Getting Started with R for Economists

Priyanga Dilini Talagala

2025-01-06

Table of contents

Preface

R is a free and powerful software environment for statistical computing and data visualization. It is widely used in academia and industry for data analysis and research. As one of the top programming languages for data science, R provides a variety of tools for statistical modeling, computing, and visualization.

Since empirical research is essential in economics, programming skills are crucial for conducting real-world data analysis. This textbook will introduce you to R and help you develop fundamental data science skills.

About This Book

This textbook is designed for beginners, providing a strong foundation in R for economic research. It focuses on **the tidyverse** ecosystem, a collection of R packages that provide a simple yet powerful approach to data analysis. You will learn how to use tidy tools to manage and analyze data efficiently, covering the entire lifecycle of a data science project.

1 Introduction to R and RStudio

1.1 Installing R and Rstudio

- **Step 1:** First download R freely from the Comprehensive R Archive Network (CRAN) https://cran.r-project.org/. (At the moment of writing, R 4.4.2 is the latest version. Choose the most recent one.)
- **Step 2:** Then install R Studio's IDE (stands for integrated development environment), a powerful user interface for R from [https://posit.co/download/rstudio-desktop/](https://posit.co/download/rstudio-desktop/. Get the Open Source Edition of RStudio Desktop. RStudio allows you to run R in a more user-friendly environment.
  - You need to install **both** R and Rstudio to use RStudio.
  - If you have a pre-existing installation of R and/or RStudio, I highly recommend that you reinstall both and get as current as possible.
- **Step 3:** Then open **Rstudio**.

1.1.1 Posit Cloud

- In 2022, RStudio changed its corporate name to Posit with the aim of expanding its focus beyond R to include users of Python and Visual Studio Code.
- If you don't want to download or install R and R Studio, you can use RStudio on Posit Cloud (https://posit.cloud/) for free.

1.2 RStudio layout

The RStudio interface consists of four panes: See Figure 1)

1. **Source pane**
2. **Console pane**
3. **Environment pane**, containing the Environment, History, Connections, Build, and Tutorial tabs

4. **Output pane**, containing the Files, Plots, Packages, Help, Viewer, and Presentation tabs

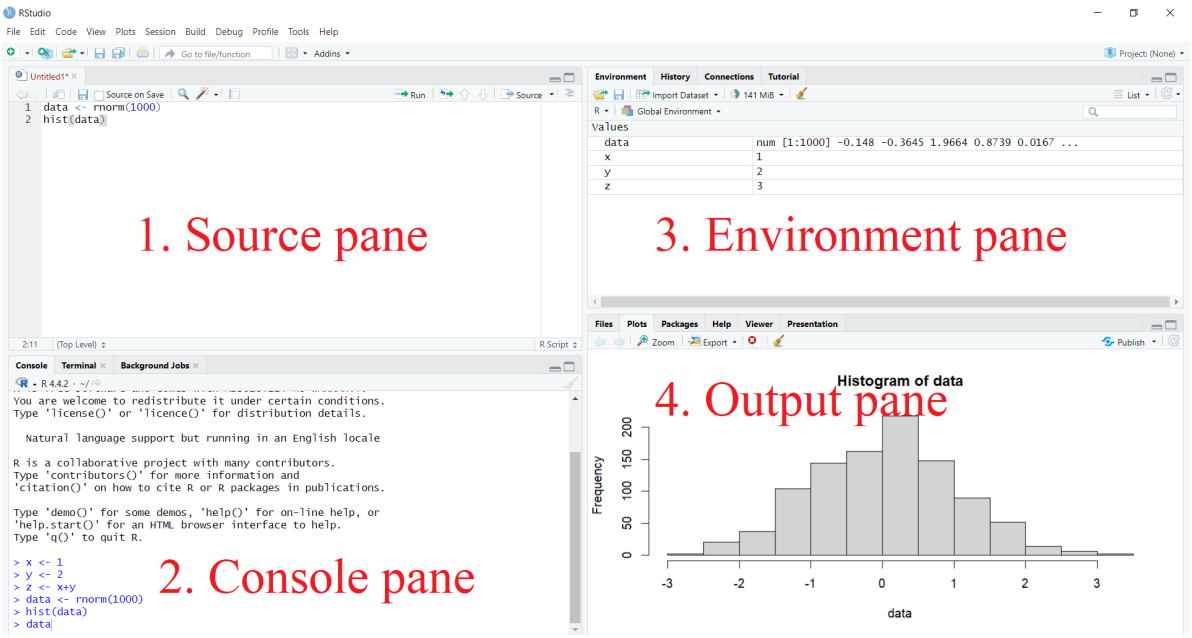

Figure 1.1: RStudio layout

1.2.1 Console Pane

- This is where you type and execute all your R commands.
- You can enter R commands after the '>' prompt, and R will process and execute them.
- This is the most essential window, as it is where R performs computations and executes your instructions.

