection of clean data.

3. **Visual analysis and hypothesis testing**: After cleansing and structuring the data, the work continues with the visual analysis of the data and hypothesis testing. Here, you can create graphical representations of the data for your audience and use statistical tools, such as econometric models, to test a particular hypothesis. This is the *heart* of the research and the stage most likely to take more development time.

4. **Reporting the results**: The final stage of a research script is reporting the results, that is, exporting selected tables and figures from R to a text processing software such as Latex, Writer (LibreOffice) or Word (Microsoft).

Each of the previous steps can be structured in a single *.R* script or in several separate files. Using multiple files is preferable when the first steps of the research demand significant processing time. For example, when importing and organizing a large database, it is worth the trouble to separate the code in different files. It will be easier to find bugs and maintain the code. Each script will do one job, and do it well.

A practical example would be the analysis of a large dataset of financial transactions. Importing and cleaning the data takes plenty of computer time. A smart organization is to insert these primary data procedures in a *.R* file and save the final objects of this stage in an external file. This local archive serves as a bridge to the next step, hypothesis testing, where the previously created file with clean data is imported. Every time a change is made to the hypothesis testing script, it is unnecessary to rebuild the whole dataset. This simple organization of files saves a lot of time. The underlying logic is simple, isolate the parts of the script that demand more computational time – and less development –, and connect them to the rest of the code using external data files.

If you are working with multiple files, one suggestion is to create a naming structure that informs the steps of the research in an intuitive way. An example would be to name the data importing code as `01-import-and-clean-data.R`, the modeling code as `02-estimate-and-report-models.R` and so on. The practical effect is that using a number in the first letter of the filenames clarifies the order of execution. We can also create a *master* script called `0-run-it-all.R` or `0-main.R` that runs (`source`) all other scripts. So, every time we make an update to the original data, we can simply run `0-run-it-all.R` and will have the new results, without the need to execute each script individually.

## 3.2   Folder Structure

A proper folder structure also benefits the reproducibility and organization of research. In simple scripts, with a small database and a low number of procedures, it is unnecessary to spend much time thinking about the organization of files. This is certainly the case for most of the code in this book. More complex programs, with several stages of data cleaning, hypothesis testing, and several sources of data, organizing the file structure is essential.

A suggestion for an effective folder structure is to create a single directory and, within it, create subdirectories for each input and output element. For example, you can create a subdirectory called `data`, where all the original data will be stored, a directory `fig` and `tables`, where figures and tables with final results will be exported. If you are using many custom-written functions in the scripts, you can also create a directory called `r-fcts` and save all files with function definitions at this location. As for the root of the directory, you should only find the main research scripts there. An example of a file structure that summarizes this idea is:

```
/Capital Markets and Inflation/
    /data/
        stock_indices.csv
        inflation_data.csv
    /resources/figs/
        SP500_and_inflation.png
    /tables/
        Table1_descriptive_table.tex
        Table2_model_results.tex
    /r-fcts/
        fct_models.R
        fct_clean_data.R
    0-run-it-all.R
    1-import-and-clean-data.R
    2-run-research.R
```

The research code should also be self-contained, with all files available within a sub-folder of the root directory. If you are using many different R packages, it is advisable to add a comment in the first lines of `0-run-it-all.R` that indicates which packages are necessary to run the code. The most friendly way to inform it is by adding a commented line that installs all required packages, as in `#install.packages('pkg1', 'pkg2', ...)`. So, when someone receives the code for the first time, all he (or she) needs to do is uncomment the line and execute it. External dependencies and steps

for their installation should also be informed.

The organization of the code directory facilitates collaboration and repro-
ducibility. If you need to share the code with other researchers, simply
compress the directory to a single file and send it to the recipient. After
decompressing the file, the structure of the folder immediately informs the
user were to change the original data, the order of execution of the scripts
in the root folder, and where the outputs are saved. The same benefit goes
when you reuse your code in the future, say three years from now. By work-
ing smarter, you will be more productive, spending less time with repetitive
and unnecessary steps for "figuring out" how the code works.

## 3.3   Important Aspects of a Research Script

In this section I'll be making some suggestions for how you can conduct
data analysis with R. Making it clear, these are personal positions from my
experience as a researcher and teacher. Many points raised here are specific
to the academic environment but can be easily extended to the practice of
data research in the industry. In short, these are suggestions I wish I knew
when I first started my career.

Firstly, **know your data!**. I can't stress enough how this is important!
The first instinct of every passionate data analyst when encountering a new
set of rich information is to immediately import it into R and perform an
analysis. However, a certain level of caution is needed. Every time you get
your hands on a new set of data, ask yourself how much you **really** know:

- How was the data collected? To what purpose?
- What information does each column of the table represents? What
  are the details often missed?
- How do the available data compare with data used in other studies?
- Is there any possibility of bias within the data collection?

Furthermore, you need to remember that the ultimate goal of any research
is communication. Thus, it is very likely that you will report your results to
people who will have an informed opinion about the subject, including the
sources and individualities of different datasets. The worst case scenario is
when a research effort of three to six months in between coding and writing
is nullified by a simple lapse in data checking. Unfortunately, this is not
uncommon.

As an example, consider the case of analyzing the long term performance of companies in the retail business. For that, you gather a recent list of available companies and download financial records about their revenue, profit and adjusted stock price for the past twenty years. Well, the problem here is in the selection of the companies. By selecting those that are available today, you missed all companies that went bankrupt during the 20 year period. That is, by looking only at companies that stayed active during the whole period, you indirectly selected those that are profitable and presented good performance. This is a well-known effect called **survival bias**. The right way of doing this research is gathering a list of companies in the retail business twenty years ago and keep track of those that went bankrupt and those that stayed alive.

The message is clear. **Be very cautious about the data you are using**. Your raw tables stand at the base of the research. A small detail that goes unnoticed can invalidate your whole work. If you are lucky and the database is accompanied by a written manual, break it down to the last detail. If the information is not clear, do not be shy about sending questions to the responsible party. Likewise, if there is an inevitable operational bias in your dataset, be open and transparent about it.

The second point here is the code. After you finish reading this book, you will have the knowledge to conduct research with R. The computer will be a powerful ally in making your research ideas come true, no matter how gigantic they may be. However, **a great power comes with great responsibility**. Said that, you need to be aware that a single misplace line in a code can easily bias and invalidate your analysis.

Remember that analyzing data is your profession and **your reputation is your most valuable asset**. If you have low confidence in the produced code, do not publish or communicate your results. The code and its results is entirely your responsibility. Check it as many times as necessary. Always be skeptical about your own work:

- Do the descriptive statistics of the variables faithfully report the database?
- Is there any relationship between the variables that can be verified in the descriptive table?
- Do the main findings of the research make sense to the current literature of the subject? If not, how to explain it?
- Is it possible that a *bug* in the code has produced the results?

I'm constantly surprised by how many studies submitted to respected journals can be denied publication based on a simple analysis of the descriptive

table. Basic errors in variable calculations can be easily spotted by a trained eye. The process of continuous evaluation of your research will not only make you stronger as a researcher but will also serve as practice for peer evaluation, much used in academic research. If you do not have enough confidence to report results, test your code extensively. If you have already done so and are still not confident, identify the lines of code you have doubts and seek help with a colleague or your advisor, if there is one. The latter is a strong ally who can help you in dealing with problems he/she already had.

All of the research work is, to some extent, based on existing work. Today it is extremely difficult to carry out ground-breaking research. Knowledge is built in the form of blocks, one over the other. There is always a collection of literature that needs to be consulted. Therefore, you should always compare your results with the results already available in the subject. If the main results are not similar to those found in the literature, one should ask himself: could a code error have created this result?

I clarify that it is possible that the results of research differ from those of the literature, but the opposite is more likely. Knowledge of this demands care with your code. *Bugs* and code problems are quite common and can go unnoticed, especially in early versions of scripts. As a data analyst, it is important to recognize this risk and learn to manage it.

## 3.4   Exercises

––––––––––––––––––––––

01 - Imagine a survey regarding your household budget over time. Financial data is available in electronic spreadsheets separated by month, for 10 years. The objective of the research is to understand if it is possible to purchase a real state property in the next five years. Within this setup, detail in text the elements in each stage of the study, from importing the data to the construction of the report.

––––––––––––––––––––––

02 - Based on the study proposed earlier, create a directory structure on your computer to accommodate the study. Create mock files for each subdirectory (see directory structure at section 3.2). Be aware you can create mock files and direction all in R (see functions `cat` and `fs::dir_create`).

# Chapter 4

# Importing Data from Local Files

In this chapter we'll learn to import and export data available as local files in the computer. Although the task is not particularly difficult, a data analyst should understand the different characteristics of file formats and how to take advantage of it. While some data formats are best suited for sharing and collaboration, others can offer a significant boost in reading and writing speed.

Here we will draw a comprehensive list of data file formats in R, including:

- text data with comma-separated values (*csv* files);
- spreadsheet data from Excel (*xlsx* files);
- R native data files (*RData* and *rds* files);
- Lightning Fast Serialization of data frames (FST) format (*fst* files);
- SQLite;
- unstructured text data.

The previous packages and functions are sufficient for getting most of the work done. Nevertheless, it is worth mentioning that R can also import data from other softwares such as SPSS, Stata, Matlab, among many others. If that is your case, I suggest a thorough study of package **{foreign}** (R Core Team, 2023a).

## 4.1 The path of local files

The first lesson in importing data from local files is that the location of the file must be explicitly stated in the code. The path of the file is then passed

to a function that will read the file. The best way to work with paths is to use the *autocomplete* feature of RStudio (see section 2.19). An example of full path is:

```
my_file <- 'C:/My Research/data/price-data.csv'
```

Note the use of forwarding slashes (**/**) to designate the file directory. Relative references also work, as in:

```
my_file <- 'data/price-data.csv'
```

Here, it is assumed that, in the current working folder, there is a directory called `data` and, inside of it, exists a file called `price-data.csv`. If the file path is simply its name, such as in `my_file <- 'price-data.csv'`, it is implicitly assumed that the file is located in the root of the working directory. From the previous chapter, recall that you can use **setwd()** to change the working folder to where the work is being done and simply create and use the "Projects" feature of RStudio, which automatically sets the working directory in the same location as the .Rproj file.

> I **again** reinforce the use of *tab* and the *autocomplete* tool of RStudio. It is much **easier and practical** to find files on your computer's hard disk using *tab* navigation than to copy and paste the address from your file explorer. To use it, open double or quotes in RStudio, place the *mouse* cursor in between the quotes and press *tab*.

Going further, another very important point here is that **the data from the file will be imported and exported as an object of type `dataframe`**. That is, a table contained in an Excel or *.csv* file will become a `dataframe` object in R. When we export data, the most common format is this same type of object. Conveniently, `dataframes` are nothing more than tables, with rows and columns.

