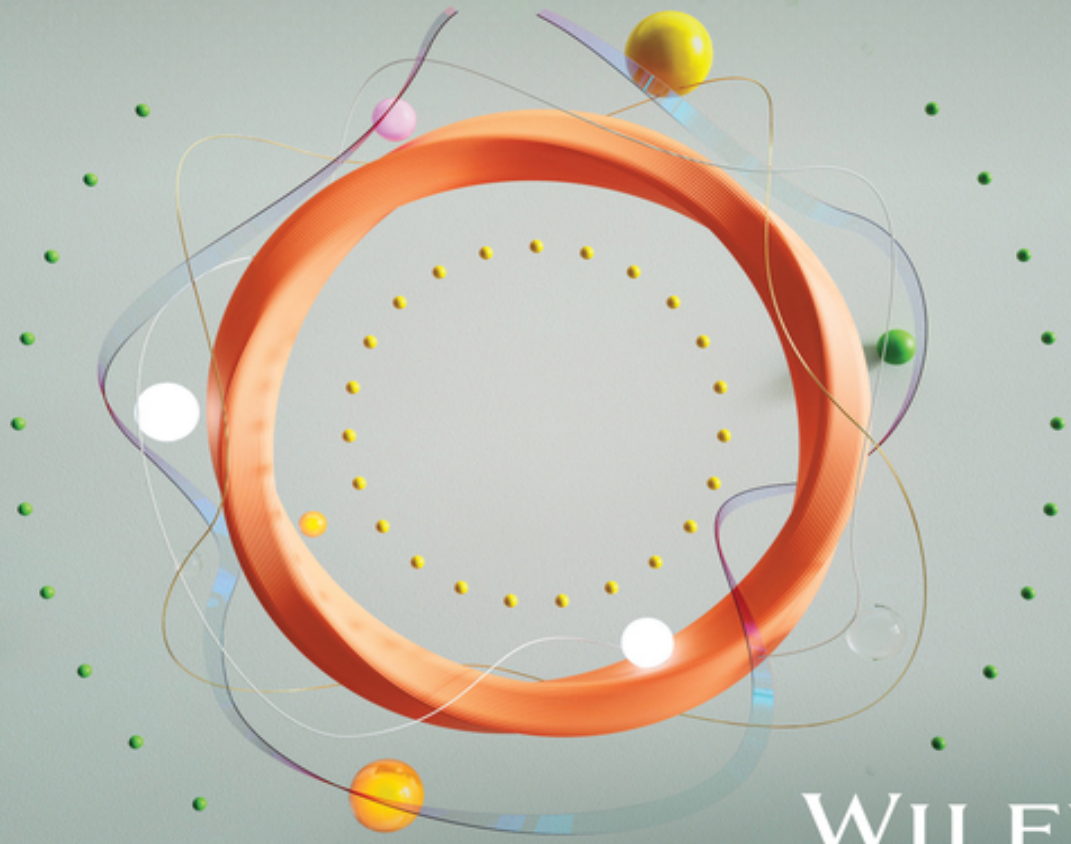
Data Science Fundamentals with R, Python, and Open Data

Marco Cremonini



WILEY

Data Science Fundamentals with R, Python, and Open Data

Data Science Fundamentals with R, Python, and Open Data

Marco Cremonini
University of Milan
Italy

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data Applied for:

Hardback ISBN: 9781394213245

Cover Design: Wiley
Cover Image: © Andriy Onufriyenko/Getty Images

Set in 9.5/12.5pt STIXTwoText by Straive, Chennai, India

Contents

Preface

Two questions come along with every new text that aims to teach someone something. The first is, Who is it addressed to? and the second is, Why does it have precisely those contents, organized in that way? These two questions, for this text, have perhaps even greater relevance than they usually do, because for both, the answer is unconventional (or at least not entirely conventional) and to some, it may seem surprising. It shouldn't be, or even better, if the answers will make the surprise a pleasant surprise.

