Modern R with the {tidyverse}

Bruno Rodrigues

2023-07-02

Table of contents

Welcome!

Learn the R programming language and the {tidyverse} suite of packages

1 Preface

2 Introduction

2.1 What is R? What are packages?

Read R's official answer to this question here[1]. To make it short: R is a multi-paradigm (procedural, imperative, object-oriented and functional) programming language that focuses on applications in *statistics*. By *statistics* I mean any field that uses statistics such as official statistics, economics, finance, data science, machine learning, etc. For the sake of simplicity, I will use the word "statistics" as a general term that encompasses all these fields and disciplines for the remainder of this book. R and RStudio are the two main pieces of software that we are going to use. R is the programming language and RStudio is a modern IDE for it, made by a company called Posit (formerly the company was also called RStudio). There are several other IDEs that you can use though, but if you're a beginner RStudio is very good choice.

2.2 Why learn R?

R is widely used and popular. It has been around for more than 3 decades and chances are that it will continue to stick around for many decades to come.

[1]https://cran.r-project.org/doc/FAQ/R-FAQ.html#What-is-R_003f

TO ADD MORE STUFF

To start using R, simply install it. Installation is simple, but operating system dependent. To download and install R for Windows, follow this link[2]. For macOS, follow this one[3]. If you run a GNU+Linux distribution, you can install R using the system's package manager. If you're running Ubuntu, you might want to take a look at r2u[4], which provides very fast installation of packages, full integration with `apt` (so dependencies get solved automatically) and covers the entirety of CRAN.

To use R, I recommend also installing RStudio. RStudio is a very popular text editor suited for R development. To install RStudio, look for your operating system here[5]. There are other popular choice to write R code, such as VS Code and Emacs (with ESS), but if you're a beginner RStudio is a solid choice.

2.3 Who is this book for?

This book can be useful to different audiences. If you have never used R in your life, and want to start, start with Chapter 1 of this book. Chapter 1 to 3 are the very basics, and should be easy to follow up to Chapter 7. Starting with Chapter 7, it gets more technical, and will be harder to follow. But I suggest you keep on going, and do not hesitate to contact me for help if you struggle! Chapter 7 is also where you can start if you are already familiar with R **and** the `{tidyverse}`, but not functional programming. If you are familiar with R but not the `{tidyverse}` (or have no clue what the `{tidyverse}` is), then you can start

[2]https://cloud.r-project.org/bin/windows/base/
[3]https://cloud.r-project.org/bin/macosx/
[4]https://github.com/eddelbuettel/r2u
[5]https://posit.co/download/rstudio-desktop/

with Chapter 4. If you are familiar with R, the `{tidyverse}` and functional programming, you might still be interested in this book, especially Chapter 9 and 10, which deal with package development and further advanced topics respectively.

2.4 Why this book?

This book is first and foremost for myself. This book is the result of years of using and teaching R at university and then at my jobs. During my university time, I wrote some notes to help me teach R and which I shared with my students. These are still the basis of Chapter 2. Then, once I had left university, and continued using R at my first "real" job, I wrote another book that dealt mostly with package development and functional programming. This book is now merged to this one and is the basis of Chapters 9 and 10. During these years at my first job, I was also tasked with teaching R. By that time, I was already quite familiar with the `{tidyverse}` so I wrote a lot of notes that were internal and adapted for the audience of my first job. These are now the basis of Chapters 3 to 8. Then, during all these years, I kept blogging about R, and reading blogs and further books. All this knowledge is condensed here, so if you are familiar with my blog, you'll definitely recognize a lot of my blog posts in here. So this book is first and foremost for me, because I need to write all of this down in a central place. So because my target audience is myself, this book is free. If you find it useful, and are in the mood of buying me a coffee, you can, but if this book is not useful to you, no harm done (unless you paid for it before reading it, in which case, I am sorry to have wasted your time). But I am quite sure you'll find some of the things written here useful, regardless of your current experience level with R.