Each column in the `dataframe` will have its own class, the most common being numeric (*numeric*), text (*character*), factor (*factor*) and date (*Date*). When importing the data, **it is imperative that each column is represented in the correct class**. A vast amount of errors can be avoided by simply checking the column classes in the `dataframe` resulting from the import process. For now, we only need to understand this basic property of `dataframes`. We will study the details of this object in chapter 6.

## 4.2 *csv* files

Consider a data file called CH04_SP500.csv, available from the book package. It contains daily closing prices of the SP500 index from 2010-01-04 until 2022-12-27. We will now use package **{afedR3}** (Perlin, 2023b) for finding the file and copying it to your local folder. If you followed the instructions in the book preface chapter, you should have package **{afedR3}** (Perlin, 2023b) already installed.

Once you install package **{afedR3}** (Perlin, 2023b), file CH04_SP500.csv and all other data files used in the book are downloaded from Github. The package also includes functions for facilitating the reproduction of code examples. Command **afedR3**::**data_path()** will return the local path of a data file by looking up its name.

Let's copy CH04_SP500.csv to your "My Documents" folder with the following code using the tilde (~) shortcut:

```
my_f <- afedR3::data_path('CH04_SP500.csv')
fs::file_copy(my_f, '~' )
```

Now, if it is your first time working with *.csv* files, use a file browser ("File Explorer" in Windows) and open CH04_SP500.csv in the "My Documents" folder with any text editor software such as Notepad. The first lines of CH04_SP500.csv, also called header lines, show the column names. Following international standards, rows are set using line breaks, and all columns are separated by commas (,). Next we show the textual content of the first ten lines of CH04_SP500.csv:

```
R> ref_date,price_close
R> 2010-01-04,1132.98999
R> 2010-01-05,1136.52002
R> 2010-01-06,1137.140015
R> 2010-01-07,1141.689941
R> 2010-01-08,1144.97998
R> 2010-01-11,1146.97998
R> 2010-01-12,1136.219971
R> 2010-01-13,1145.680054
R> 2010-01-14,1148.459961
```

The data in CH04_SP500.csv is organized in a standard way, and we should have no problem importing it in R. However, you should be aware this is not always the case. So, if you want to avoid the most common issues when importing data from *csv* files, I suggest you follow these steps:

1) Check the existence of text before the actual data. A standard *.csv* file will only have the contents of a table but, sometimes, you will find a header text with metadata (extra information about the dataset). If necessary, you can control how many lines you skip in the *csv* reading function;

2) Verify the existence of names for all columns and if those names are readable;

3) Check the symbol for column separation. Normally it is a comma, but you never know for sure;

4) For the numerical data, verify the decimal symbol. R will expect it to be a dot. If necessary, you can adjust this information in the code, making sure R knows how to correctly parse numerical data.

5) Check the encoding of the text file. Normally it is one of UTF-8, Latin1 (ISO-8859) or Windows 1252. These are broad encoding formats and should suffice for most cases. Whenever you find strange symbols in the text columns of the resulting `dataframe`, the problem is probably due to a mismatch between the encoding of the file and R. Windows users can check the encoding of a text file by opening it in Notepad++[1]. The encoding format is available in the bottom right corner of the Notepad++ editor. Linux and Mac users can find the same information in any advanced text editor software such as Kate[2].

> Whenever you find an unexpected text structure in a *.csv* file, use the arguments of the *csv* reading function to import the information correctly. As a rule of thumb, **never modify raw data manually**. Its far more efficient to use R code to deal with different structures of *.csv* files. It takes a bit of work, but such a policy will save you a lot of time in the future as, in a couple of months, you are unlikely to remember how you manually cleaned that *csv* file in order to import it more easily in your R session.

## 4.2.1   Importing Data

The **{base}** (R Core Team, 2023b) package includes a native function called **read.csv()** for importing data from *.csv* files. However, we will prefer the **{tidyverse}** (Wickham, 2023) alternative, **readr**::**read_csv()** , as it

---

[1]https://notepad-plus-plus.org/
[2]https://kate-editor.org/

is more efficient and easier to work with. In short, the benefit of using **readr**::**read_csv()** is that it reads the data very quickly, with clever internal rules for defining the classes of imported columns.

This is the first package from the **{tidyverse}** (Wickham, 2023) that we will use. Before doing so, it is necessary to install it in your R session. A simple way of installing all **{tidyverse}** (Wickham, 2023) packages as a bundle is as follows:

```r
install.packages('tidyverse')
```

After running the previous code, all **{tidyverse}** (Wickham, 2023) packages will be installed on your computer. Once it finishes, let's load it.

```r
# load library
library(tidyverse)
```

Back to importing data from *.csv* files, to load the contents of file CH04_SP500.csv in R, use the **readr**::**read_csv()** function.

```r
# set file to read
my_f <- afedR3::data_path('CH04_SP500.csv')

# read file
my_df_sp500 <- readr::read_csv(my_f)

# print it
print(head(my_df_sp500))
```

```
R> # A tibble: 6 x 2
R>   ref_date    price_close
R>   <date>          <dbl>
R> 1 2010-01-04       1133.
R> 2 2010-01-05       1137.
R> 3 2010-01-06       1137.
R> 4 2010-01-07       1142.
R> 5 2010-01-08       1145.
R> 6 2010-01-11       1147.
```

As previously mentioned, the contents of the imported file becomes a `dataframe` object in R, and each column of a `dataframe` has a class. We can check the classes of object `my_df_sp500` using function **dplyr**::**glimpse()** :

```r
# Check the content of dataframe
dplyr::glimpse(my_df_sp500)
```

```
R> Rows: 3,269
R> Columns: 2
R> $ ref_date    <date> 2010-01-04, 2010-01-05, 2010-01-06, 2~
R> $ price_close <dbl> 1132.99, 1136.52, 1137.14, 1141.69, 11~
```

Note that the column of dates – `ref_date` – was imported as a `Date` vector and the closing prices – `price_close` – as numeric (`dbl`, double accuracy). This is exactly what we expected. Internally, function **readr::read_csv()** identifies columns classes according to their content.

When calling **readr::read_csv()** without extra arguments, the function will present in the prompt the classes of each column. Internally, the function sets the attributes of the columns by reading the first 1000 lines of the file. Column `ref_date` was imported with the *Date* class and column `price_close` was imported as *double* (*dbl*). We can use this information in our own code by copying the text and assigning it to a variable. Have a look:

```
# set cols from import message
my_cols <- readr::cols(
  ref_date = readr::col_date(),
  price_close = readr::col_double()
  )


# read file with readr::read_csv
my_df_sp500 <- readr::read_csv(my_f, col_types = my_cols)
```

This time, not message was set to the prompt. As an exercise, Let's import the same data, but use a `character` class for both columns:

```
# set cols from import message
# set cols from import message
my_cols <- readr::cols(
  ref_date = readr::col_character(),
  price_close = readr::col_character()
  )

# read file with readr::read_csv
my_df_sp500 <- readr::read_csv(my_f, col_types = my_cols)

# glimpse the dataframe
dplyr::glimpse(my_df_sp500)
```

```
R> Rows: 3,269
R> Columns: 2
R> $ ref_date    <chr> "2010-01-04", "2010-01-05", "2010-01-0~
```

```
R> $ price_close <chr> "1132.98999", "1136.52002", "1137.1400~
```

As expected, both columns are now of class `character`.

Going further, **readr::read_csv()** has several other input options such as:

- change the format of the import data, including symbols for decimal places and encoding (`locale` option);
- change column names (argument `col_names`);
- skip *n* lines before importation (`skip` option);
- custom definition for NA values (`na` option)

As an example, let's study the case of a messy *.csv* file. In the book package we have a file called `funky_csv_file.csv` where:

- the header has textual information;
- the file will use the comma as a decimal;
- the file text will contain Latin characters.

The first 10 lines of the files contain the following content:

```
R> Example of funky file:
R> - columns separated by ";"
R> - decimal points as ","
R>
R> Data build in 2022-12-28
R> Origin: www.funkysite.com.br
R>
R> ID;Race;Age;Sex;Hour;IQ;Height;Died
R> 001;White;80;Male;00:00:00;92;68;FALSE
R> 002;Hispanic;25;Female;00:00:00;99;68;TRUE
```

We have a header text up to line number 7 and the columns being separated by a semicolon (";"). When importing the data with standard (and wrong) options, we will have the following output:

```
my_f <- afedR3::data_path('CH04_funky-csv-file.csv')

df_funky <- readr::read_csv(my_f)
```

The output shows that only one column of class *character* was read from the file. This happens because function **readr::read_csv()** expects actual data (and not text) in the first line of the file. To solve it, we need to use several input arguments to handle the particularities of the file:

```r
df_not_funky <- readr::read_delim(
        file = my_f, # path of file
        skip = 7, # how many lines do skip
        delim = ';', # symbol for column separator
        col_types = readr::cols(), # column types
        locale = readr::locale(decimal_mark = ',') # locale
)


dplyr::glimpse(df_not_funky)
```

```
R> Rows: 100
R> Columns: 8
R> $ ID     <chr> "001", "002", "003", "004", "005", "006", "~
R> $ Race   <chr> "White", "Hispanic", "Asian", "White", "Whi~
R> $ Age    <dbl> 80, 25, 25, 64, 76, 89, 33, 61, 23, 59, 80,~
R> $ Sex    <chr> "Male", "Female", "Male", "Male", "Female",~
R> $ Hour   <time> 00:00:00, 00:00:00, 00:00:00, 00:00:00, 00~
R> $ IQ     <dbl> 92, 99, 98, 105, 109, 84, 109, 109, 99, 126~
R> $ Height <dbl> 68, 68, 69, 69, 67, 73, 65, 72, 70, 66, 63,~
R> $ Died   <lgl> FALSE, TRUE, TRUE, TRUE, FALSE, TRUE, FALSE~
```

Note that the data has now been correctly imported, with the correct column classes. For that, we use the alternative function **readr::read_delim()** with custom inputs. Package **{readr}** (Wickham et al., 2023c) also provides several other functions for specific import situations.

## 4.2.2   Exporting Data

To write a *.csv* file, use the **readr::write_csv()** function. First, we create a new dataframe with some random data:

```r
# set the number of rows
N <- 100


# set dataframe
my_df <- data.frame(y = runif(N),
                    z = rep('a', N))


# print it
print(head(my_df))
```

```
R>           y z
R> 1 0.3699533 a
```

```
R> 2 0.9320087 a
R> 3 0.7509318 a
R> 4 0.5766172 a
R> 5 0.6606423 a
R> 6 0.4357343 a
```

And now we use **readr::write_csv()** to save it in a new (and temporary) *.csv* file:

```
# set file out
f_out <- fs::file_temp(ext = 'csv')

# write to files
readr::write_csv(x = my_df,
          file = f_out)
```

In the previous example, we save the object `my_df` into a temporary file with path file4f561324f31e.csv. We can read it back and check its contents using **readr::read_csv()** once again:

```
# read it
my_df_imported <- readr::read_csv(f_out)

# print first five rows
print(head(my_df_imported))
```

```
R> # A tibble: 6 x 2
R>       y z
R>   <dbl> <chr>
R> 1 0.370 a
R> 2 0.932 a
R> 3 0.751 a
R> 4 0.577 a
R> 5 0.661 a
R> 6 0.436 a
```

As we can see, the data imported from the file is identical to the one created in the other code chunk.