Let's start with the first question: Who is the target of a text that introduces the fundamentals of two programming languages, R and Python, for the discipline called data science? Those who study to become data scientists, computer scientists, or computer engineers, it seems obvious, right? Instead, it is not so. For sure, future data scientists, computer scientists, and computer engineers could find this text useful. However, the real recipients should be others, simply all the others, the non-specialists, those who do not work or study to make IT or data science their main profession. Those who study to become or already are sociologists, political scientists, economists, psychologists, marketing or human resource management experts, and those aiming to have a career in business management and in managing global supply chains and distribution networks. Also, those studying to be biologists, chemists, geologists, climatologists, or even physicians. Then there are law students, human rights activists, experts of traditional and social media, memes and social networks, linguists, archaeologists, and paleontologists (I'm not joking, there really are fabulous data science works applied to linguistics, archeology, and paleontology). Certainly, in this roundup, I have forgotten many who deserved to be mentioned like the others. Don't feel left out. The artists I forgot! There are contaminations between art, data science, and data visualization of incredible interest. Art absorbs and re-elaborates, and in a certain way, this is also what data science does: it absorbs and re-elaborates. Finally, there are also all those who just don't know yet what they want to be; they will figure it out along the way, and having certain tools can come in handy in many cases.

Everyone can successfully learn the fundamentals of data science and the use of these computational tools, even with a few basic computer skills, with some efforts and time, of course, necessary but reasonable. Everyone could find opportunities for application in all, or almost all, existing professions, sciences, humanities, and cultural fields. And above all, without the need to take on the role of computer scientist or data scientist when you already have other roles to take on, which rightly demand time and dedication.

Therefore, the fact of not considering computer scientists and data scientists as the principal recipients of this book is not to diminish their role for non-existent reasons, but because for them there is no need to explain why a book that presents programming languages for data science has, at least in theory, something to do with what they typically do.

It is to the much wider audience of non-specialists that the exhortation to learn the fundamentals of data science should be addressed to, explaining that they do not have to transform themselves into computer scientists to be able to do so (or even worse, into geeks), which, with excellent reasons that are difficult to dispute, have no intention to do. It doesn't matter if they have always been convinced to be "unfit for computer stuff," and that, frankly, the rhetoric of past twenty years about "digital natives," "being a coder," or "joining the digital revolution" sounds just annoying. None of this should matter, time to move on. How? Everyone should look at what digital skills and technologies would be useful for their own discipline and do the training for those goals. Do you want to be a computer scientist or a data scientist? Well, do it; there is no shortage of possibilities. Do you want to be an economist, a biologist, or a marketing expert? Very well, do it, but you must not be cut off from adequate training on digital methodologies and tools from which you will benefit, as much as you are not cut off from a legal, statistical, historical, or sociological training if this knowledge is part of the skills needed for your profession or education. What is the objection that is usually made? No one can know everything, and generalists end up knowing a little of everything and nothing adequately. It's as true as clichés are, but that's not what we're talking about. A doctor who acquires statistical or legal training is no less a doctor for this; on the contrary, in many cases she/he is able to carry out the medical profession in a better way. No one reproaches an economist who becomes an expert in statistical analysis that she/he should have taken a degree in statistics. And soon (indeed already now), to the same economist who will become an expert in machine learning techniques for classification problems for fintech projects, no one, hopefully, will reproach that as an economist she/he should leave those skills to computer scientists. Like it or not, computer skills are spreading and will do so more and more among non-computer scientists, it's a matter of base rate, notoriously easy to be misinterpreted, as all students who have taken an introductory course in statistics know.

Let's consider the second question: Why this text presents two languages instead of just one as it is usually done? Isn't it enough to learn just one? Which is better? A friend of mine told me he's heard that Python is famous, the other one he has never heard of. Come on, seriously two? It's a miracle if I learn half of just one! Stop. That's enough.

It's not a competition or a beauty contest between programming languages, and not even a question of cheering, as with sports teams, where you have to choose one, none is admissible, but you can't root for two. R and Python are tools, in some ways complex, not necessarily complicated, professional, but also within anyone's reach. Above all, they are the result of the continuous work of many people; they are evolving objects and are extraordinary teaching aids for those who want to learn. Speaking of evolution, a recent and interesting one is the increasingly frequent convergence between the two languages presented in this text. Convergence means the possibility of coordinated, alternating, and complementary use: Complement the benefits of both, exploit what is innovative in one and what the other has, and above all, the real didactic value, learning not to be afraid to change technology, because much of what you learned with one will be found and will be useful with the other. There is another reason, this one is more specific. It is true that Python is so famous that almost everyone has heard its name while only relatively few know R, except that practically everyone involved in data science knows it and most of them uses it, and that's for a pretty simple reason: It's a great tool with a large community of people who have been contributing new features for many years. What about Python? Python is used by millions of people, mainly to make web services, so it has enormous application possibilities. A part of Python has specialized in data science and is growing rapidly, taking advantage of the ease of extension to dynamic and web-oriented applications. One last piece of information: Learning the first programming language could look difficult. The learning curve, so-called how fast you learn, is steep at first, you struggle