2.5 Why *modern* R?

Modern R instead of "just" R because we are going to learn how to use modern packages (mostly those from the {tidyverse}[6]) and concepts, such as functional programming (which is quite an old concept actually, but one that came into fashion recently). R is derived from S, which is a programming language that has roots in FORTRAN and other languages too. If you learned R at university, you've probably learned to use it as you would have used FORTRAN; very long scripts where data are represented as matrices and where row-wise (or column-wise) operations are implemented with `for` loops. There's nothing wrong with that, mind you, but R was also influenced by Scheme and Common Lisp, which are functional programming languages. In my opinion, functional programming is a programming paradigm that works really well when dealing with statistical problems. This is because programming in a functional style is just like writing math. For instance, suppose you want to sum all the elements of a vector. In mathematical notation, you would write something like:

[*{i = 1}^{100}* x{i}]

where x is a vector of length 100. Solving this using a loop would look something like this:

```r
res <- 0
for(i in 1:length(x)){
  res <- x[i] + res
}
```

[6] https://posit.co/download/rstudio-desktop/

This does not look like the math notation at all! You have to define a variable that will hold the result outside of the loop, and then you have to define `res` as something plus `res` inside the body of the loop. This is really unnatural. The functional programming approach is much easier:

```
Reduce(`+`, x)
```

We will learn about `Reduce()` later (to be more precise, we will learn about `purrr::reduce()`, the "tidy" version of `Reduce()`), but already you see that the notation looks a lot more like the mathematical notation.

At its core, functional programming uses functions, and functions are so-called *first class* objects in R, which means that there is nothing special about them… you can pass them to other functions, create functions that return functions and do any kind of operation on them just as with any other object. This means that functions in R are extremely powerful and flexible tools. In the first part of the book, we are going to use functions that are already available in R, and then use those available in packages, mostly those from the `tidyverse`. The `tidyverse` is a collection of packages developed by Hadley Wickham, and several of his colleagues at RStudio, Inc. By using the packages from the `tidyverse` and R's built-in functional programming capabilities, we can write code that is faster and easier to explain to colleagues, and also easier to maintain. This also means that you might have to change your expectations and what you know already from R, if you learned it at University but haven't touched it in a long time. For example for and while loops, are relegated to chapter 8. This does not mean that you will have to wait for 8 chapter to know how to repeat instructions N times, but that *for* and *while* loops are tools that are very useful for very specific situations that will be discussed at that point.

In the second part of the book, we are going to move from using R to solve statistical problems to developing with R. We are going to learn about creating your own package. If you do not know what packages are, don't worry, this will be discussed just below.

2.6 What to expect from this book?

The idea of Chapters 1 to 7 is to make you efficient with R as quickly as possible, especially if you already have prior programming knowledge. Starting with Chapter 8 you will learn more advanced topics, especially programming with R. R is a programming language, and you can't write "programming language" without "language". And just as you wouldn't expect to learn French, Portuguese or Icelandic by reading a single book, you shouldn't expect to become fluent in R by reading a single book, not even by reading 10 books. Programming is an art which requires a lot of practice. Teach yourself programming in 10 years is a blog post written by Peter Norvig which explains that just as with any craft, mastering programming takes time. And even if you don't need or want to become an expert in R, if you wish to use R effectively and in a way that ultimately saves you time, you need to have some fluency in it, and this only comes by continuing to learn about the language, and most importantly practicing. If you keep using R every day, you'll definitely become very fluent. To stay informed about developments of the language, and the latest news, I advise you read blogs, especially R-bloggers which aggregates blog posts by more than 750 blogs discussing R.

So what you can expect from this book is that this book is not the only one you should read.