## 4.3 *Excel* Files (*xlsx*)

Although it is not an efficient or portable data storage format, Microsoft Excel is a popular software in Finance and Economics due to its spreadsheet-like capacities. It is not uncommon for data to be stored and distributed in

this format.

The downside of using Excel files for storing data is its low portability and the longer reading and writing times. This may not be a problem for small tables, but when handling a large volume of data, using Excel files can be very frustrating. If you can, my advice is to avoid the use of Excel files in your work cycle.

## 4.3.1   Importing Data

R does not have a native function for importing Excel files. Therefore, we must install and use packages to perform this operation. There are several good options including **{XLConnect}** (Mirai Solutions GmbH, 2023), **{xlsx}** (Dragulescu and Arendt, 2020), **{readxl}** (Wickham and Bryan, 2023) and **{tidyxl}** (Garmonsway, 2023).

Despite their similar goals, each package has its peculiarities. If reading Excel files is important to your work, I strongly advise the study of each package. For example, package **{tidyxl}** (Garmonsway, 2023) was specially designed to read unstructured Excel files, where the desired information is not contained in a tabular format. Alternatively, package **{XLConnect}** (Mirai Solutions GmbH, 2023) allows the user to open a live connection and, from R, control an Excel file, making it possible to export and send data, format cells, and so on.

In this section, we will give priority to package **{readxl}** (Wickham and Bryan, 2023), one of the most straightforward packages to interact with Excel files. Conveniently, it does not require the installation of external software.

Let's start with an example. Consider a file called `CH04_SP500-Excel.xlsx` that contains the same SP500 data from previous section. We can import the information from the file using function **readxl::read_excel()** :

```r
# set excel file
my_f <- afedR3::data_path('CH04_SP500-Excel.xlsx')

# read excel file
my_df <- readxl::read_excel(my_f, sheet = 'Sheet1')

# print with head (first five rows)
dplyr::glimpse(my_df)
```

```
R> Rows: 3,269
R> Columns: 2
```

```
R> $ ref_date   <dttm> 2010-01-04, 2010-01-05, 2010-01-06, 2~
R> $ price_close <dbl> 1132.99, 1136.52, 1137.14, 1141.69, 11~
```

One of the benefits of using Excel files is that the column's classes are directly inherited from the file. If the column classes are correct in the Excel file, then they will automatically be correct in R. In our case, the date column of file CH04_SP500-Excel.xlsx was correctly set as a `dttm` object, a special type of `DateTime` class. Likewise, even if the Excel file used commas for decimals, the import process would still succeed as the conversion is handled internally.

### 4.3.2 Exporting Data

Exporting a `dataframe` to an Excel file is also easy. Again, no native function in R performs this procedure. We can, however, use package **{writexl}** (Ooms, 2023):

```
# set number of rows
N <- 25

# create random dfs
my_df_A <- data.frame(y = seq(1, N),
                      z = rep('a', N))

writexl::write_xlsx(
  x = my_df_A,
  path = f_out
  )
```

## 4.4  *RData* and *rds* Files

R offers two native formats to write objects to a local file, *RData*, and *rds*. The benefit of using both is that the saved file is compact and its access is very fast. The downside is the low portability, i.e., it's difficult to use the files in other software.

The difference between *RData* and *rds* is that the first can save many R objects in a single file, while the latter only one. This, however, is not a hard restriction for the *rds* format as we can incorporate several objects into a single one using `lists`. In practice, a *rds* file can store as many objects as needed. We will learn more about `lists` in chapter 6.

### 4.4.1   Importing Data

To create a new *.RData* file, use the **save()** function.  See the following example, where we create a *.RData* file with some content, clear R's memory, and then load the previously created file:

```r
# set a object
my_x <- 1:100

# set name of RData file
my_file_rdata <- fs::file_temp(ext = 'RData')
my_file_rds <- fs::file_temp(ext = 'rds')

# save it in RData
save(list = c('my_x'), file = my_file)

# save it in rds
readr::write_rds(my_x, my_file_rds)
```

We can verify the existence of the file with the **fs::file_exists()** function:

```r
# check if file exists
fs::file_exists(my_file_rdata)
```

```
R> /tmp/RtmpCpmYn0/file4f569d1f12.RData
R>                               FALSE
```

As expected, file price-data.csv is available.

Now, to import data from **.rds** files, we use function **readr::read_rds()** :

```r
# load content into workspace
my_x_2 <- readr::read_rds(file = my_file_rds)
```

Comparing the code between using **.RData** and **.rds** files, note that the **.rds** format allows the explicit definition of the output object.  The contents of `my_file_rds` is saved in `my_x_2`. When we use the **load()** function for *RData* files, we cannot name the output directly.  This is particularly inconvenient when you need to modify the name of the imported object.

> **!**  As a suggestion, use the *.rds* format to write and read R data. Its use is more practical, resulting in cleaner code.  The difference in speed between one and the other is minimal. The benefit of importing multiple objects into the same *RData* file becomes irrelevant when using `list` objects, which can incorporate other objects into its content.

## 4.4.2  Exporting Data

We can create a new *RData* file with command **save()** :

```
# set vars
my_x <- 1:100
my_y <- 1:100

# write to RData
my_file <- fs::file_temp(ext = 'RData')
save(list = c('my_x', 'my_y'),
     file = my_file)
```

We can again check if the file exists:

```
fs::file_exists(my_file)
```

```
R> /tmp/RtmpCpmYn0/file4f567c72cdfd.RData
R>                                     TRUE
```

The result is TRUE as expected.

As for *.rds* files, we save it with function **readr::write_rds()** :

```
# set data and file
my_x <- 1:100
my_file <- fs::file_temp(ext = 'rds')

# save as .rds
readr::write_rds(x = my_x,
          file = my_file)

# read it
my_x2 <- readr::read_rds(file = my_file)

# test equality
print(identical(my_x, my_x2))
```

```
R> [1] TRUE
```

Command **identical()** tests if both objects are equal.  Again, as expected, we find the result to be TRUE.

## 4.5   *fst* files

The *fst* format[3], R package **{fst}** (Klik, 2022), is specially designed to enable quick writing and reading time from tabular data, with minimal disk space. Using this format is particularly beneficial when working with large databases in powerful computers. The trick here is the use of all computer cores to import and export data, while all other formats only use one. If you have a computer with several cores, the gain in speed is impressive, as we will soon learn.

### 4.5.1   Importing Data

Using *fst* file format is similar to the previous cases. We use function **fst::read_fst()** to read files:

```r
# set file location
my_file <- afedR3::data_path('CH04_example-fst.fst')

# read fst file
my_df <- fst::read_fst(my_file)

# check contents
dplyr::glimpse(my_df)
```

```
R> Rows: 100
R> Columns: 8
R> $ ID     <chr> "001", "002", "003", "004", "005", "006", "~
R> $ Race   <fct> Black, White, Hispanic, Black, White, White~
R> $ Age    <int> 33, 35, 23, 87, 65, 51, 58, 67, 22, 52, 52,~
R> $ Sex    <fct> Male, Female, Male, Female, Male, Male, Fem~
R> $ Hour   <dbl> 0.00000000, 0.00000000, 0.00000000, 0.00000~
R> $ IQ     <dbl> 108, 108, 85, 106, 92, 92, 88, 100, 86, 80,~
R> $ Height <dbl> 72, 63, 77, 72, 71, 74, 64, 69, 63, 72, 70,~
R> $ Died   <lgl> FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, TRUE~
```

As with the other cases, the data from file CH04_example-fst.fst is available in the workspace.

### 4.5.2   Exporting Data

We use function **fst::write_fst()** to save `dataframes` in the *fst* format:

---

[3]http://www.fstpackage.org/

```r
# create dataframe
N <- 1000
my_file <- fs::file_temp(ext = 'fst')
my_df <- data.frame(x = runif(N))

# write to fst
fst::write_fst(x = my_df, path = my_file)
```

## 4.5.3 Timing the *fst* format

As a test of the potential of the `fst` format, we will now time the read and write time between `fst` and `rds` for a large table: 5.000.000 rows and 2 columns. We will also report the size of the resulting file.

```r
# set number of rows
N <- 5000000

# create random dfs
my_df <- data.frame(y = seq(1,N),
                    z = rep('a',N))

# set files
my_file_1 <- fs::file_temp(ext = 'rds')
my_file_2 <- fs::file_temp(ext = 'fst')

# test write
time_write_rds <- system.time(readr::write_rds(my_df, my_file_1 ))
time_write_fst <- system.time(fst::write_fst(my_df, my_file_2 ))

# test read
time_read_rds <- system.time(readr::read_rds(my_file_1))
time_read_fst <- system.time(fst::read_fst(my_file_2))

# test file size (MB)
file_size_rds <- file.size(my_file_1)/1000000
file_size_fst <- file.size(my_file_2)/1000000
```

And now we check the results:

```r
# results
my_formats <- c('.rds', '.fst')
results_read <- c(time_read_rds[3], time_read_fst[3])
results_write<- c(time_write_rds[3], time_write_fst[3])
```

```
results_file_size <- c(file_size_rds , file_size_fst)

# print text
my_text <- paste0('\nTime to WRITE dataframe with ',
                  my_formats, ': ',
                  results_write, ' seconds', collapse = '')
message(my_text)
```

```
R>
R> Time to WRITE dataframe with .rds: 0.545999999999992 seconds
R> Time to WRITE dataframe with .fst: 0.089999999999975 seconds
```

```
my_text <- paste0('\nTime to READ dataframe with ',
                  my_formats, ': ',
                  results_read, ' seconds', collapse = '')
message(my_text)
```

```
R>
R> Time to READ dataframe with .rds: 0.617000000000019 seconds
R> Time to READ dataframe with .fst: 0.099000000000018 seconds
```

```
my_text <- paste0('\nResulting FILE SIZE for ',
                  my_formats, ': ',
                  results_file_size, ' MBs', collapse = '')
message(my_text)
```

```
R>
R> Resulting FILE SIZE for .rds: 65.000177 MBs
R> Resulting FILE SIZE for .fst: 14.791938 MBs
```

The difference in performance is impressive! The `fst` not only reads and writes faster but also results in smaller file sizes. Be aware, however, this result is found in a sixteen core computer in which the book was compiled. You may not be able to replicate the same result in a more modest machine.

> **!** Due to the use of all the computer's cores, the *fst* format is highly recommended when working with large data on a powerful computer. Not only will the resulting files be smaller, but the writing and reading process will be considerably faster.