at the very beginning, but after a while it softens, and you run. This is for the first one. Same ramp to climb with the second one too? Not at all. Attempting an estimate, I would say that just one-third of the effort is needed to learn the second, a bargain that probably few are aware of. Therefore, let's do both of them.

One last comment because one could certainly think that this discussion is only valid in theory, putting it into practice is quite another thing. Over the years I have required hundreds of social science students to learn the fundamentals of both R and Python for data science and I can tell you that it is true that most of them struggled initially, some complained more or less aloud that they were unfit, then they learned very quickly and ended up demonstrating that it was possible for them to acquire excellent computational skills without having to transform into computer scientists or data scientists (to tell the truth, someone transformed into one, but that's fine too), without possessing nonexistent digital native geniuses, without having to be anything other than what they study for, future experts in social sciences, management, human resources, or economics, and what is true for them is certainly true for everyone. This is the pleasant surprise.

Milan, Italy
2023

Marco Cremonini

About the Companion Website

This book is accompanied by student companion website.

www.wiley.com/go/DSFRPythonOpenData 

The student website includes:

- MCQs
- Software

Introduction

This text introduces the fundamentals of data science using two main programming languages and open-source technologies : R and Python. These are accompanied by the respective application contexts formed by tools to support coding scripts, i.e. logical sequences of instructions with the aim to produce certain results or functionalities. The tools can be of the command line interface (CLI) type, which are consoles to be used with textual commands, and integrated development environment (IDE), which are of interactive type to support the use of languages. Other elements that make up the application context are the supplementary libraries that contain the additional functions in addition to the basic ones coming with the language, package managers for the automated management of the download and installation of new libraries, online documentation, cheat sheets, tutorials, and online forums of discussion and help for users. This context, formed by a language, tools, additional features, discussions between users, and online documentation produced by developers, is what we mean when we say "R" and "Python," not the simple programming language tool, which by itself would be very little. It is like talking only about the engine when instead you want to explain how to drive a car on busy roads.

R and Python, together and with the meaning just described, represent the knowledge to start approaching data science, carry out the first simple steps, complete the educational examples, get acquainted with real data, consider more advanced features, familiarize oneself with other real data, experiment with particular cases, analyze the logic behind mechanisms, gain experience with more complex real data, analyze online discussions on exceptional cases, look for data sources in the world of open data, think about the results to be obtained, even more sources of data now to put together, familiarize yourself with different data formats, with large datasets, with datasets that will drive you crazy before obtaining a workable version, and finally be ready to move to other technologies, other applications, uses, types of results, projects of ever-increasing complexity. This is the journey that starts here, and as discussed in the preface, it is within the reach of anyone who puts some effort and time into it. A single book, of course, cannot contain everything, but it can help to start, proceed in the right direction, and accompany for a while.

With this text, we will start from the elementary steps to gain speed quickly. We will use simplified teaching examples, but also immediately familiarize ourselves with the type of data that exists in reality, rather than in the unreality of the teaching examples. We will finish by addressing some elaborate examples, in which even the inconsistencies and errors that are part of daily reality will emerge, requiring us to find solutions.

Approach

It often happens that students dealing with these contents, especially the younger ones, initially find it difficult to figure out the right way to approach their studies in order to learn effectively. One of the main causes of this difficulty lies in the fact that many are accustomed to the idea that the goal of learning is to never make mistakes. This is not surprising, indeed, since it's the criteria adopted by many exams, the more mistakes, the lower the grade. This is not the place to discuss the effectiveness of exam methodologies or teaching philosophies; we are pragmatists, and the goal is to learn R and Python, computational logic, and everything that revolves around it. But it is precisely from a wholly pragmatic perspective that the problem of the inadequacy of the approach that seeks to minimize errors arises, and this for at least two good reasons. The first is that inevitably the goal of never making mistakes leads to mnemonic study. Sequences of passages, names, formulas, sentences, and specific cases are memorized, and the variability of the examples considered is reduced, tending toward schematism. The second reason is simply that trying to never fail is exactly the opposite of what it takes to effectively learn R and Python and any digital technology.