2.7 The author?

to add stuff

3 Getting to know RStudio

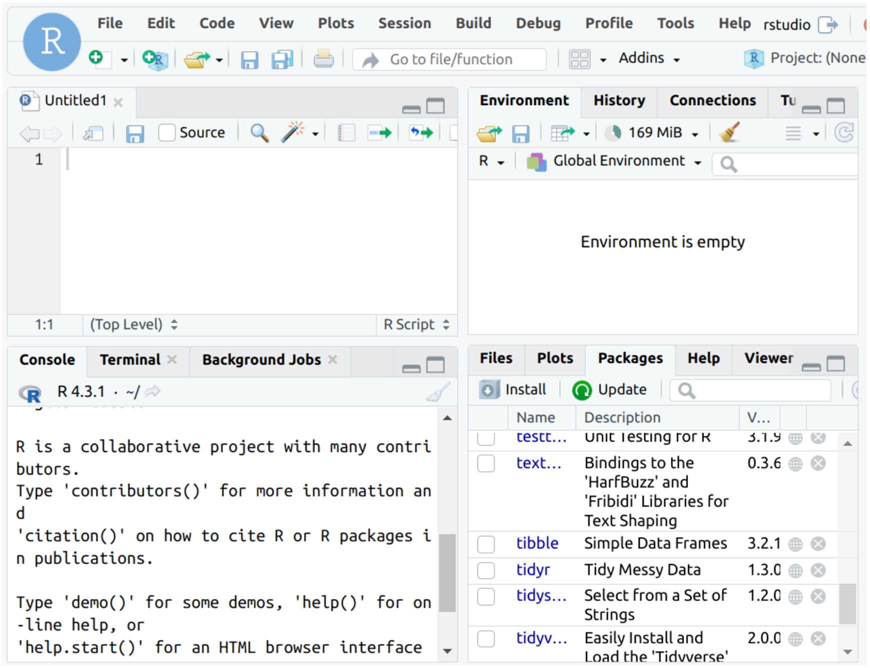
RStudio is an IDE (Integrated development environment) for the R programming language made by a company called Posit. You can install RStudio by visiting this link[1]. Posit, also develops Shiny, a package to create full-fledged web-apps. I am not going to cover Shiny in this book, since there's already a lot[2] of material that you can learn from.

Once you have installed RStudio, launch it and let's go through the interface together.

3.1 Panes

RStudio is divided into different panes. Each pane has a specific function. The image below shows some of these panes:

[1]https://posit.co/download/rstudio-desktop/
[2]https://shiny.posit.co/r/getstarted/next-steps/



```
R is a collaborative project with many contri
butors.
Type 'contributors()' for more information an
d
'citation()' on how to cite R or R packages i
n publications.

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

Name	Description	V...
testt...	Unit Testing for R	3.1.9
text...	Bindings to the 'HarfBuzz' and 'Fribidi' Libraries for Text Shaping	0.3.6
tibble	Simple Data Frames	3.2.1
tidyr	Tidy Messy Data	1.3.0
tidys...	Select from a Set of Strings	1.2.0
tidyv...	Easily Install and Load the 'Tidyverse'	2.0.0

Figure 3.1: The different panes of RStudio.

Take some time to look around what each pane shows you. Some panes are empty; for example the *Plots* pane or the *Viewer* pane. *Plots* shows you the plots you make. You can browse the plots and save them. We will see this in more detail in a later chapter. *Viewer* shows you previews of documents that you generate with R. More on this later. You can also minimize and maximize the panes by clicking these two buttons:

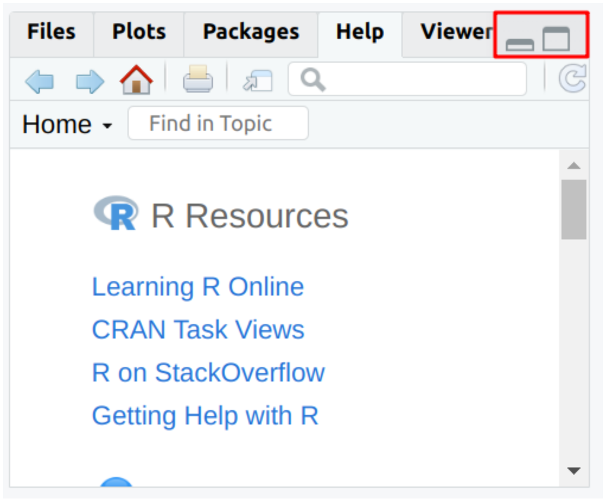


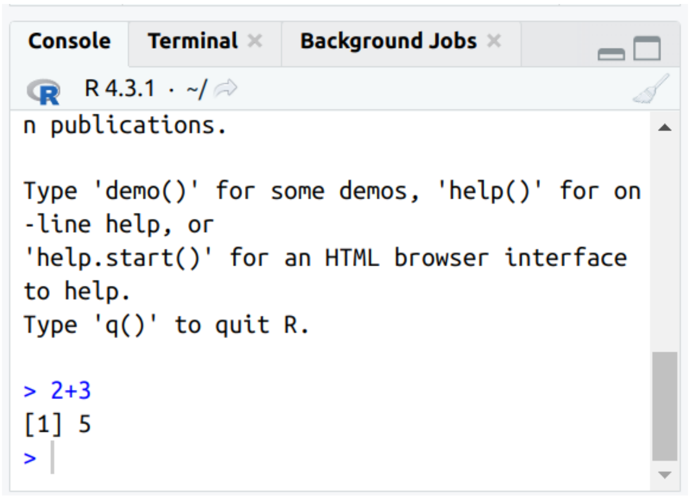
Figure 3.2: Minimize or maximize the panes.

3.2 Console

The *Console* pane is where you can execute R code. Write the following in the console:

```r
2 + 3
```

and you'll get the answer, `5`. However, do not write a lot of lines in the console. It is better write your code inside a script. Output is also shown inside the console.



```
n 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.