# 4.6 SQLite Files

The use of *.csv* or *.rds* files for storing objects has its limits as the volume of data increases. If you are waiting a long time to read a `dataframe` from a file or if you are only interested in a small portion of a large table, you should look for alternatives. Likewise, if you have access to the computer network in your institution, including databases, it is important to learn how to directly import the corporate data into your R session.

This brings us to the topic of **database software**. These specific programs usually work with a query language, called *SQL* (*Structured Query Language*). It allows the user to read portions of the data and even manipulate it efficiently. Many options of database software integrate nicely with R. The list includes **MySQL**, **SQLite** and **MariaDB**. Here, we will provide a quick tutorial on this topic using SQLite, which is the easiest one to work with. Unlike other database software, SQLite stores data and configurations from a single local file, without the need of a formal server.

## 4.6.1 Importing Data

Before moving to the examples, we need to understand how to use database software. First, R will connect to the database and return a connection object. Based on this connection, we will send queries for manipulating data using the *SQL* language. The main advantage is we can have a large database of, let's say, 10 GB and only load a small portion of it in R. This operation is also very quick, allowing efficient access to the available tables.

Assuming the existence of an SQLite file in the computer, we can import its tables with package **{RSQLite}** (Müller et al., 2023) :

```r
# set name of SQLITE file
f_sqlite <- afedR3::data_path('CH04_example-sqlite.SQLite')

# open connection
my_con <- RSQLite::dbConnect(
  drv = RSQLite::SQLite(),
  f_sqlite)

# read table
my_df <- RSQLite::dbReadTable(conn = my_con,
                    name = 'MyTable1') # name of table in sqlite

# print with str
dplyr::glimpse(my_df)
```

```
R> Rows: 1,000
R> Columns: 2
R> $ x <dbl> 0.24645265, 0.47972926, 0.36782192, 0.09451063, ~
R> $ G <chr> "B", "B", "A", "A", "A", "A", "A", "B", "B", "A"~
```

It worked. The `dataframe` from table `MyTable1` is exactly as expected.

Another example of using SQLite is with the actual SQL statements. Notice, in the previous code, we used function **RSQLite::dbReadTable()** to get the contents of all rows in table `MyTable1`. Now, let's use an SQL command to get from `MyTable2` only the rows where the `G` column is equal to `A`.

```
# set sql statement
my_SQL_statement <- "select * from myTable2 where G='A'"

# get query
my_df_A <- RSQLite::dbGetQuery(conn = my_con,
                        statement = my_SQL_statement)

# disconnect from db
RSQLite::dbDisconnect(my_con)

# print with glimpse
dplyr::glimpse(my_df_A)
```

```
R> Rows: 522
R> Columns: 2
R> $ x <dbl> 0.9263521, 0.1304614, 0.1758632, 0.2323157, 0.84~
R> $ G <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A"~
```

It also worked, as expected.

In this simple example, we can see how easy it is to create a connection to a database, retrieve tables, and disconnect. If you have to work with large tables, which, in my opinion, is any database that occupies more than 4 GB of your computer memory, it is worth moving it to proper database software. You'll be able to retrieve data faster, without the need of loading the whole database in the computer's memory. If you have a server available in your workplace, I strongly advise learning how to connect to it and use the SQL language to your advantage.

## 4.6.2   Exporting Data

As an example of exporting data to an SQLite file, let's first create an SQLite database. For that, we will set two large `dataframes` with random data and

save both in an SQLite file.

```r
# set number of rows in df
N = 10^6

# create simulated dataframe
my_large_df_1 <- data.frame(x=runif(N),
                            G= sample(c('A','B'),
                                      size = N,
                                      replace = TRUE))

my_large_df_2 <- data.frame(x=runif(N),
                            G = sample(c('A','B'),
                                       size = N,
                                       replace = TRUE))

# set path of SQLITE file
f_sqlite <- fs::file_temp(ext = 'SQLITE')

# open connection
my_con <- RSQLite::dbConnect(drv = RSQLite::SQLite(), f_sqlite)

# write df to sqlite
RSQLite::dbWriteTable(conn = my_con, name = 'MyTable1',
             value = my_large_df_1)

RSQLite::dbWriteTable(conn = my_con, name = 'MyTable2',
             value = my_large_df_2)

# disconnect
RSQLite::dbDisconnect(my_con)
```

The TRUE output of **RSQLite**::**dbWriteTable()** indicates everything went well. A connection was opened using function **RSQLite**::**dbConnect()** , and both `dataframes` were written to an SQLite file, called file4f56c75fe63.SQLITE.

# 4.7 Unstructured Data and Other Formats

Another example of data importation is the case of reading and processing unstructured text files. If none of the previous packages can read the data,

then it must be parsed line by line. Let's explore this problem.

## 4.7.1   Importing Data

You can read the contents of a text file with function **readr::read_lines()**:

```
# set file to read
my_f <- afedR3::data_path('CH04_price-and-prejudice.txt')

# read file line by line
my_txt <- readr::read_lines(my_f)

# print 50 characters of first fifteen lines
print(stringr::str_sub(string = my_txt[1:15],
                       start = 1,
                       end = 50))
```

```
R>  [1] "                            [Illustration:"
R>  [2] ""
R>  [3] "                           GEORGE ALLEN"
R>  [4] "                             PUBLISHER"
R>  [5] ""
R>  [6] "                       156 CHARING CROSS ROAD"
R>  [7] "                            LONDON"
R>  [8] ""
R>  [9] "                          RUSKIN HOUSE"
R> [10] "                                ]"
R> [11] ""
R> [12] "                            [Illustration:"
R> [13] ""
R> [14] "                 _Reading Jane's Letters._      _Cha"
R> [15] "                                ]"
```

In this example, file CH04_price-and-prejudice.txt contains the whole content of the book *Pride and Prejudice* by Jane Austen, freely available in the Gutenberg[4] project, and downloaded with package **{gutenbergr}** (Johnston and Robinson, 2023). We imported the entire content of the file as a `character` vector named `my_txt`. Each element of `my_txt` is a line from the raw text file. Based on it, we can check the number of lines in the book, and also the number of times that the name `'Bennet'`, one of the protagonists, appears in the text:

---

[4]http://www.gutenberg.org/

```r
# count number of lines
n_lines <- length(my_txt)

# set target text
name_to_search <- 'Bennet'

# set function for counting words
fct_count_bennet <- function(str_in, target_text) {

  n_words <- length(
    stringr::str_locate_all(string = str_in,
                                      pattern = target_text)[[1]])

  return(n_words)
}

# use fct for all lines of Pride and Prejudice
n_times <- sum(sapply(X = my_txt,
                      FUN = fct_count_bennet,
                      target_text = name_to_search))

# print results
my_msg <- paste0('The number of lines found in the file is ',
                 n_lines, '.\n',
                 'The word "', name_to_search, '" appears ',
                 n_times, ' in the book.')
message(my_msg)
```

```
R> The number of lines found in the file is 14529.
R> The word "Bennet" appears 696 in the book.
```

In the example, we used function **sapply()** . In this case, it allowed us to use a function for each element of `my_txt`. We searched and counted the number of times the word "Bennet" was found.

## 4.7.2  Exporting Data

A typical case of exporting unstructured text is saving the log record of a procedure. This is quite simple. Using function **readr::write_lines()** , use the input `file` to set the name of the local file and `x` for the actual textual content.

```r
# set file
my_f <- 'data/temp.txt'

# set some string
my_text <- paste0('Today is ', Sys.Date(), '\n',
                  'Tomorrow is ', Sys.Date()+1)

# save string to file
readr::write_lines(x = my_text, file = my_f, append = FALSE)
```

In the previous example, we created a simple text object and saved it in data/temp.txt. We can check the result with the **readr::read_lines()** function:

```r
print(readr::read_lines(my_f))
```

```
R> [1] "Today is 2023-12-13"    "Tomorrow is 2023-12-14"
```

As we can see, it worked as expected.

## 4.8   How to Select a Data File Format

The choice of file format is an important topic and might actually be a time-saver at your work. In that decision, my first and most important suggestion is **to avoid the Excel format at all costs**. As for alternatives, we must consider three points in the decision:

- **speed** of reading and writing operations;
- **size** of the resulting file;
- **compatibility** with other software and operating systems.

Usually, the use of *csv* files easily satisfies these requirements. A *csv* file is nothing more than a text file that can be opened, viewed, and imported into any other statistical software. This makes it easy to share it with other people. Also, the size of *csv* files is usually not restrictive and, if needed, the file can be compressed using the **zip()** function. For these reasons, the use of *csv* files for importing and exporting data is preferable in the vast majority of situations.

However, there are cases where the speed of import and export operations matter. If you don't mind giving up portability, the *rds* format is a great choice for most projects. If you have good hardware and execution speed

with *rds* is still not great, then the best alternative is the *fst* format, which uses all cores to import and export data.

# 4.9 Exercises

---

01 - Create a `dataframe` with the following code:

```
library(dplyr)

my_N <- 10000
my_df <- tibble(x = 1:my_N,
                y = runif(my_N))
```

Export the resulting `dataframe` to each of the five formats: csv, rds, xlsx, fst. Which of the formats took up the most space in the computer's memory? Tip: `file.size` calculates the size of files within R.

---

02 - Improve the previous code by measuring the execution time for saving the data in different formats. Which file format resulted in the fastest execution for exporting data? Tip: use the `system.time` function or the `tictoc` package to calculate the execution times.

---

03 - For the previous code, reset the value of `my_N` to `1000000`. Does it change the answers to the last two questions?

---

04 - Use `afedR3::data_path` function to access the CH04_SP500.csv file in the book's data repository. Import the contents of the file into R with the function `readr::read_csv`. How many lines are there in the resulting `dataframe`?

---

05 - At link https://eeecon.uibk.ac.at/~zeileis/grunfeld/Grunfeld.csv/ you'll find a *.csv* file for the *Grunfeld* data. This is a particularly famous table due to its use as reference data in econometric models. Using `readr::read_csv` function, read this file using the direct link as input `read_csv`. How many columns do you find in the resulting `dataframe`?

---

06 - Use function `afedR3::data_path` function to access the CH04_example-tsv.tsv file in the book's data repository. Note that the columns of the data are separated by the tab symbol (`'\t'`), and not the usual comma. After reading the `readr::read_delim` manual, import the information from this file to your R session. How many rows does the resulting `dataframe` contain?

---

07 - In the book package you'll find data file called CH04_another-funky-csv-file.csv, with a particularly bizarre format. Open it in a text editor and try to understand how the columns are separated and what is symbol for decimals. After that, study the inputs of function `utils::read.table` and import the table into your R session. If we add the number of rows to the number of columns in the imported table, what is the result?

# Chapter 5

# Importing Data from the Internet

One of the great advantages of using R in Finance and Economics is the large amount of data that can be imported using the internet, substituting the tedious, unreliable and soul-crushing work of manual data collection. It also becomes easier to share reproducible code, as anyone can feasibly download the same tables with a single line of code.