1.2.2 Source pane

- In this window, a collection of commands (scripts) can be edited and saved.
- If this window is not visible, you can open it via File → New File → R Script.
- Simply typing a command in the Source pane is not enough; it must be sent to the Console before R executes it.
- To run a line from the Source pane, place your cursor on the desired line or select multiple lines to execute, then click Run or press CTRL + ENTER to send them to the Console pane.

- Make sure to save the 'Untitled1' file as a `*.R` script.

1.2.3 Environment pane

- This window contains multiple tabs: Environment, History, Connections, Build, and Tutorial.

The Environment tab displays all active objects.

- For data frames, clicking the grid symbol opens the full data frame in the Source pane.
- The History tab shows previously typed commands.
- To send a command in the history tab to the Source pane, select it and click the "To Source" icon, or click "To Console" to execute it in the Console.

1.2.4 Output Pane

- This window contains multiple tabs: Files, Plots, Packages, Help, Viewer, and Presentation.
- It allows you to open files, view plots (including previous ones), install and load packages, access help functions, and display web content such as Shiny apps and Quarto-generated web pages.

Now you are familiar with the layout. Let's begin with R basics.

1.3 Installing an R Package

- The primary source for R packages is CRAN (Comprehensive R Archive Network).
- Packages can be installed using the `install.packages()` function in R.
- To install a single package, pass its name as the first argument to `install.packages()`.
- The following code installs the tidyverse package from CRAN:

```
install.packages("tidyverse")
```

- This command downloads and installs the `tidyverse` package from CRAN.
- Any dependencies required by the package will also be downloaded and installed.

- Installing the tidyverse package may take several minutes, but you only need to do this once. Think of it like installing a mobile app—you install it once on your smartphone and can use its features until a new version is released, at which point you may need to update it

1.3.1 Alternative way to install R packages in Rstudio

- An alternative way to install R packages is through the Packages tab in the Output Pane.
- Navigate to the Packages tab in the Output Pane and click Install.
- Under "Install from," select "Repository (CRAN)".
- In the Packages field, enter the name of the package you want to install.
- To install multiple packages at once, separate the package names with commas.
- Finally, click Install.

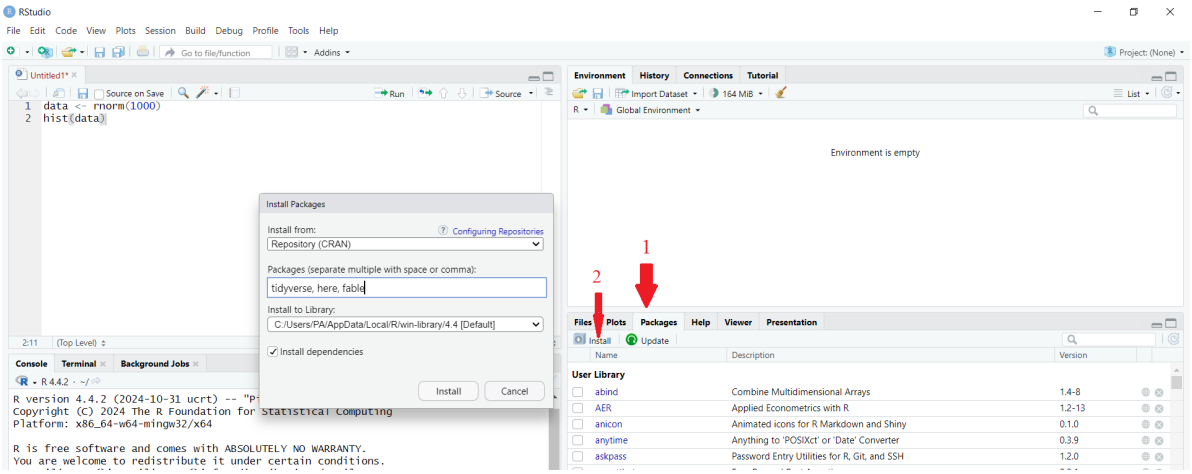

Figure 1.2: Alternative way to install R packages in Rstudio

1.4 Loading an R Package

- Installing a package does not automatically make it available for use; you must load it. It's like a mobile app—you need to open it to access its functionalities.
- The `library()` function is used to load installed packages into R.
- To load the tidyverse package, use:

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

- **Note: Do not put the package name in quotes when using library().**
- Some packages display messages when loaded, while others do not.