Learning computational skills for the data science necessarily requires a hands-on approach. This involves carrying out many practical exercises, meticulously redoing those proposed by the text, but also varying them, introducing modifications, and replicating them with different data. All those of the didactic examples can obviously be modified, but also all those with open data can easily be varied. Instead of certain information, others could be used, and instead of a certain result, a slightly different one could be sought, or different data made available by the same source could be tried. Proceeding methodically (being methodical, meticulous, and patient are fundamental traits for effective learning) is the way to go. Returning to the methodological doubts that often afflict students when they start, the following golden rule applies, which must necessarily be emphasized because it is of fundamental importance: exercises are used to make mistakes, an exercise without errors is useless.

Open Data

The use of open data, right from the first examples and to a much greater extent than examples with simplified educational datasets, is one of the characteristics, perhaps the main one, of this text. The datasets taken from open data are 26, sourced from the United States and other countries, large international organizations (the World Bank and the United Nations), as well as charities and independent research institutes, gender discrimination observatories, and government agencies for air traffic control, energy production and consumption, pollutant emissions, and other environmental information. This also includes data made available by cities like Milan, Berlin, and New York City. This selection is just a drop in the sea of open data available and constantly growing in terms of quantity and quality.

Using open data to the extent it has been done in this text is a precise choice that certainly imposes an additional effort on those who undertake the learning path, a choice based both on personal experience in teaching the fundamentals of data science to students of social and political sciences (every year I have increasingly anticipated the use of open data), and on the fundamental drawback of carrying out examples and exercises mainly with didactic cases, which are inevitably unreal and unrealistic. Of course, the didactic cases, also present in this text, are perfectly fit for showing a

specific functionality, an effect or behavior of the computational tool. As mentioned before, though, the issue at stake is about learning to drive in urban traffic, not just understanding some engine mechanics, and at the end the only way to do that is … driving in traffic, there's no alternative. For us it is the same, anyone who works with data knows that one of the fundamental skills is to prepare the data for analysis (first there would be that of finding the data) and also that this task can easily be the most time- and effort-demanding part of the whole job. Studying mainly with simplified teaching examples erases this fundamental part of knowledge and experience, for this reason, they are always unreal and unrealistic, however you try to fix them. There is no alternative to putting your hands and banging your head on real data, handling datasets even of hundreds of thousands or millions of rows (the largest one we use in this text has more than 500 000 rows, the data of all US domestic flights of January 2022) with their errors, explanations that must be read and sometimes misinterpreted, even with cases where data was recorded inconsistently (we will see one of this kind quite amusing). Familiarity with real data should be achieved as soon as possible, to figure out their typical characteristics and the fact that behind data there are organizations made up of people, and it is thanks to them if we can extract new information and knowledge. You need to arm yourself with patience and untangle, one step at a time, each knot. This is part of the fundamentals to learn.

What You Don't Learn

One book alone can't cover everything; we've already said it and it's obvious. However, the point to decide is what to leave out. One possibility is that the author tries to discuss as many different topics as she/he can think of. This is the encyclopedic model, popular but not very compatible with a reasonably limited number of pages. It is no coincidence that the most famous of the encyclopedias have dozens of ponderous volumes. The short version of the encyclopedic model is a "synthesis," i.e. a reasonably short overview that is necessarily not very thorough and has to simplify complex topics. Many educational books choose this form, which has the advantage of the breadth of topics combined with a fair amount of simplification.

This book has a hybrid form, from this point of view. It is broader than the standard because it includes two languages instead of one, but it doesn't have the form of synthesis because it focuses on a certain specific type of data and functionality: data frames, with the final addition of lists/dictionaries, transformation and pivoting operations, group indexing, aggregation, advanced transformations and data frame joins, and on these issues, it goes into the details. Basically, it offers the essential toolbox for data science.

What's left out? Very much, indeed. The techniques and tools for data visualization, descriptive and predictive models, including machine learning techniques, obviously the statistical analysis part (although this is traditionally an autonomous part), technologies for "Big Data," i.e. distributed, scalable software infrastructures capable of managing not only a lot of data but above all data streams, i.e. real-time data flows, and the many web-oriented extensions, starting from data collection techniques from websites up to integration with dynamic dashboards and web services, are not included. Again, there are specialized standards, such as those for climate data, financial data, biomedical data, and coding used by some of the large international institutions that are not treated. The list could go on.