> 2+3
[1] 5
>
```

Figure 3.3: You can execute code by typing it in the console.

3.3 Scripts

Instead of writing code in the console, it is better to write code in a so-called script. Scripts are simple text files that can be written and executed by RStudio. To write a new script, click on the top-right icon and select "R script":

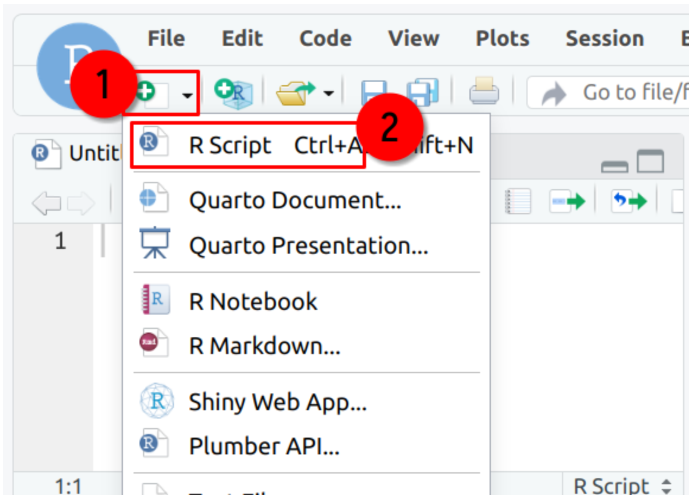


Figure 3.4: Select R Script to open an empty script.

In Figure 3.4, we see the user creating a new R script. If you have a background in the social sciences you might be familiar with STATA: STATA also uses scripts, colloquially called `.do` files. The C programming language uses `.c` files. R scripts have the `.r` or `.R` extension. But `.R` files are not the only type of files that you can edit with RStudio. We will explore other formats later in the book.

3.3.1 The help pane

The *Help* pane allows you to consult documentation for R, its packages etc:

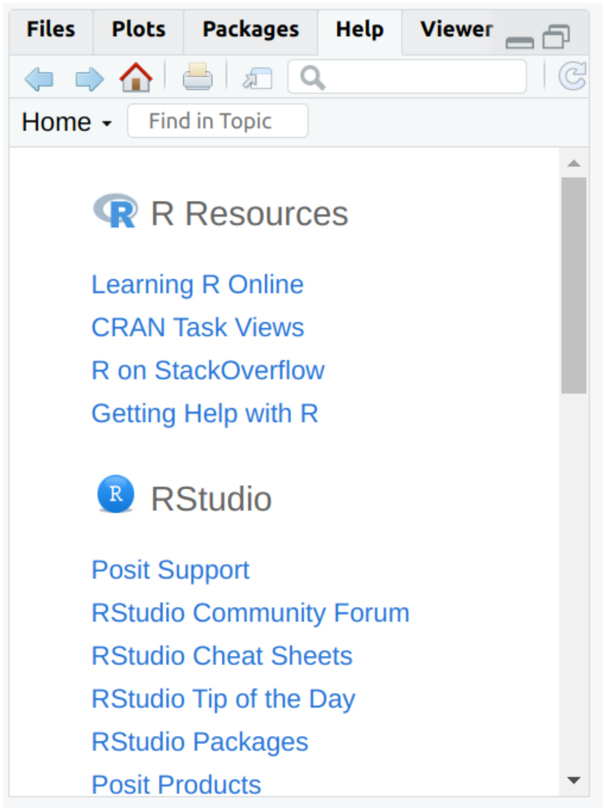


Figure 3.5: Read the flipping manual.

You can also read the help file of a specific function by writing `?function` in the console, where `function` is the function you want to know about.

I highly recommend you take some time to check out the "CRAN Task Views". These *views* provide a very nice summary of the different packages available for different scientific fields. For example, if you're doing econometrics, you should read the CRAN Task View: Econometrics[3].