1.5 Getting Started with R

For a detailed introduction to R, refer to:

An Introduction to R: https://cran.r-project.org/doc/manuals/R-intro.pdf

2 R Programming Basics

- R has two main parts: data structures and functions.
  - Data structures help store and organize data so it can be used efficiently.

```
income <- c(1000, 4000, 3400, 9700, 9800)
income
```

```
[1] 1000 4000 3400 9700 9800
```

- Functions tell R what to do.

```
mean(income)
```

```
[1] 5580
```

```
summary(income)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1000    3400    4000    5580    9700    9800
```

2.1 Data structures

- R's basic data structures can be grouped by:
  - Their dimensions (1D, 2D, or multi-dimensional).
  - Whether they store one type of data (homogeneous) or different types of data (heterogeneous).
- This gives us five common data structures in R:
  - Homogeneous (same type of data): Vectors, Matrices, Arrays
  - Heterogeneous (different types of data): Data Frames, Lists

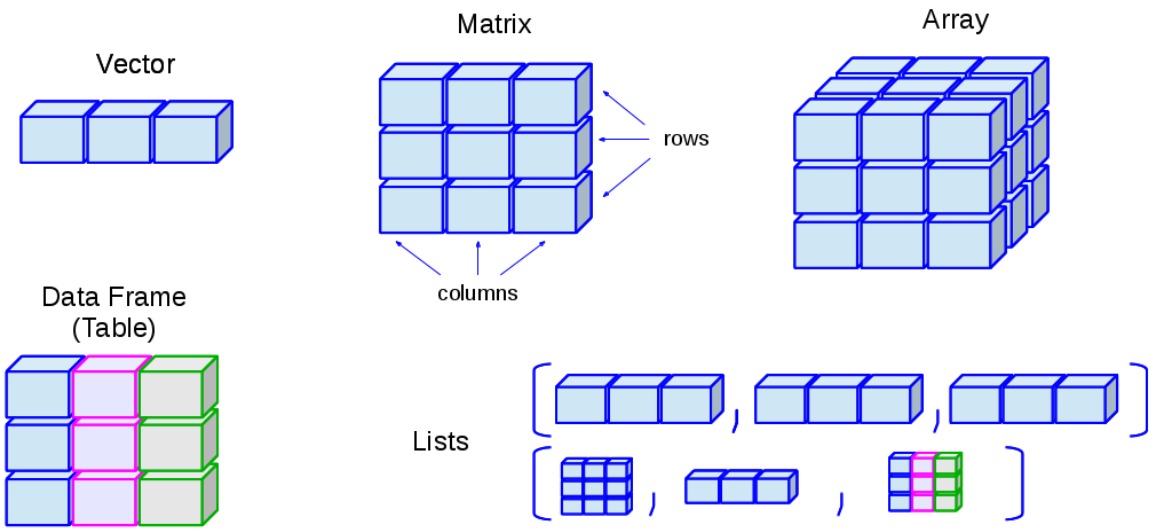

Figure 2.1: Data Structure Types. Image Source: http://venus.ifca.unican.es/Rintro/dataStruct.html

- Among these, vectors and data frames are the most commonly used data structures in practical applications.

2.1.1 Vectors

Creating vectors

Syntax

```
vector_name <- c(element1, element2, element3)
```

Example

```
x <- c(5, 6, 3, 1, 100)
x
```

```
[1]   5   6   3   1 100
```

Combine two vectors

```
p <- c(1, 2, 3)
p
```

```
[1] 1 2 3
```

```
q <- c(10, 20, 30)
q
```

```
[1] 10 20 30
```

```
r <- c(p, q)
r
```

```
[1]  1  2  3 10 20 30
```

Vector with charactor elements

```
countries <- c("Sri Lanka", "Afghanistan", "Bangladesh", "Bhutan", "India", "Iran", "Maldives

countries
```

```
[1] "Sri Lanka"   "Afghanistan" "Bangladesh"  "Bhutan"      "India"
[6] "Iran"        "Maldives"    "Nepal"       "Pakistan"
```