This additional knowledge, which is part of data science, deserves to be learned. For this, you need the fundamentals that this book presents. Once equipped with them, it's the personal interests

and the cultural and professional path of each one to play the main role, driving in a certain direction or in another. But again, once it has been verified firsthand that it is possible, regardless of one's background, to profitably acquire the fundamentals of the discipline with R and Python, any further insights and developments can be tackled, in exactly the same way, with the same approach and spirit used to learn the fundamentals.

1

Open-Source Tools for Data Science

1.1 R Language and RStudio

In this first section, we introduce the main tools for the R environment: the ***R language*** and the ***RStudio IDE*** (interactive development environment). The first is an open-source programming language developed by the community, specifically for statistical analysis and data science; the second is an open-source development tool produced by *Posit* (www.posit.com), formerly called RStudio, representing the standard IDE for R-based data science projects. Posit offers a freeware version of RStudio called *RStudio Desktop* that fully supports all features for R development; it has been used (v. 2022.07.2) in the preparation of all the R code presented in this book. Commercial versions of RStudio add supporting features typical of managing production software in corporate environments. An alternative to RStudio Desktop is ***RStudio Cloud***, the same IDE offered as a service on a cloud premise. Graphically and functionally, the cloud version is exactly the same as the desktop one; however, its free usage has limitations.

The official distribution of the R language and the RStudio IDE are just the starting points though. This is what distinguishes an open-source technology from a proprietary one. With an open-source technology actively developed by a large online community, as is the case for R, the official distribution provides the basic functionality and, on top of that, layers of additional, advanced, or specialistic features could be stacked, all of them developed by the open-source community. Therefore, it is a constantly evolving environment, not a commercial product subject to the typical life cycle mostly mandated by corporate marketing. What is better, an open-source or a proprietary tool? This is an ill-posed question, mostly irrelevant in generic terms because the only reasonable answer is, "It depends." The point is that they are different in a number of fundamental ways.

With R, we will use many features provided by additional packages to be installed on top of the base distribution. This is the normal course of action and is exactly what everybody using this technology is supposed to do in order to support the goal of a certain data analysis or data science project. Clearly, the additional features employed in the examples of this book are not all those available, and neither are all those somehow important, that would be simply impossible to cover. New features come out continuously, so in learning the fundamentals, it is important to practice with the approach, familiarize yourself with the environment, and exercise with the most fundamental tools, so as to be perfectly able to explore the new features and tools that become available.

Just keep in mind that these are professional-grade tools, not merely didactic ones to be abandoned after the training period. Thousands of experienced data scientists use these tools in their daily jobs and for top-level data science projects, so the instruments you start knowing and handling are powerful.

Data Science Fundamentals with R, Python, and Open Data, First Edition. Marco Cremonini.

Companion website: www.wiley.com/go/DSFRPythonOpenData

1.1.1 R Language

CRAN (the Comprehensive R Archive Network, https://cloud.r-project.org/) is the official online archive for all R versions and software packages available to install. CRAN is mirrored on a number of servers worldwide, so, in practice, it is always available.

The R base package is basically compliant with all desktop platforms: Windows, MacOS, and Linux. The installation is guided through a standard wizard and is effortless. Mobile platforms such as iOS and Android, as well as hybrid products, like the Chromebook, are not supported. For old operating system versions, the currently available version of R might not be compatible. In that case, under *R Binaries*, all previous versions of R are accessible, the most recent compatible one can be installed with confidence, and all the important features will be available.

At the end of the installation, a link to an R execution file will be created in the programs or applications menu/folder. That is not the R language, but an old-fashioned IDE that comes with the language. You do not need that if you use RStudio, as is recommended. You just need to install the R language, that is all.

1.1.2 RStudio Desktop

The *RStudio Desktop* is an *integrated development environment* (IDE) for R programming, recently enhanced with features to interpret Python scripts too (https://posit.co/download/rstudio-desktop/). In short, this means that it is a tool offering a graphical interface that accommodates most of the necessary functionalities for developing projects using R, which is a separate component, as we have seen in the previous section. The RStudio IDE is unanimously considered one of the best available IDEs, being complete, robust, and consistent throughout the versions. For this reason, there is not much competition in that market, at least until now. It is simply the safest and best choice. Icons of R and of RStudio might be confusing at first, but they both show a big R.

