

Gearing Up

For this book, we rely on a number of different software tools, and with the exception of Microsoft Excel, all of them are available free of charge and for all major operating systems (Windows, macOS, or Linux). The most important one is R, a free, open-source statistical toolkit that comes with its own programming language. As stated in the previous chapter, it is required that readers have some experience in R, as the book does not include a basic introduction. The best way to work with R is to use RStudio, a powerful interface to the R engine. You will be able to complete Parts I and II of the book with R and RStudio only; if you also cover the more advanced chapters in Parts III and IV, you will also need the PostgreSQL database management system.

In this chapter, we go through the software required for the book. Detailed installation steps, as well as the sample datasets discussed in the book, are provided as part of the book's companion website at

<https://dmbook.org>

where you will always find up-to-date instructions and data. You do not have to install all the software tools below at once. It is perfectly possible to start with R and RStudio, and later return to this setup as you begin exploring the more advanced chapters on database systems, starting with Chapter 8.

2.1 R AND RSTUDIO

Please follow the installation instructions on the book's website to install the R statistical toolkit on your system. The R software includes the main

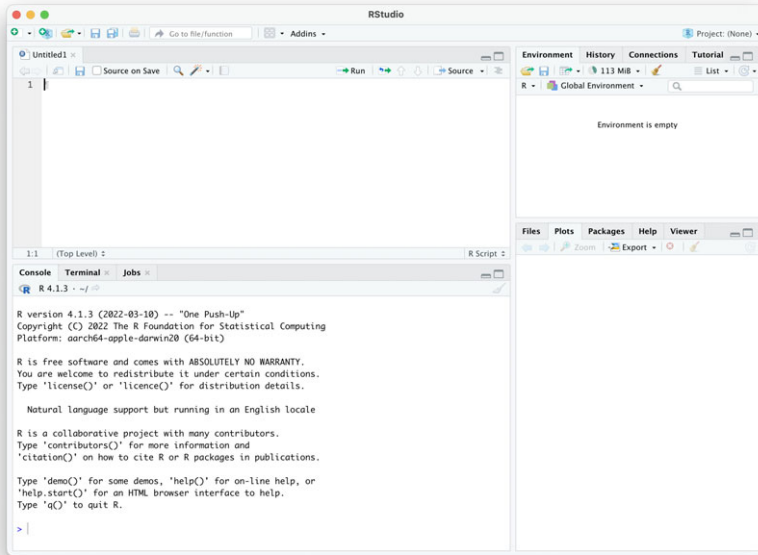


FIGURE 2.1. The RStudio interface.

engine that does most of the work: It executes the commands you enter, reads datasets, runs statistical models, and generates plots. The commands for doing this must be specified in the R programming language. While it is possible to work with R out of the box, I recommend that you also install a much more powerful development interface for R: RStudio. Instructions for this are also provided on the book's companion website. After the installation, start RStudio, and click on **File** » **New File** » **R Script**. Your RStudio window should now look like Figure 2.1.

Let us go through some of the main elements of RStudio. At the bottom left, you see the R console. This is where you see the output produced by R (unless this output is graphical). You can also use the console to send short commands to R. For example, if you type

```
Sys.Date()
```

on the console and hit **Return**, R will show you the current date. While you could do all the work for this book via the console by entering commands one by one, this is generally not a good idea since all this work would be lost when you close RStudio. This is why we typically work with *files* of R code, such as the one you created (which, at the moment, is still empty).

The files are displayed in the editor window of RStudio, at the top left just above the console. You can enter the aforementioned command in this window and save the file with   under a specific name. This will preserve your R code such that you can later open and modify it. The green arrow at the top of the code editor allows you to run the currently selected part of code. It is absolutely essential for reasons of transparency and replicability that you properly store your code in files, so always use the code editor unless you are testing short commands!

On the right of the RStudio interface you can see two sets of panels, one with different panels called “Environment,” “History,” etc, the other with “Files,” “Plots,” etc. These panels become active once you continue to work with your R project. For example, if you load a dataset into R (which we will do in later chapters), you will see a new entry in the “Environment” panel that allows you to view the new dataset. Also, you can view graphics created by R in the “Plots” panel at the bottom. RStudio is an extremely powerful and versatile development environment for R, and we cannot go into more details here. There are a number of introductions available online, which you should consult if you want to learn more about RStudio’s features.

Nevertheless, I want to make one recommendation: Resist the temptation to use the different menu-based features in RStudio. For example, it is possible to read data using RStudio’s import feature under  . This will internally execute one of R’s import functions for you. However, unless you save the corresponding R code displayed on the console explicitly as part of your R file, it will be lost when you close RStudio, making replication and error correction impossible. This is why I recommend that, wherever possible, you rely entirely on R code written by yourself, which you can properly save in your R file. This way, you later have a complete record of the individuals steps you carried out, which makes it possible to correct/extend your analysis when necessary, or share it with others so that they can replicate exactly what you did.