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

Take some time to browse the different CRAN Task Views here[4].

3.3.2 The Environment pane

The *Environment* pane shows every object created in the current section. It is especially useful if you have defined lists or have loaded data into R as it makes it easy to explore these more complex objects. As you will write code and create objects throughout a session, the environment pane will get more and more populated.

3.4 Projects

One of the best features of RStudio are projects. Creating a project is simple; simply click on the top right corner of RStudio and then "New Project":

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

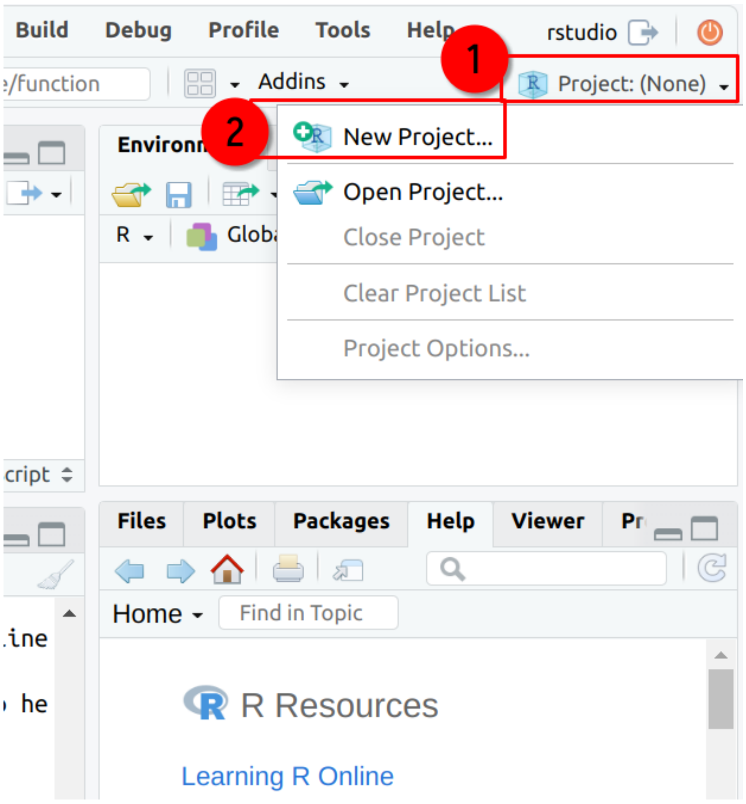


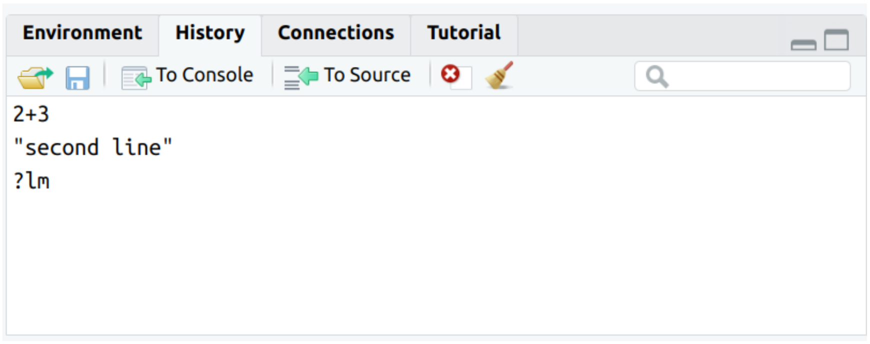
Figure 3.6: Here is how you create a project.

Projects make a lot of things easier, such as managing paths (more on this in the chapter about reading data). Another useful feature of projects is that the scripts you open in project A will stay open even if you switch to another project B, and then switch back to the project A again.

You can also use version control (with Git) inside a project. Version control is very useful, but I won't discuss it in this book.

3.5 History

The *History* pane saves all the previous lines you executed. You can then select these lines and send them back to the console or the script.