It is important to familiarize yourself with RStudio's layout because of the many functionalities available and useful in data science projects. The layout is divided into four main *quadrants*, as shown in Figure 1.1, with quadrant *Q1* that appears only when an R Script is created from the drop-down menu of the top-left icon.

- *Q1*: The quadrant for editing the *R code*, with different scripts is shown in separate tabs on top.
- *Q2*: The main feature is the *R Console,* where single command line instructions can be executed and the output of the execution of an R script appears.
- *Q3*: Information about the environment is provided through this quadrant, such as R objects (variables) created in memory during the execution of code; Python objects too could be shown if software allowing for integration between the two languages is used.
- *Q4*: A multifunction quadrant allowing for exploring the local file system (tab *Files*), visualizing graphics (tab *Plots*), the R package manager (tab *Packages*), and online documentation (tab *Help*).

1.1.3 Package Manager

The package manager is a key component of open-source environments, frequently used for updating a configuration, adding new functionalities, duplicating a configuration for testing purposes, and so forth. Installing new components is a common and recurrent activity in environments like R and Python, so it has to be simple and efficient. This is the crucial role of a package manager.

A package manager is typically a piece of software with few functionalities that basically revolve around listing the installed packages, updating them, searching for new ones, installing them, and removing useless packages. Everything else is basically accessory features that are not

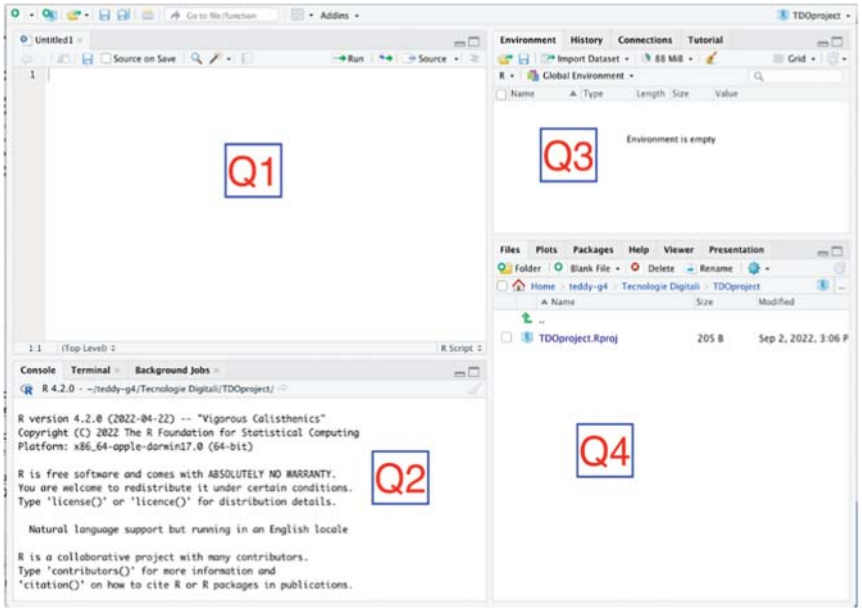


Figure 1.1 RStudio Desktop's standard layout

strictly necessary. Given the few specialized features a package manager must have, it should come without any surprise that modern package managers have their origins in classical command line tools. Actually, they still exist and thrive; they are often used as command line tools both in R and Python environments, just because they are simple to use and have limited options.

At any rate, a graphical interface exists, and RStudio offers it with the tab *Packages* in the Q4 quadrant. It is simple, just a list of installed packages and a selection box indicating if a package is also loaded or not. Installing and loading a package are two distinct operations. *Installing* means retrieving the executable code, for example, by downloading it from CRAN and configuring it in the local system. *Loading a package means making its functionalities available for a certain script*, which translates into the fundamental function **`library(`**`<name of the package to load>`**`)`**. Ticking the box beside a package in the RStudio package manager will execute on the R Console (quadrant Q2) the corresponding `library()` instruction. Therefore, using the console or ticking the box for loading a package is exactly the same.