Logical vector

```
result <- c(TRUE, FALSE, FALSE, TRUE, FALSE)
result
```

```
[1]  TRUE FALSE FALSE  TRUE FALSE
```

Simplifying vector creation

- `rep` is a function in R that repeats the values in a vector

```
id <- 1:10
id
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
treatment <- rep(1:3, each=2)
treatment
```

```
[1] 1 1 2 2 3 3
```

Vector operations

```r
x <- c(1, 2, 3)
y <- c(10, 20, 30)
x+y
```

```
[1] 11 22 33
```

```r
p <- c(100, 1000)
x+p
```

```
Warning in x + p: longer object length is not a multiple of shorter object
length
```

```
[1]  101 1002  103
```

2.2 Data Frames

Required R package to deal with data frames

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

Create a tibble

- A tibble is an improved version of a traditional data frame in R, designed to be more user-friendly and consistent.
- It is part of the `tidyverse` package and makes it easier to view and analyze data, especially when working with large datasets.

Lets consider the following example

GDP per capita (current US$) - South Asia

Data Source : https://data.worldbank.org/

Country	Most Recent Year	Most Recent Value
Afghanistan	2023	415.7
Bangladesh	2023	2,551.0
Bhutan	2023	NA
India	2023	2,480.8
Maldives	2023	12,530.4
Nepal	2023	1,377.6
Pakistan	2023	1,365.3
Sri Lanka	2023	3,828.0

```
country <- c("Afghanistan", "Bangladesh", "Bhutan", "India", "Maldives",
             "Nepal", "Pakistan", "Sri Lanka")
year <- c(rep(2023,8))
value <- c(415.7, 2551.0, NA, 2480.8, 12530.4, 1377.6, 1365.3, 3828.0)

final <- tibble(Country = country, Recent_Year = year, Value = value )
final
```

```
# A tibble: 8 x 3
  Country     Recent_Year  Value
  <chr>             <dbl>  <dbl>
1 Afghanistan        2023   416.
2 Bangladesh         2023  2551
3 Bhutan             2023    NA
4 India              2023  2481.
5 Maldives           2023 12530.
6 Nepal              2023  1378.
7 Pakistan           2023  1365.
8 Sri Lanka          2023  3828
```

2.3 Functions in R

- In R, functions are blocks of code that perform specific tasks.
- They take input values (called arguments), process them, and return an output.
- Functions help automate repetitive tasks and make code more efficient.
- There are two main types of functions in R:
  - Built-in functions – Predefined in R (e.g., mean(), sum(), rep()).

```
summary(final$Value)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  415.7  1371.5  2480.8  3507.0  3189.5 12530.4       1
```

```
- User-defined functions - Created by users for specific needs using function().
```

2.4 Subsetting

```
final
```

```
# A tibble: 8 x 3
  Country     Recent_Year  Value
  <chr>             <dbl>  <dbl>
1 Afghanistan        2023   416.
2 Bangladesh         2023  2551
3 Bhutan             2023    NA
4 India              2023  2481.
5 Maldives           2023 12530.
6 Nepal              2023  1378.
7 Pakistan           2023  1365.
8 Sri Lanka          2023  3828
```

```
final[1, 1]
```

```
# A tibble: 1 x 1
  Country
  <chr>
1 Afghanistan
```

```
final[, 1]
```

```
# A tibble: 8 x 1
  Country
  <chr>
1 Afghanistan
2 Bangladesh
3 Bhutan
4 India
5 Maldives
6 Nepal
7 Pakistan
8 Sri Lanka
```

```
final[1, ]
```

```
# A tibble: 1 x 3
  Country     Recent_Year Value
  <chr>             <dbl> <dbl>
1 Afghanistan        2023  416.
```

```
final$Country
```

```
[1] "Afghanistan" "Bangladesh"  "Bhutan"      "India"       "Maldives"
[6] "Nepal"       "Pakistan"    "Sri Lanka"
```

2.5 Help file for Built-in functions

- To access the help file for a built-in function in R, you can use either `?` or the `help()` function.
- Running either of these commands will open the help page for the specified function.