2.2 SETTING UP THE PROJECT ENVIRONMENT FOR YOUR WORK

The examples in the book cover many files and datasets, and they require a number of R packages. This setup has been prepared as a pre-configured RStudio project, to make it as easy as possible for you to get started. Go to the companion website for this book, which includes a link to this material. The download comes as a single zip file. Unpack the archive by

double-clicking, and move *the entire content of the archive* to a newly created folder that you would like to use as your main directory for the exercises, for example, `dmbook`.

R uses a given folder as a working directory, which is where it looks first when you open a file, or where it saves a file unless you specify a different path. For example, if the `dmbook` folder is placed in your Documents folder, then `Documents/dmbook` should be your main working directory (or “project” directory) for the book. Note that the directory paths provided here use the notation on macOS and Linux systems (with a forward slash / separating the different folder levels). On Windows systems, directory paths are specified using backslashes, for example `C:\Documents\dmbook`, so the paths will look slightly different.

Inside your main project directory, you will find a number of files and directories, which were originally contained in the archive you downloaded. This is roughly what your project directory looks like:

```
/Users/nils/Documents/dmbook/
├── ch04/
├── ch05/
├── ...
├── ch13/
├── dmbook.Rproj
├── ex04/
├── ...
├── ex13/
├── renv/
└── renv.lock
```

Let us quickly go through the most important folders and files. The data used in the code examples of the book is contained in the sub-folders (`ch04`, `ch05`, etc) for each chapter. If you follow the code examples in the book, you will need the files in these folders. Similarly, additional data for the exercises is contained in the sub-folders `ex04`, `ex05`, etc, again ordered by chapter. The file `dmbook.Rproj` is a project configuration file for RStudio. It is good practice to use these project files when working with RStudio. When you double-click this file (don’t do this yet!), RStudio will open a new session and switch to the directory containing the file as the working directory. The `renv` folder and lockfile contain the project setup for the book, which we introduce below.

R will treat all file names as *relative to the working directory, that is, the location of this project file*. There is one issue, however, that arises due

to the differences in how operating systems denote file paths. I mentioned earlier that macOS and Linux use forward slashes, and Windows uses backslashes. In our code examples, we often access files, for example, when importing data into R for further processing. To avoid including separate code examples for macOS/Linux and Windows, I rely on the built-in `file.path()` function that adjusts file paths depending on the underlying operating system. For example, if we want to access the file `csv-example.csv` in the `ch04` subfolder, we can simply refer to this file with

```
file.path("ch04", "csv-example.csv")
```

in our code, and R will automatically convert this to `ch04/csv-example.csv` for macOS or Linux, and to `ch04\csv-example.csv` if you use Windows. This file path is relative to the working directory, so we can omit the path to this directory (e.g., `/Users/nils/Documents/dmbook` or `C:\Documents\dmbook`).

2.2.1 R's Extension Libraries

One of the core strengths of the R system is its extensibility. There are thousands of packages for R that extend R's functionality in different ways. In this book, we rely on a number of these packages. Before you can load a package in an R session to use it, you must install it on your system. The standard way of doing this is via R's command line with

```
install.packages("tidyverse", dependencies = T)
```

This will make sure that apart from the new package itself, R will also install other packages that the new package depends on. Alternatively, you can use RStudio for installing packages, using  **Install Packages** in the menu bar.

Along with the code of the installed packages, you get the documentation of the functions it contains. It is absolutely essential that you learn to use this documentation, as it contains all the necessary information for you to use the package correctly and efficiently. In many cases, these documentations are not written as accessible introductions and may be difficult to read. This is why I give many pointers to useful functions and parameters, which you can then look up yourself if you need more details about how they work. The simplest way to display the reference for a function is the `?` operator, followed by the name of a package or a function. For example,

```
?install.packages
```

shows you the documentation for the `install.packages()` function in RStudio's help window on the bottom right.

Many users do not worry too much about package management and simply install packages in their main local library when they need them. This is R's default behavior, and it works just fine for most applications. However, I prefer a more sophisticated approach to package management: the use of different R *environments*. An environment is simply the set of all packages used for a particular project, such that each project keeps a separate list of packages (and their versions) it requires, without interfering with others. This also has the advantage of us being able to distribute a project along with a list of required packages, such that R can *automatically install all of them*.