```
2+3
"second line"
?lm
```

Figure 3.7: The history pane contains all executed lines of code.

3.6 Plots

All the plots you make during a session are visible in the *Plots* pane. From there, you can export them in different formats.

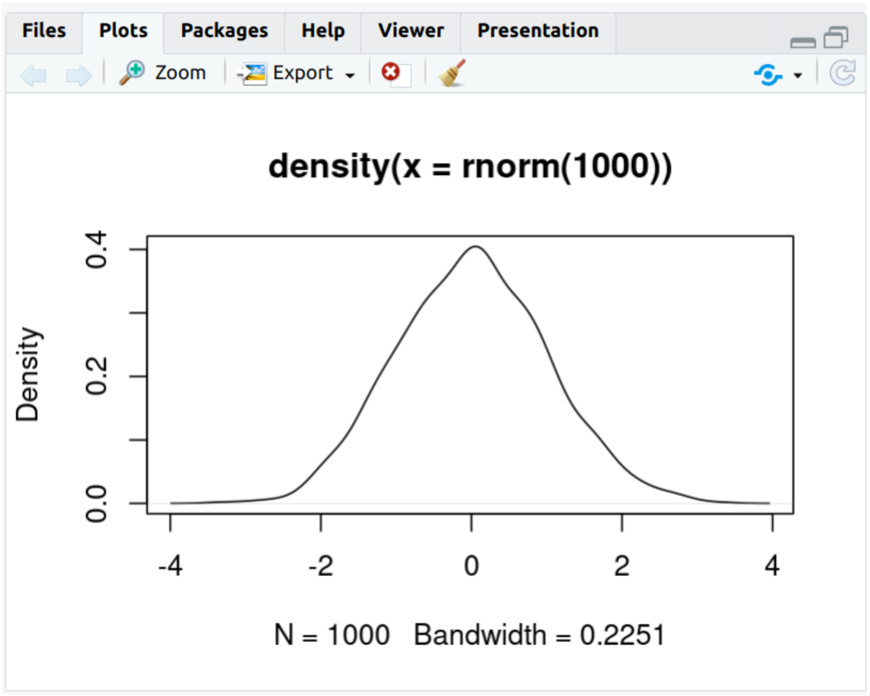


Figure 3.8: All the plots that you make get stored in the *plots* pane.

There are several ways to create plots: later in the book I will teach you some basics of the `{ggplot2}` package.

3.7 Addins

Some packages install addins, which are accessible through the addins button:

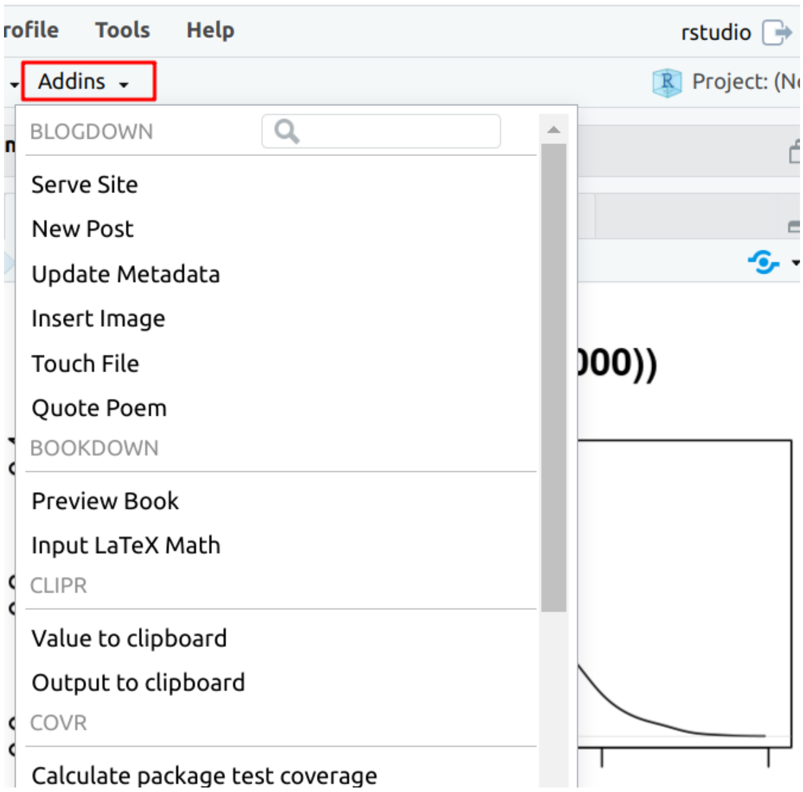


Figure 3.9: You can browse the available addins.

These addins make it easier to use some functions and you can read more about them here[5].

There are other panes that I will not discuss here, but you will naturally discover their use as you go. For example, we will discuss the *Build* pane in Chapter 11.

[5]https://rstudio.github.io/rstudioaddins/

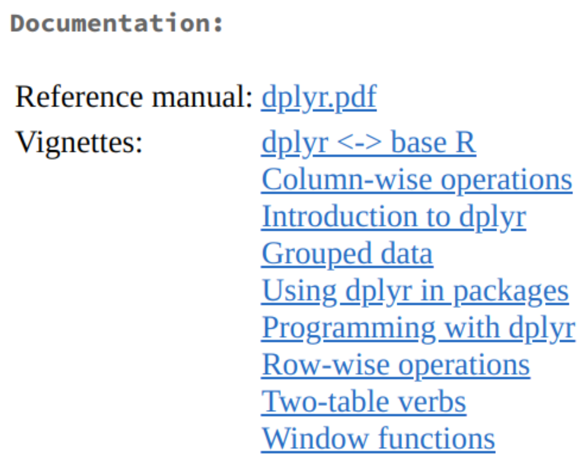
3.8 Packages

You can think of packages as addons that extend R's core functionality. You can browse all available packages on CRAN[6]. To make it easier to find what you might be interested in, you can also browse the CRAN Task Views as mentioned previously. Each package has a landing page that summarises its dependencies, version number etc. For example, for the `{dplyr}` package: https://CRAN.R-project.org/package=dplyr. To install a package, you can either type the following command in the console:

```r
install.packages("dplyr")
```

or you can go to the *Packages* pane and click on the *Install* button. Before installing a package, you can consult its manual and *vignettes* online. For example, if you go back to `{dplyr}`'s landing page, you can take a look at the *Reference Manual* and *Vignettes*:

[6]https://cloud.r-project.org/

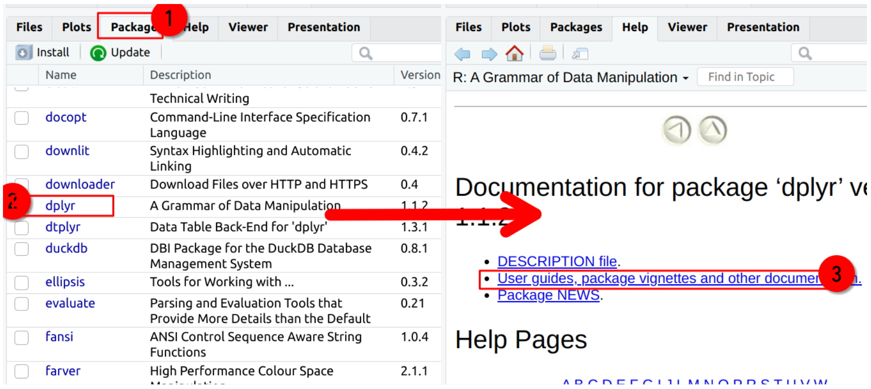


Documentation:

Reference manual:	dplyr.pdf
Vignettes:	dplyr <-> base R
	Column-wise operations
	Introduction to dplyr
	Grouped data
	Using dplyr in packages
	Programming with dplyr
	Row-wise operations
	Two-table verbs
	Window functions

Figure 3.10: Always take the time to read the document of the packages you use on a daily basis.

Vignettes are valuable documents; inside vignettes, the purpose of the package is explained in plain English, usually with accompanying examples. The reference manuals list the available functions inside the packages. You can also find vignettes from within RStudio (but you need to install the package first):



Name	Description	Version
	Technical Writing	
docopt	Command-Line Interface Specification Language	0.7.1
downlit	Syntax Highlighting and Automatic Linking	0.4.2
downloader	Download Files over HTTP and HTTPS	0.4
dplyr	A Grammar of Data Manipulation	1.1.2
dtplyr	Data Table Back-End for 'dplyr'	1.3.1
duckdb	DBI Package for the DuckDB Database Management System	0.8.1
ellipsis	Tools for Working with ...	0.3.2
evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.21
fansi	ANSI Control Sequence Aware String Functions	1.0.4
farver	High Performance Colour Space	2.1.1

R: A Grammar of Data Manipulation

Documentation for package 'dplyr' ve
1.1.2

- DESCRIPTION file.
- User guides, package vignettes and other documen n.
- Package NEWS.

Help Pages

Figure 3.11: You can consult the vignettes of installed packages.

Once you installed a package, you have to load it before you can use it. To load packages you use the `library()` function:

```r
library(dplyr)
library(janitor)
# and so on...
```

If you only need to use one single function once, you don't need to load an entire package. You can write the following:

```r
dplyr::full_join(A, B)
```

using the `::` operator, you can access functions from packages without having to load the whole package beforehand.

It is possible and easy to create your own packages. This is useful if you have to write a lot of functions that you use daily. We will lean about that, in Chapter 10.

3.9 Exercises

Exercise 1

Change the look and feel of RStudio to suit your tastes! I personally like to move the console to the right and use a dark theme. Take some 5 minutes to customize by clicking on *Tools* and then *Global options*. Browse through all the options.

4 Conclusion

5 References