In this chapter, we will study the most important and reliable packages for data importation in the fields of Finance and Economics. It is a small, but comprehensive list of packages that cover a large range of research topics. The list includes:

**{GetQuandlData} (Perlin, 2023c)** Imports economical and financial data from the Quandl platform.

**{yfR} (Perlin, 2023a)** Imports adjusted and unadjusted stock price data from Yahoo Finance.

**{simfinapi} (Gomolka, 2023)** Imports financial statements and adjusted stock prices from the SimFin project[1].

**{tidyquant} (Dancho and Vaughan, 2023)** Imports several financial information about stock prices and fundamental data.

## 5.1   Package {GetQuandlData}

*Quandl* is an established and comprehensive platform that provides access to a series of free and paid data. Several central banks and research institutions

---

[1]https://simfin.com/

provide free economic and financial information on this platform. I strongly recommend browsing the available tables from the Quandl website[2]. It is likely that you'll find datasets that you're familiar with.

In R, package **{Quandl}** (Raymond McTaggart et al., 2021) is the official extension offered by the company and available in CRAN. However, the package has some issues (see blog post here[3]), which are fixed with the alternative package **{GetQuandlData}** (Perlin, 2023c).

The **first and mandatory** step in using **{GetQuandlData}** (Perlin, 2023c) is to register a user at the Quandl website[4]. Soon after, go to *account settings* and click *API KEY*. This page should show a code, such as `Asv8Ac7zuZzJSCGxynfG`. Copy it to the clipboard (*control + c*) and, in R, define a character object containing the copied content as follows:

```
# set FAKE api key to quandl
my_api_key <- 'Asv8Ac7zuZzJSCGxynfG'
```

The API key is unique to each user, and the one presented here will not work on your computer. You'll need to get your own API key to run the examples of the book. After finding and setting your key, go to Quandl's website and use the search box to look for the symbol of the time series of interest. As an example, we will use data for gold prices in the London Market, with a Quandl code equivalent to `'LBMA/GOLD'`. Do notice that the structure of a Quandl code is always the same, with the name of the main database at first, and the name of table second, separated by a forward slash (/).

Now, with the API key and the Quandl symbol, we use function **GetQuandlData::get_Quandl_series()** to download the data from 1980-01-01 to 2023-01-01:

```
# set symbol and dates
my_symbol <- c('GOLD' = 'LBMA/GOLD')
first_date <- '1980-01-01'
last_date <- '2023-01-01'

# get data!
df_quandl <- GetQuandlData::get_Quandl_series(
  id_in = my_symbol,
  api_key = my_api_key,
  first_date = first_date,
```

---

[2]https://data.nasdaq.com/
[3]https://www.msperlin.com/post/2019-10-01-new-package-getquandldata/
[4]https://data.nasdaq.com/

```
  last_date = last_date,
  do_cache = FALSE)

# check it
dplyr::glimpse(df_quandl)
```

```
R> Rows: 10,866
R> Columns: 9
R> $ `USD (AM)`  <chr> "559", "632", "596", "634", "615.75", ~
R> $ `USD (PM)`  <chr> "559.5", "634", "588", "633.5", "610",~
R> $ `GBP (AM)`  <chr> "251.123", "281.577", "266.285", "281.~
R> $ `GBP (PM)`  <chr> "250.841", "282.468", "262.77", "280.0~
R> $ `EURO (AM)` <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
R> $ `EURO (PM)` <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
R> $ series_name <chr> "GOLD", "GOLD", "GOLD", "GOLD", "GOLD"~
R> $ ref_date    <date> 1980-01-02, 1980-01-03, 1980-01-04, 1~
R> $ id_quandl   <chr> "LBMA/GOLD", "LBMA/GOLD", "LBMA/GOLD",~
```

Notice how we set the name of the time series in line `id_in = c('GOLD' = 'LBMA/GOLD')`. The name of the element becomes the value of column `series_name` in `df_quandl`. If we had more time series, they would be stacked in the same table, but with different `series_name` value.

There are other `Quandl` API options available with inputs `order`, `collapse` and `transform`. If using `Quandl` is important to your work, I strongly recommend reading the available parameters for querying data[5]. Several choices for data transformations can be passed to function **GetQuandlData**::**get_Quandl_series()** .

As an inspection check, let's plot the prices of Gold in USD over time.

Overall, gold prices were fairly stable between 1980 and 2000, reaching a spike after 2010. One possible explanation is the higher demand for safer assets, such as gold, after the 2009 financial crisis.

# 5.2 Package {yfR}

Package **{yfR}** (Perlin, 2023a) is all about downloading stock price data from Yahoo Finance. Unlike other packages, **{yfR}** (Perlin, 2023a) focuses on large batch downloads of structured and clean/tidy data. Its main features are:

---

[5]https://docs.data.nasdaq.com/docs/parameters-2

- Fetches daily/weekly/monthly/annual stock prices/returns from yahoo finance and outputs a dataframe (tibble) in the long format (stacked data);

- A feature called **collections** facilitates download of multiple tickers from a particular market/index. You can, for example, download data for all stocks in the SP500 index with a simple call to `yf_collection_get("SP500")`;

- A session-persistent smart cache system is available by default. This means that the data is saved locally and only missing portions are downloaded, if needed.

- All dates are compared to a benchmark ticker such as SP500 and, whenever an individual asset does not have a sufficient number of dates, the software drops it from the output. This means you can choose to ignore tickers with a high proportion of missing dates.

- A customized function called **yfR::yf_convert_to_wide()** can transform a long `dataframe` into a wide format (tickers as columns), much used in portfolio optimization. The output is a list where each element is a different target variable (prices, returns, volumes).

- Parallel computing with package `furrr` is available, speeding up the data importation process.

As an example of usage, let's download the prices of four stocks in the previous five years using function **yfR::yf_get()** . We choose these companies: Microsoft (MSFT), Google (GOOGL), JP Morgan (JPM) and General Electric (GE).

In the call to function **yfR::yf_get()** , we set arguments `thresh_bad_data = 0.95` and `bench_ticker = '^GSPC'`. These choices make sure that all returned data have at least 95% of valid prices when compared to data from the SP500 index (ticker `'^GSPC'`).

```
# set tickers
tickers <- c('MSFT','GOOGL','JPM','GE')

# set dates
first_date <- Sys.Date()-5*365 # past five years
last_date <- Sys.Date()    # today
thresh_bad_data <- 0.95    # sets percent threshold for bad data
bench_ticker <- '^GSPC'    # set benchmark as SP500
```

```
df_yf <- yfR::yf_get(tickers = tickers,
                     first_date = first_date,
                     last_date = last_date,
                     bench_ticker = bench_ticker,
                     thresh_bad_data = thresh_bad_data)
```

The output of **yfR**::**yf_get()** is an object of type `dataframe`, a table with prices and returns:

```
# print df.tickers
dplyr::glimpse(df_yf)
```

```
R> Rows: 5,028
R> Columns: 11
R> $ ticker                <chr> "GE", "GE", "GE", "GE", "GE~
R> $ ref_date              <date> 2018-12-14, 2018-12-17, 20~
R> $ price_open            <dbl> 42.51486, 42.57491, 43.1153~
R> $ price_high            <dbl> 43.53570, 43.41560, 45.8175~
R> $ price_low             <dbl> 42.03447, 42.09452, 42.9952~
R> $ price_close           <dbl> 42.63496, 42.93521, 43.7158~
R> $ volume                <dbl> 21449181, 21604120, 2444219~
R> $ price_adjusted        <dbl> 41.77919, 42.07340, 42.8383~
R> $ ret_adjusted_prices   <dbl> NA, 0.007042079, 0.01818195~
R> $ ret_closing_prices    <dbl> NA, 0.007042277, 0.01818180~
R> $ cumret_adjusted_prices <dbl> 1.0000000, 1.0070421, 1.025~
```

As expected, we find information about stock prices, daily returns and traded volumes. Notice it also includes column `ticker`, which contains the symbols of the stocks. In the tidy format, each stock has a chunk of data that is pilled in top of each other. Later, in chapter 8, we will use this column to split the data and build summary tables. To inspect the data, create a figure with the prices:

Prices of four stocks



Data from Yahoo Finance
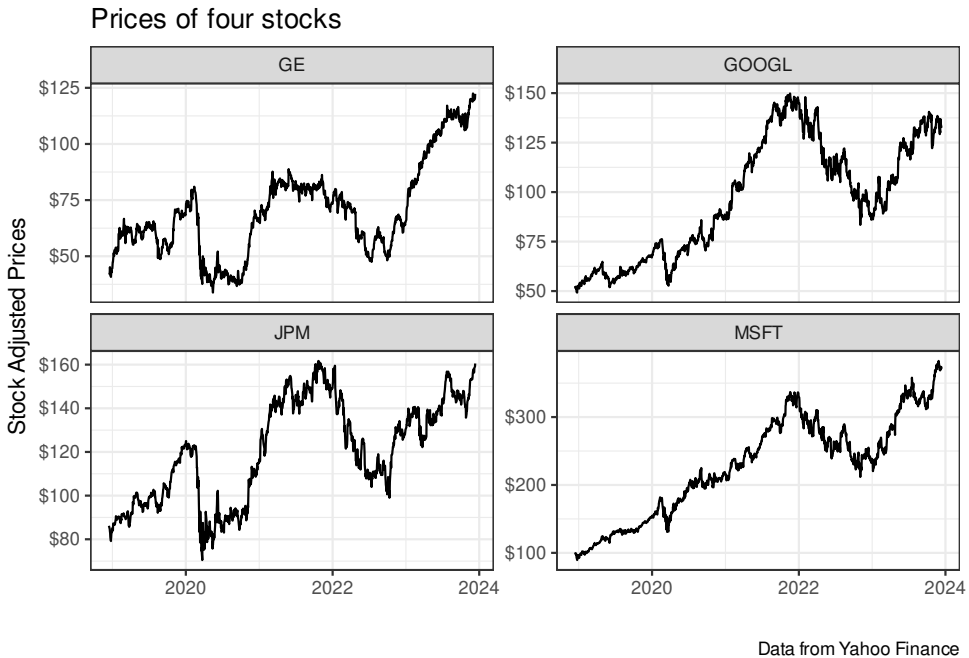
We see that General Eletric (GE) stock was not kind to its investors. Someone that bought the stock at its peak in mid-2016 has found its current value at less than half. Now, when it comes to the GOOGL, JPM and MSFT, we see an overall upward increase in stock prices, and a recent drop. These are profitable and competitive companies in their sectors and not surprisingly, the stock prices surged over the long period of time.