We use the `renv` package to enable package management within an environment specifically for this book. The R project environment you downloaded above has `renv` enabled by default. If you double-click the `dmbook.Rproj` file that was distributed with the online material for the book, RStudio opens a new session and initializes the environment. It first downloads `renv` and does a check if all required packages (as specified in the `renv.lock` file) are installed. If packages are missing or are not available in the specified version, a warning appears. You can now type

```
renv::restore()
```

on the R console, and `renv` shows you a list of all required packages. After you confirm with `y`, it downloads and installs them. Note that the packages are installed in the respective version that was tested for the book, which is probably not the latest one. However, this is not a problem; in line with `renv`'s approach to compartmentalize installed packages into different environments, these package will be available only in the project environment we use for the book. This means that they are *not* available for your other projects unless you install them there as well. Under Windows, some package installations can fail, in particular for those where `renv` cannot find the pre-compiled version and instead relies on a source package. If you encounter this problem, I recommend that you do a

```
renv::equip()
```

and then try `renv::restore()` again.

2.3 THE POSTGRESQL DATABASE SYSTEM

We discussed in the introductory chapter that for certain applications, it is useful to keep your data in a specific system optimized for data storage and processing, a database management system (DBMS). There are different DBMS for different kinds of data, and in this book we will examine one of them in particular: The PostgreSQL database management system, which we use in Chapter 8 and the following ones. PostgreSQL is a *relational* DBMS designed for databases that contain tables, but it can also deal with more complex types of data. The installation process differs slightly between operating systems, which is why you should once again refer to the online repository to obtain more information required for the precise steps required (see the link at the beginning of this chapter). *Before proceeding, it is necessary that you complete the individual steps for your operating system described on the book's companion website.*

PostgreSQL is a multi-user system, and each user must identify with a username and a password. PostgreSQL installations under different operating systems use different approaches here. The default usernames differ, and some allow you to set your own password while others do not require a password (just a username). This is why after installing PostgreSQL, *make sure to memorize the username and the password* to access PostgreSQL on your system. The online installation instructions for this book contain more information about this.

With PostgreSQL set up on your computer, it is a good idea to test whether the connection works. In R, make sure that you have the RPostgres package installed along with all the other packages it depends on (if you use the pre-configured R environment described earlier, this is done automatically). The following code should then output the PostgreSQL version you are running. Make sure to adjust the username and password to match your setup (see the online instructions). `postgres` and `pgpasswd` are just placeholders, which we use here and later in the book – they may not work on your system.

```
library(RPostgres)
db <- dbConnect(Postgres(),
  user = "postgres",
  password = "pgpasswd")
dbGetQuery(db, "SELECT version()")
dbDisconnect(db)
```

2.3.1 Setting Up a New PostgreSQL Database

A database server can work with multiple databases, each of which is a collection of data that belong to one project. In this book, I follow the convention to use *a new database for each chapter of the book*, such that the examples and exercises for each of the chapters do not interfere with each other. The code below shows how to create a new database `dbintro`, which we use in Chapter 8 of the book. Again, make sure to adjust your username and password! You can use this code to create more databases for the subsequent chapters – just replace `dbintro` with the name of the database you would like to create. The code is presented here without much further explanation; in Chapter 8, we go through the process of connecting to the server step by step.

```
library(RPostgres)
db <- dbConnect(Postgres(),
  user = "postgres",
  password = "pgpasswd")
dbExecute(db, "CREATE DATABASE dbintro")
dbDisconnect(db)
```

2.3.2 Code Examples and Style

R allows you to be quite flexible in how you write your code, within the limits of the R syntax. To be consistent in the code I present in this book, I follow Hadley Wickham's *tidyverse Style Guide* (Wickham, 2021). Although it is designed for R code within the *tidyverse* framework (see Chapter 7), much of the recommendations also apply to code outside this framework. Here are some conventions used throughout this book:

- All file and directory names are lowercase. Different parts of the file name will be separated with a hyphen. Example: `csv-example.csv`
- R objects have lowercase names, and different parts of the object name are separated with an underscore. Example: `dataset_new`
- We use the `.R` ending for R code files.

For readers with an electronic copy of this book, it may be tempting to simply copy and paste the code examples into RStudio. Try not to do this. Rather, I strongly recommend that you *type the code yourself* and make modifications to it. This allows you to become more independent and experienced as an R user, but also to find out what does *not* work and why.

2.4 SUMMARY AND OUTLOOK

Data management requires a number of different tools, and in this chapter we covered those required for this book. Most importantly, we rely on the R statistical toolkit and the RStudio environment for most of the exercises. When you work in R, you mostly rely on data stored in files. This works for many applications, but sometimes our datasets become bigger and more complex. In these cases, it is useful to store data in specialized DBMS. These systems allow you to quickly search and filter large datasets, to check your data for consistency, or to manage access to the data by multiple users. We use the DBMS later to perform various operations, such as creating a database or loading data into it. With the technical preparations out of the way, we can now proceed to lay some theoretical groundwork. Chapter 3 discusses some general concepts about data, and introduces the most important data structure for the social sciences: tables.