However, neither of them is a good way to proceed, when we are writing R scripts, because a script should be reproducible, or at least understandable by others, at a later time, possibly a long time later. This means that all information necessary for reproducing it should be explicit, and if the list of packages to be loaded is defined externally by ticking selection boxes or running commands on the console, that knowledge is hidden, and it will be more difficult to understand exactly all the features of the script. So *the correct way to proceed is to explicitly write all necessary* `library()` *instructions in the script, loading all required packages.*

The opposite operation of loading a package is unloading it, which is certainly less frequent; normally, it is not needed in scripts. From the RStudio interface, it could be executed by unticking a package or by executing the corresponding instruction `detach("package:<name of the package>", unload=TRUE)`.

A reasonable doubt may arise about the reason why installed packages are not just all loaded by default. Why bother with this case-by-case procedure? The reason is memory, the RAM, in

particular, that is not only finite and shared by all processes executed on the computer, but is often a scarce resource that should be used efficiently. Loading all installed packages, which could be dozens or even hundreds, when normally just a few are needed by the script in execution, is clearly a very inefficient way of using the RAM. In short, we bother with the manual loading of packages to save memory space, which is good when we have to execute computations on data.

Installing R packages is straightforward. The interactive button **Install** in tab *Packages* is handy and provides all the functionalities we need. From the window that opens, the following choices should be made:

- *Install from*: From which repository should the package be downloaded? Options are: **CRAN**, the official online repository, this is the default and the normal case. **Package Archive File** is only useful if the package to install has been saved locally, which may happen for experimental packages not available from GitHub, which is a rare combination. Packages available from GitHub could be retrieved and installed with a specialized command (`githubinstall ("PackageName")`).
- *Packages*: The name of the package(s) to install; the autocomplete feature looks up names from CRAN.
- *Install to library*: The installation path on the local system depends on the R version currently installed.
- *Install dependencies*: Dependencies are logical relationships between different packages. It is customary for new to packages exploit features of previous packages for many reasons, either because they are core or ancillary functionalities with respect to the features provided by the package. In this case, those functionalities are not reimplemented, but the package providing them is logically linked to the new one. This, in short, is the meaning of dependencies. It means that when a package is installed if it has dependencies, those should be installed too (with the required version). This option, when selected, automatically takes care of all dependencies, downloading and installing them, if not present. The alternative is to manually download and install the packages required as dependencies by a certain package. The automatic choice is usually the most convenient. Errors may arise because of dependencies, for example, when for any reason, the downloading of a package fails, or the version installed is not compatible. In those cases, the problem should be fixed manually, either by installing the missing dependencies or the one with the correct version.

1.1.4 Package Tidyverse

The main package we use in this book is called ***tidyverse*** (https://www.tidyverse.org/). It is a particular package because it does not directly provide new features, but rather groups a bunch of other packages, which are then installed all at once, and these provide the additional features. In a way, *tidyverse* is a shortcut created to simplify the life of people approaching data science with R, instead of installing a certain number of packages individually, common to the majority of projects, they have been encapsulated in just one package that does the whole job.

There are criticisms of this way of doing things based on the assumption that only necessary packages should be installed and, most of all, loaded. This principle is correct and should be followed as a general rule. However, a trade-off is also reasonable in most cases. Therefore, you may install *tidyverse* and then load only specific packages in a script, or just load the whole lot contained in *tidyverse*. Usually, it does not make much difference; you can choose without worrying too much about this.

In any case, *tidyverse* is widely used, and for this, it is useful to spend some time reading the description of the packages included in it because this provides a glimpse into what most data science projects use, the types of operations and more general features. In our examples, most of the functions we will use are defined in one of the *tidyverse* packages, with some exceptions that will be introduced.

The installation of *tidyverse* is the standard one, through the RStudio package manager, or the console with command `install.packages("tidyverse")`. Loading it in a script is done with `library(tidyverse)` for a whole lot of packages, or alternatively, for single packages such as `library(readr)`, where *readr* is the name of a package contained in *tidyverse*. In all cases, after the execution of a `library` instruction, the console shows if and what packages have been loaded.

In all our examples, it should be assumed that the first instruction to be executed is `library(tidyverse)`, even when not explicitly specified.

1.2 Python Language and Tools

Python's environment is more heterogeneous than R's, mostly because of the different scope of the language – Python is a general-purpose language mostly used for web and mobile applications, and in a myriad of other cases, data science is among them – which implies that several options are available as a convenient setup for data science projects. Here, one of the most popular is considered, but there are good reasons to make different choices.