Now, let's look at an example of a large download of stock price data. Here, we will download the current composition of the SP500 index using function **yfR::yf_collection_get()** .

```
# set dates
first_date <- '2020-01-01'
last_date <- '2023-01-01'

df_dow <- yfR::yf_collection_get(
  "DOW",
  first_date,
  last_date
)
```

And now we check the resulting data:

```
dplyr::glimpse(df_dow)
```

```
R> Rows: 22,680
```

```
R> Columns: 11
R> $ ticker                 <chr> "AAPL", "AAPL", "AAPL", "AA~
R> $ ref_date               <date> 2020-01-02, 2020-01-03, 20~
R> $ price_open             <dbl> 74.0600, 74.2875, 73.4475, ~
R> $ price_high             <dbl> 75.1500, 75.1450, 74.9900, ~
R> $ price_low              <dbl> 73.7975, 74.1250, 73.1875, ~
R> $ price_close            <dbl> 75.0875, 74.3575, 74.9500, ~
R> $ volume                 <dbl> 135480400, 146322800, 11838~
R> $ price_adjusted         <dbl> 73.15265, 72.44146, 73.0186~
R> $ ret_adjusted_prices    <dbl> NA, -0.009721989, 0.0079681~
R> $ ret_closing_prices     <dbl> NA, -0.009722036, 0.0079682~
R> $ cumret_adjusted_prices <dbl> 1.0000000, 0.9902780, 0.998~
```

We get a fairly sized table with 22680 rows and 11 columns in object `df_dow`.
Notice how easy it was to get that large volume of data from Yahoo Finance
with a simple call to **yfR**::**yf_collection_get()** .

> Be aware that Yahoo Finance (YF) data for **adjusted prices
> of single stocks** over long periods of time is not trustworthy.
> If you compare it to other data vendors, you'll easily find large
> differences. The issue is that Yahoo Finance does not adjust for
> dividends, only for stock splits. This means that, when looking at
> a price series over a long period of time, there is a downward bias
> in overall return. As a rule of thumb, in a formal research, **never
> use individual stock data from Yahoo Finance**, specially if
> the stock return is important to the research. The exception is for
> financial indexes, such as the SP500, where Yahoo Finance data is
> quite reliable since indexes do not undergo the same adjustments
> as individual stocks.

# 5.3   Package {simfinapi}

SimFin[6] is a reliable repository of financial data from around the world.
It works by gathering, cleaning and organizing data from different stock
exchanges and financial reports. From its own website[7]:

> Our core goal is to make financial data as freely available as
> possible because we believe that having the right tools for in-
> vesting/research shouldn't be the privilege of those that can

---

[6]https://simfin.com/

[7]https://simfin.com/

afford to spend thousands of dollars per year on data.

As of february 2023, the platform offers a generous free plan, with a daily limit of 2000 api calls. This is enough calls for most projects. If you need more calls, the premium version[8] is a fraction of what other data vendors usually request.

Package **{simfinapi}** (Gomolka, 2023) facilitates importing data from the SimFin API. First, it makes sure the requested data exists and only then calls the api. As usual, all api queries are saved locally using package `memoise`. This means that the second time you ask for a particular data about a company/year, the function will load a local copy, and will not call the web api, helping you stay below the API limits.

## 5.3.1    Example 01 - Apple Inc Annual Profit

The first step in using `simfinR` is registering at the SimFin website. Once done, click on Data Access[9]. It should now show an API key such as `'rluwSlN3O4NpyJeBjlxZPspfBBhfJR4o'`. Save it in an R object for later use.

```r
my_api_key <- 'rluwSlN3O4NpyJeBjlxZPspfBBhfJR4o'
```

Be aware that the **API key in `my_api_key` is fake** and will not work for you. You need to get your own to execute the examples.

With the API key in hand, the second step is to find the numerical id of the company of interest. For that, we can find all available companies and their respective ids and ticker with **simfinapi::sfa_get_entities()** .

```r
cache_dir <- fs::path_temp("cache-simfin")
fs::dir_create(cache_dir)


simfinapi::sfa_set_api_key(my_api_key)
simfinapi::sfa_set_cache_dir(cache_dir)

# get info
df_info_companies <- simfinapi::sfa_get_entities()

# check it
glimpse(df_info_companies)
```

---

[8]https://simfin.com/simfin-plus
[9]https://simfin.com/data/access/api

```
R> Rows: 5,082
R> Columns: 2
R> $ simfin_id <int> 854465, 45846, 1253413, 1333027, 367153,~
R> $ ticker    <chr> "1COV.DE", "A", "A18", "A21", "AA", "AAC~
```

Now, based on ticker "AAPL", lets download the download the profit and loss (PL) statement for Apple INC in 2022:

```
ticker <- "AAPL" # ticker of APPLE INC
type_statements <- 'pl' # profit/loss statement
period <- 'fy' # final year
fiscal_year <- 2022


PL_aapl <- simfinapi::sfa_get_statement(
  ticker = ticker,
  statement = type_statements,
  period = period,
  fyear = fiscal_year)


# select columns
PL_aapl <- PL_aapl |>
  dplyr::select(ticker, simfin_id, revenue, net_income)


dplyr::glimpse(PL_aapl)
```

```
R> Rows: 1
R> Columns: 4
R> $ ticker     <chr> "AAPL"
R> $ simfin_id  <int> 111052
R> $ revenue    <dbl> 3.94328e+11
R> $ net_income <dbl> 9.9803e+10
```

Not bad! Financially speaking, the year 2022 was very good for Apple.

## 5.3.2 Example 02 - Annual Net Profit of Many Companies

Package `simfinapi` can also fetch information for many companies in a single call. Let's run another example by selecting three companies: Apple, Google and Amazon, and downloading end of year information:

```
tickers <- c("AAPL", 'GOOG', "AMZN") # ticker of APPLE INC
type_statements <- 'pl' # profit/loss statement
period <- 'fy' # final year
```

```r
fiscal_year <- 2022

PL <- simfinapi::sfa_get_statement(
  ticker = tickers,
  statement = type_statements,
  period = period,
  fyear = fiscal_year)

# select columns
PL <- PL |>
  dplyr::select(ticker, simfin_id, revenue, net_income)

dplyr::glimpse(PL)
```

```
R> Rows: 3
R> Columns: 4
R> $ ticker     <chr> "AAPL", "AMZN", "GOOG"
R> $ simfin_id  <int> 111052, 62747, 18
R> $ revenue    <dbl> 3.94328e+11, 5.13983e+11, 2.82836e+11
R> $ net_income <dbl> 99803000000, -2722000000, 59972000000
```

As you can see, it is fairly straightforward to download financial data for multiple companies using package **{simfinapi}** (Gomolka, 2023).

### 5.3.3   Example 03 - Fetching price data

The simfin project also provides prices of stocks, adjusted for dividends, splits and other corporate events. Have a look at the next example, where we download adjusted stock prices for the previous three companies:

```r
df_prices  <- simfinapi::sfa_get_prices(tickers)

dplyr::glimpse(df_prices)
```

```
R> Rows: 14,496
R> Columns: 12
R> $ simfin_id            <int> 111052, 111052, 111052, ~
R> $ ticker               <chr> "AAPL", "AAPL", "AAPL", ~
R> $ date                 <date> 2000-01-03, 2000-01-04,~
R> $ currency             <chr> "USD", "USD", "USD", "US~
R> $ open                 <dbl> 0.94, 0.97, 0.93, 0.95, ~
R> $ high                 <dbl> 1.00, 0.99, 0.99, 0.96, ~
R> $ low                  <dbl> 0.91, 0.90, 0.92, 0.85, ~
```

```
R> $ close                    <dbl> 1.00, 0.92, 0.93, 0.85, ~
R> $ adj_close                <dbl> 0.85, 0.78, 0.79, 0.72, ~
R> $ volume                   <dbl> 535797336, 512378112, 77~
R> $ dividend                 <dbl> NA, NA, NA, NA, NA, NA, ~
R> $ common_shares_outstanding <dbl> NA, NA, NA, NA, NA, NA, ~
```



Adjusted stock prices for four companies

Price data from simfin
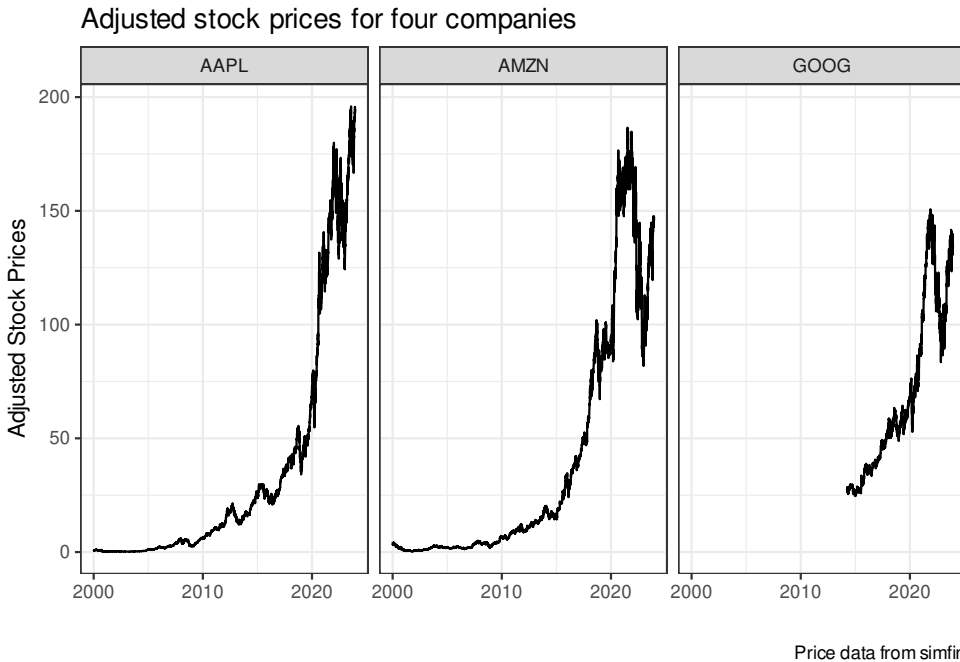
As you can see, the data is comprehensive and should suffice for many different corporate finance research topics.

## 5.4  Package {tidyquant}

Package **{tidyquant}** (Dancho and Vaughan, 2023) provides functions related to financial data acquisition and analysis. It is an ambitious project that offers many solutions in the field of finance. The package includes functions for obtaining financial data from the web, manipulation of such data, and the calculation of performance measures of portfolios.

First, we will obtain price data for Apple stocks (AAPL) using function **tidyquant**::**tq_get()** .

```
# set stock and dates
ticker <- 'AAPL'
first_date <- '2020-01-01'
```

```
last_date <-  Sys.Date()

# get data with tq_get
df_prices <- tidyquant::tq_get(ticker,
                      get = "stock.prices",
                      from = first_date,
                      to = last_date)

dplyr::glimpse(df_prices)
```

```
R> Rows: 994
R> Columns: 8
R> $ symbol   <chr> "AAPL", "AAPL", "AAPL", "AAPL", "AAPL", "~
R> $ date     <date> 2020-01-02, 2020-01-03, 2020-01-06, 2020~
R> $ open     <dbl> 74.0600, 74.2875, 73.4475, 74.9600, 74.29~
R> $ high     <dbl> 75.1500, 75.1450, 74.9900, 75.2250, 76.11~
R> $ low      <dbl> 73.7975, 74.1250, 73.1875, 74.3700, 74.29~
R> $ close    <dbl> 75.0875, 74.3575, 74.9500, 74.5975, 75.79~
R> $ volume   <dbl> 135480400, 146322800, 118387200, 10887200~
R> $ adjusted <dbl> 73.15265, 72.44147, 73.01868, 72.67528, 7~
```

As we can see, except for column names, the price data has a similar format to the one we got with **{yfR}** (Perlin, 2023a). This is not surprising as both share the same origin, Yahoo Finance.

We can also get information about components of an index using function **tidyquant::tq_index()** . The available market indices are:

```
# print available indices
print(tidyquant::tq_index_options())
```

```
R> [1] "DOW"       "DOWGLOBAL" "SP400"       "SP500"
R> [5] "SP600"
```

Let's get information for **"DOWGLOBAL"**.

```
# get components of "DOWJONES"
print(tidyquant::tq_index("DOWGLOBAL"))
```

```
R> Getting holdings for DOWGLOBAL

R> # A tibble: 179 x 8
R>    symbol company            identifier sedol  weight sector
R>    <chr>  <chr>              <chr>      <chr>   <dbl> <chr>
R>  1 QCOM   QUALCOMM INC       747525103  2714~ 0.00828 -
```

```
R>  2 VWS    VESTAS WIND SYSTE~ BN4MYF907  BN4M~ 0.00810 -
R>  3 DBK    DEUTSCHE BANK AG ~ 575035902  5750~ 0.00801 -
R>  4 NKE    NIKE INC  CL B     654106103  2640~ 0.00786 -
R>  5 BBVA   BANCO BILBAO VIZC~ 550190904  5501~ 0.00783 -
R>  6 UCG    UNICREDIT SPA      BYMXPS901  BYMX~ 0.00782 -
R>  7 AVGO   BROADCOM INC       11135F101  BDZ7~ 0.00780 -
R>  8 BA     BOEING CO/THE      097023105  2108~ 0.00763 -
R>  9 SPG    SIMON PROPERTY GR~ 828806109  2812~ 0.00754 -
R> 10 SIE    SIEMENS AG REG     572797900  5727~ 0.00750 -
R> # i 169 more rows
R> # i 2 more variables: shares_held <dbl>,
R> #   local_currency <chr>
```

We only looked into a few functions from the package. **{tidyquant}** (Dancho and Vaughan, 2023) also offers solutions for the usual financial manipulations, such as calculating returns and functions for portfolio analytics. You can find more details about this package in its website[10].

## 5.5  Other Packages

In CRAN, you'll find many more packages for importing financial datasets in R. In this section, we focused on packages, which are free and easy to use. Interface with commercial data sources is also possible. Several companies provide APIs for serving data to their clients. Packages such as **{Rblpapi}** (Armstrong et al., 2022) for Bloomberg, **{IBrokers}** (**?**) for Interactive Brokers can make R communicate with commercial platforms. If the company you use is not available, check the list of packages in CRAN[11]. It is very likely you'll find what you need.

## 5.6  Accessing Data from Web Pages (*web-scraping*)

Packages from previous section facilitates data importation over the internet. However, in many cases, the information of interest is not available through a package, but on a web page. Fortunately, we can use R to read the data and import the desired information into an R session. The main advantage is that, every time we execute the code, we get the same content available in the website.

---

[10]https://business-science.github.io/tidyquant/

[11]https://cran.r-project.org/

The process of extracting information from web pages is called *webscraping*. Depending on the structure and technology used on the internet page, importing its content can be as trivial as a single line in R or a complex process, taking hundreds of lines of code.

## 5.6.1   Scraping the Components of the SP500 Index from Wikipedia

As an example of webscraping, let's retrieve tabular information about the SP500 index from Wikipedia. In its website, Wikipedia offers a section[12] about the components of the SP500 index. This information is presented in a tabular format, Figure 5.1.



Figure 5.1: Mirror of Wikipedia page on SP500 components

The information on this web page is constantly updated, and we can use it to import information about the stocks belonging to the SP500 index. Before delving into the R code, we need to understand how a webpage works. Briefly, a webpage is nothing more than a lengthy code interpreted by your browser. A numerical value or text presented on the website can usually be found within the code. This code has a particular tree-like structure with branches and classes. Moreover, every element of a webpage has an address, called *xpath*. In chrome and firefox browsers, you can see the actual code of a webpage by using the mouse to right-click any part of the webpage and selecting *View page source*.

The first step in webscraping is finding out the location of the information you need. In Chrome, you can do that by right-clicking in the specific location of the number/text on the website and selecting *inspect*. This will

---

[12]https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

open an extra window in the browser. Once you do that, right-click in the selection and click in *copy* and *copy xpath*. In Figure 5.2, we see a mirror of what you should be seeing in your browser.
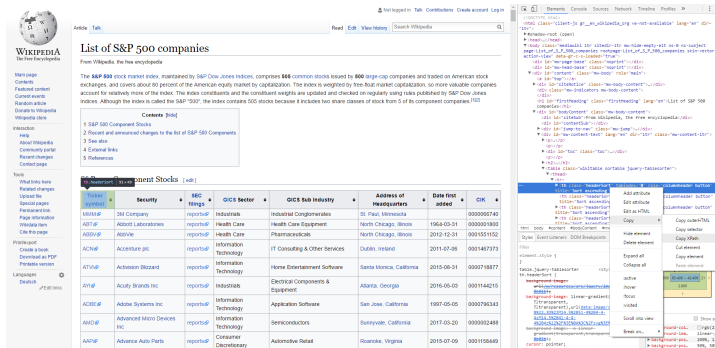


Figure 5.2: Finding xpath from website

Here, the copied *xpath* is:

```
'//*[@id="mw-content-text"]/table[1]/thead/tr/th[2]'
```

This is the address of the header of the table. For the whole content of the table, including header, rows, and columns, we need to set an upper level of the HTML tree. This is equivalent to address `//*[@id="MW-content-text"]/table[1]`.

Now that we have the location of what we want, let's load package **{rvest}** (Wickham, 2022) and use functions **rvest::read_html()** , **rvest::html_nodes()** and **rvest::html_table()** to import the desired table into R:

```r
# set url and xpath
my_url <- paste0('https://en.wikipedia.org/wiki/',
                 'List_of_S%26P_500_companies')
my_xpath <- '//*[@id="mw-content-text"]/div/table[1]'

# get nodes from html
out_nodes <- rvest::html_nodes(rvest::read_html(my_url),
                        xpath = my_xpath)

# get table from nodes (each element in
# list is a table)
df_SP500_comp <- rvest::html_table(out_nodes)

# isolate it
```

```
df_SP500_comp <- df_SP500_comp[[1]]

# change column names (remove space)
names(df_SP500_comp) <- make.names(names(df_SP500_comp))

# print it
dplyr::glimpse(df_SP500_comp)
```

```
R> Rows: 503
R> Columns: 8
R> $ Symbol               <chr> "MMM", "AOS", "ABT", "ABBV",~
R> $ Security             <chr> "3M", "A. O. Smith", "Abbott~
R> $ GICS.Sector          <chr> "Industrials", "Industrials"~
R> $ GICS.Sub.Industry    <chr> "Industrial Conglomerates", ~
R> $ Headquarters.Location <chr> "Saint Paul, Minnesota", "Mi~
R> $ Date.added           <chr> "1957-03-04", "2017-07-26", ~
R> $ CIK                  <int> 66740, 91142, 1800, 1551152,~
R> $ Founded              <chr> "1902", "1916", "1888", "201~
```

Object `df_SP500_comp` contains a mirror of the data from the Wikipedia website. The names of the columns require some work, but the raw data is intact and could be further used in a script.

Learning *webscraping* techniques can give you access to an immense amount of information available on the web. However, each scenario of *webscraping* is particular. It is not always the case you can import data directly and easily as in previous example.

Another problem is that the webscrapping code depends on the structure of the website. Any simple change in the *html* structure and your code will fail. You should be aware that maintaining a *webscraping* code can demand significant time and effort from the developer. If possible, you should always check for alternative sources of the same information.

Readers interested in learning more about this topic should study the functionalities of packages **{XML}** (Temple Lang, 2023) and **{RSelenium}** (Harrison, 2022)

# 5.7  Exercises

---

01 - Using the `yfR` package, download daily data of the Facebook stock (META) from *Yahoo Finance* for the period between 2019 and 2023. What

is the lowest **unadjusted closing price (column `price.close`)** in the analyzed period?

---

02 - If you have not already done so, create a profile on the Quandl website[13] and download the arabica coffee price data in the CEPEA database (Center for Advanced Studies in Applied Economics) ) between 2010-01-01 and 2020-12-31. What is the value of the most recent price?

---

03 - Use function `simfinapi::sfa_get_entities()` to import data about all available companies in Simfin. How many companies do you find? (see function `dplyr::n_distinct()`).

---

04 - With package `simfinapi`, download the PL (profit/loss) statement for FY (final year) data for TESLA (ticker = "TSLA") for year 2022. What is the latest Profit/Loss of the company for that particular year?

---

05 - Using function `tidyquant::tq_index`, download the current composition of index DOWGLOBAL. What is the company with the highest percentage in the composition of the index?

Be aware that the answer is time-dependent and the reported result might be different from what you actually got in your R session.

---

06 - Using again the `yfR` package, download financial data between 2019-01-01 and 2020-01-01 for the following tickers:

- AAPL: Apple Inc
- BAC: Bank of America Corporation
- GE: General Electric Company
- TSLA: Tesla, Inc.
- SNAP: Snap Inc.

Using the **adjusted closing price** column, what company provided higher return to the stock holder during the analyzed period?

---

[13]https://www.quandl.com/

Tip: this is an advanced exercise that will require some coding. To solve it, check out function `split` to split the dataframe of price data and `lapply` to map a function to each dataframe.

# Chapter 6

# Dataframes and Other Objects

ⓘ | You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 6.1  Dataframes

ⓘ | You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.1   Creating `dataframes`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.2   Inspecting a Dataframe

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.3   The *pipeline* Operators (`|>` and `|>`)

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.4  Accessing Columns

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.5  Modifying a `dataframe`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.6  Filtering rows of a `dataframe`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.7  Sorting a `dataframe`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars.  Purchasing this book is a great way of supporting this and other projects of the author.  If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com).  The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.8  Combining and Aggregating `dataframes`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars.  Purchasing this book is a great way of supporting this and other projects of the author.  If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com).  The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.1.9  Extensions of the `dataframe` Class

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars.  Purchasing this book is a great way of supporting this and other projects of the author.  If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com).  The book is a lifelong project and I'll keep improving it based on the received feedback.