The first issue to deal with is that, until now, there is not a data science Python IDE comparable to RStudio for R, which is the *de facto* standard and offers almost everything that is needed. In Python, you have to choose if you want to go with a classical IDE for coding; there are many, which is fine, but they are not much tailored for data science wrangling operations; or if you want to go with an IDE based on the ***computational notebook*** format (just *notebook* for short). The notebook format is a hybrid document that combines *formatted text*, usually based on a ***Markdown*** syntax and blocks of executable code. For several reasons, mostly related to utility in many contexts to have both formatted text and executable code, and the ease of use of these tools, IDEs based on the notebook format have become popular for Python data science and data analysis. The code in the examples of the following chapters has been produced and tested using the main one of these IDEs, ***JupyterLab*** (https://jupyterlab.readthedocs.io/en/latest/). It is widely adopted, well-documented, easy to install, and free to use. If you are going to write short blocks of Python code with associated descriptions, a typical situation in data science, it is a good choice. If you have to write a massive amount of code, then a classical IDE is definitely better. Jupyter notebooks are textual files with canonical extension ***.ipynb*** and an internal structure close to JSON specifications.

So, the environment we need has the *Python base distribution*, a *package manager*, for the same reasons we need it with R, the two packages specifically developed for data science functionalities called ***NumPy*** and ***pandas***, and the *notebook-based IDE* ***JupyterLab***. These are the pieces. In order to have them installed and set up, there are two ways of proceeding: one is easy but inefficient, and the other is a little less easy but more efficient. Below, with ***A*** and ***B,*** the two options are summarized.

A. A single installer package, equipped with a graphical wizard, installs and sets up everything that is needed, but also much more than you will likely ever use, for a total required memory space of approximately 5 GB on your hard disk or SSD memory.

B. A manual procedure individually installs the required components: first Python and the package manager, then the data science libraries *NumPy* and *pandas*, and finally the JupyterLab IDE. This requires using the command line shell (or terminal) to run the few installation instructions for the package manager, but the occupied memory space is just approximately 400 MB.

Both ways, the result is the Python setup for learning the fundamentals of data science, getting ready, and working. The little difficulty of the B option, i.e. using the command line to install components, is truly minimal and, in any case, the whole program described in this book is about familiarizing with command line tools for writing R or Python scripts, so nobody should be worried for a few almost identical commands to run with the package manager.

So, the personal suggestion is to try the ***B*** option, as described in the following, operationally better and able to teach some useful skills. At worst, it is always possible to backtrack and go with the easier ***A*** option on a second try.

1.2.1 Option A: Anaconda Distribution

Option A is easy to explain. There is a tool called **Anaconda Distribution** (https://www.anaconda.com/products/distribution) that provides everything needed for an initial Python data science environment. It contains all the components we have listed as well as tons of other tools and libraries. In addition, it offers a desktop application called *Anaconda Navigator*, which is basically a graphical interface to the package manager ***conda***. Unfortunately, this interface is quite bulky. From this interface, it is also possible to launch the **JupyterLab** IDE.

1.2.2 Option B: Manual Installation

Option B requires a few steps:

Step 1: Python and package manager installation.
From the official repository of Python distribution (https://www.python.org/downloads/), the installer for the latest (or previous) distribution could be downloaded and launched. A graphical wizard guides the process. In the end, the Python language with its basic libraries and two package managers will be installed: ***pip***, the standard Python package manager, and ***conda***, the Anaconda's one. The differences between the two are minimal, and for all our concerns, they are equivalent. Even the syntax of the commands is basically the same. The only recommendation is to choose one and continue using that one for package management; this avoids some possible problems with dependencies. We show the examples using *pip*, but *conda* is fine as well.

Step 2: Installing data science packages *NumPy, pandas*, and *JupyterLab* IDE.
From a shell (e.g. *Terminal* on MacOS, *Powershell* on Windows), to run the package manager, it suffices to digit `pip` (or `conda`, for the other one) and return.

This way, a list of the options is shown. The most useful are:

- `pip list`: list all installed packages with the version.
- `pip install <package_name>`: install a package.
- `pip uninstall <package_name>`: uninstall a package.

When a package is installed or uninstalled, on the command line appears a request to confirm the operation; the syntax is `[n/Y]`, with ***n*** for No and ***Y*** for Yes.