## 6.1.10 Other Useful Functions for Handling `dataframes`

> ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 6.2 Lists

> ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 6.2.1 Creating `lists`

> ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.2.2 Accessing the Elements of a `list`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.2.3 Adding and Removing Elements from a `list`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.2.4 Processing the Elements of a `list`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.2.5 Other Useful Functions

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 6.3 Matrices

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.3.1 Selecting Elements from a `matrix`

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 6.3.2   Other Useful Functions

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 6.4   Exercises

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# Basic Object Classes

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.1 Numeric Objects

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.1   Creating and Manipulating `numeric` Objects

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.2   Creating a `numeric` Sequence

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.3   Creating Vectors with Repeated Elements

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.4 Creating Vectors with Random Numbers

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.5 Accessing the Elements of a `numeric` Vector

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.6 Modifying and Removing Elements of a `numeric` Vector

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.7   Creating Groups

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.1.8   Other Useful Functions

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.2   Character Objects

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.1 Creating a Simple `character` Object

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.2 Creating Structured `character` Objects

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.3 `character` Constants

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.2.4   Selecting Pieces of a Text Object

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.2.5   Finding and Replacing Characters of a Text

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.2.6   Splitting Text

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.7 Finding the Number of Characters in a Text

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.8 Generating Combinations of Text

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.9 Encoding of `character` Objects

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.2.10   Other Useful Functions

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.3   Factor Objects

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.3.1   Creating `factors`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.3.2 Modifying `factors`

> ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.3.3 Converting `factors` to Other Classes

> ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.3.4 Creating Contingency Tables

> ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.3.5   Other Useful Functions

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.4   Logical Objects

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.4.1   Creating `logical` Objects

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 7.5 Date and Time

 You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.5.1 Creating Simple Dates

 You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.5.2 Creating a Sequence of `Dates`

 You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.5.3  Operations with `Dates`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.5.4  Dealing with Time

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.5.5  Customizing the Format of Dates and Times

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.5.6 Extracting Elements of a Date

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.5.7 Find the Current Date and Time

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.5.8 Other Useful Functions

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 7.6   Missing Data - `NA` (*Not available*)

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.6.1   Defining `NA` Values

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.6.2   Finding and Replacing `NA`

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 7.6.3 Other Useful Functions

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 7.7 Exercises

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# Chapter 8

# Programming and Data Analysis

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.1   R Functions

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.2   Using `for` Loops

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.3   Conditional Statements (`if`, `else`, `switch`)

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.4   Functional Programming

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.4.1 Using lapply()

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.4.2 Using sapply()

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.4.3 Using tapply()

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.4.4 Using mapply()

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.4.5 Using apply()

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.4.6 Using by()

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 8.5 Using package {purrr}

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.5.1 Function map()

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.5.2 Function safely()

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.5.3   Function pmap()

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.6   Data   Manipulation   with   Package {dplyr}

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.6.1   Group Operations

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 8.6.2 Complex Group Operations

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 8.7 Exercises

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# Cleaning and Structuring Data

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 9.1 The Format of a `dataframe`

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 9.1.1 Converting a `dataframe` Structure (long and wide)

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 9.2 Converting `lists` into `dataframes`

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 9.3 Removing Outliers

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 9.3.1 Treating Outliers in `dataframes`

## 9.4 Inflation and Price Data

## 9.5 Modifying Time Frequency and Aggregating Data

## 9.6   Exercises

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 10

# Data Visualization with {ggplot2}

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.1 Principles for Data visualization

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.2 The {ggplot2} Package

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.3 Using Graphics Windows

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.4 Creating Figures with Function ggplot()

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.5 Data Visualization for Groups

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 10.5.1 The US Yield Curve

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.6 Using Themes

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.7 Creating Panels with `facet_wrap`

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.8 Using the Pipeline

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.9 Creating Statistical Graphics

> **i** You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 10.9.1   Creating Histograms

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 10.9.2   Creating *boxplot* Figures

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 10.9.3   Creating *QQ* Plots

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.10   Saving Graphics to a File

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 10.11   Exercises

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

Chapter **11**

# Financial Econometrics with R

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.1 Linear Models (OLS)

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.1.1   Simulating a Linear Model

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.1.2   Estimating a Linear Model

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.1.3   Statistical Inference in Linear Models

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 11.2 Generalized Linear Models (GLM)

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.2.1 Simulating a GLM Model

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.2.2 Estimating a GLM Model

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 11.3   Panel Data Models

ⓘ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.3.1   Simulating Panel Data Models

ⓘ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.3.2   Estimating Panel Data Models

ⓘ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 11.4   Arima Models

ℹ   You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.4.1   Simulating Arima Models

ℹ   You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.4.2   Estimating Arima Models

ℹ   You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.4.3 Forecasting Arima Models

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.5 GARCH Models

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.5.1 Simulating Garch Models

ℹ️ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.5.2 Estimating Garch Models

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.5.3 Forecasting Garch Models

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 11.6 Dealing with Several Models

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.6.1   Using tapply() and sapply()

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.6.2   Using by()

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 11.6.3   Using dplyr::group_by()

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# 11.7 Exercises

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

Chapter 12

# Reporting Results

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 12.1 Reporting Tables

ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 12.2   Reporting Models

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 12.3   Creating Reports with *RMarkdown*

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 12.4   Exercises

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# Optimizing Code

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 13.1 Optimizing your Programming Time

> ℹ You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 13.2   Optimizing Code Speed

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 13.2.1   Profiling Code

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 13.2.2   Simple Strategies to Improve Code Speed

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 13.2.2.1 Use Vector Operations

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 13.2.2.2 Repetitive binding of `dataframes`

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 13.2.3 Using C++ code (package {Rcpp})

> You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

### 13.2.4   Using cache (package {memoise})

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

#### 13.2.4.1   Using parallel processing (package {furrr})

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

## 13.3   Exercises

You reached the end of the online version of *Analyzing Financial and Economic Data with R*. The full content of the book can be acquired at Amazon for less than ten dollars. Purchasing this book is a great way of supporting this and other projects of the author. If you are satisfied with the content, please leave your feedback at Amazon or by email (marceloperlin@gmail.com). The book is a lifelong project and I'll keep improving it based on the received feedback.

# Bibliography

Armstrong, W., Eddelbuettel, D., and Laing, J. (2022). *Rblpapi: R Interface to Bloomberg.* R package version 0.3.14, https://github.com/Rblp/Rblpapi.

Bache, S. M. and Wickham, H. (2022). *magrittr: A Forward-Pipe Operator for R.* R package version 2.0.3, https://github.com/tidyverse/magrittr.

Chang, W., Cheng, J., Allaire, J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., and Borges, B. (2021). *shiny: Web Application Framework for R.* R package version 1.6.0.

Dancho, M. and Vaughan, D. (2023). *tidyquant: Tidy Quantitative Financial Analysis.* R package version 1.0.7.

Dragulescu, A. and Arendt, C. (2020). *xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files.* R package version 0.6.5.

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Ucar, I., Bates, D., and Chambers, J. (2023). *Rcpp: Seamless R and C++ Integration.* R package version 1.0.11, https://dirk.eddelbuettel.com/code/rcpp.html.

Garmonsway, D. (2023). *tidyxl: Read Untidy Excel Files.* R package version 1.0.9, https://nacnudus.github.io/tidyxl/.

Gomolka, M. (2023). *simfinapi: Accessing SimFin Data.* R package version 0.2.4.

Harrison, J. (2022). *RSelenium: R Bindings for Selenium WebDriver.* R package version 1.7.9.

Hester, J., Wickham, H., and Csárdi, G. (2023). *fs: Cross-Platform File System Operations Based on libuv.* R package version 1.6.3.

Johnston, M. and Robinson, D. (2023). *gutenbergr: Download and Process Public Domain Works from Project Gutenberg.* R package version 0.2.4, https://github.com/ropensci/gutenbergr.

Klik, M. (2022). *fst: Lightning Fast Serialization of Data Frames.* R package version 0.9.8.

Mirai Solutions GmbH (2023). *XLConnect: Excel Connector for R.* R package version 1.0.7.

Müller, K., Wickham, H., James, D. A., and Falcon, S. (2023). *RSQLite: SQLite Interface for R.* R package version 2.3.4.

Ooms, J. (2023). *writexl: Export Data Frames to Excel xlsx Format.* R package version 1.4.2.

Perlin, M. (2023a). *yfR: Downloads and Organizes Financial Data from Yahoo Finance.* R package version 1.1.0.

Perlin, M. S. (2023b). *afedR3: Data and Functions for third edition of Book "Analyzing Financial and Economical Data with R".* R package version 0.1.

Perlin, M. S. (2023c). *GetQuandlData: Fast and Cached Import of Data from Quandl Using the json API.* R package version 1.0.0.

R Core Team (2023a). *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, …* R package version 0.8-86.

R Core Team (2023b). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.

Raymond McTaggart, Gergely Daroczi, and Clement Leung (2021). *Quandl: API Wrapper for Quandl.com.* R package version 2.11.0.

Ryan, J. A. and Ulrich, J. M. (2023). *quantmod: Quantitative Financial Modelling Framework.* R package version 0.4.25.

Teetor, P. (2011). *R cookbook.* " O'Reilly Media, Inc.".

Temple Lang, D. (2023). *XML: Tools for Parsing and Generating XML Within R and S-Plus.* R package version 3.99-0.16.

Vaughan, D. and Dancho, M. (2022). *furrr: Apply Mapping Functions in Parallel using Futures.* R package version 0.3.1, https://furrr.futureverse.org/.

Venables, W. N., Smith, D. M., Team, R. D. C., et al. (2004). An introduction to r.

Wickham, H. (2019). *Advanced r.* CRC press.

Wickham, H. (2022). *rvest: Easily Harvest (Scrape) Web Pages.* R package version 1.0.3.

Wickham, H. (2023). *tidyverse: Easily Install and Load the Tidyverse.* R package version 2.0.0, https://github.com/tidyverse/tidyverse.

Wickham, H. and Bryan, J. (2023). *readxl: Read Excel Files.* R package version 1.4.3.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., and Dunnington, D. (2023a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.* R package version 3.4.4, https://github.com/tidyverse/ggplot2.

Wickham, H., François, R., Henry, L., Müller, K., and Vaughan, D. (2023b). *dplyr: A Grammar of Data Manipulation.* R package version 1.1.4.

Wickham, H., Hester, J., and Bryan, J. (2023c). *readr: Read Rectangular Text Data.* R package version 2.1.4.

Wickham, H., Hester, J., Chang, W., and Bryan, J. (2022). *devtools: Tools to Make Developing R Packages Easier.* R package version 2.4.5.

Wickham, H., Hester, J., Chang, W., Müller, K., and Cook, D. (2021). *memoise: Memoisation of Functions.* R package version 2.0.1.

# Index