= For example, using `?summary` or `help(summary)` will display the help file for the summary function, which is part of the base R package.

```
?summary

# or

help(summary)
```

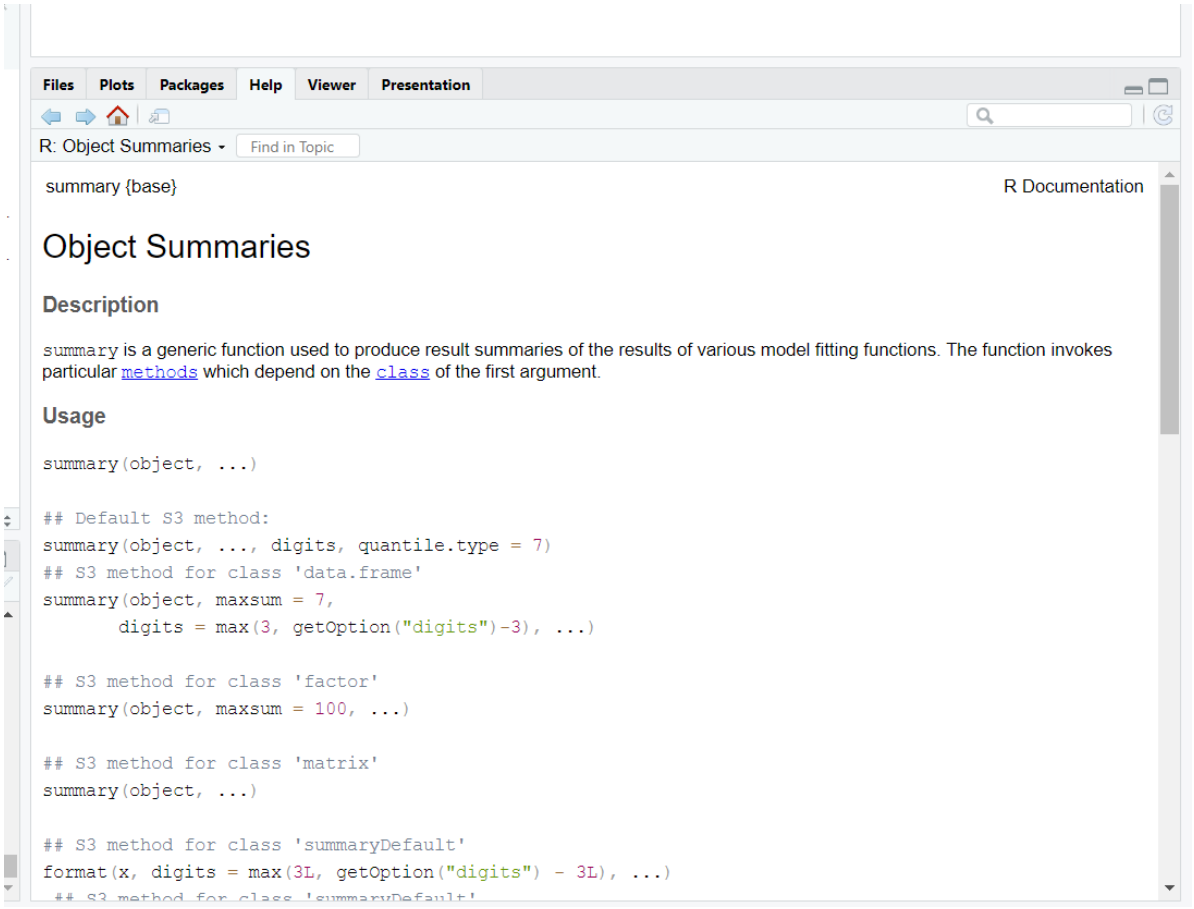

Figure 2.2: Help Page

2.6 Commenting

```
mean(final$Value) # Calculate the average GDP value
```

```
[1] NA
```

When you go to the help page for the `mean()` function, you'll find that by default, the `na.rm` argument is set to `FALSE`. This means that if the data contains missing values (NA), they will be included in the calculation, and the result will also be NA. To calculate the mean while ignoring missing values, you should set `na.rm = TRUE`.

```r
mean(final$Value, na.rm = TRUE) # Calculate the average GDP value
```

```
[1] 3506.971
```

2.7 Pipe operator (|>)

- The pipe operator (`|>`) in the base R package helps improve the readability of your code.
- It takes the output of one function and passes it directly into another function as an argument, making the steps in your data analysis more connected.
- Instead of using nested function calls like `function(first_input, other_inputs)`, you can write the same command in a simpler format: `first_input |> function(other_inputs)`.

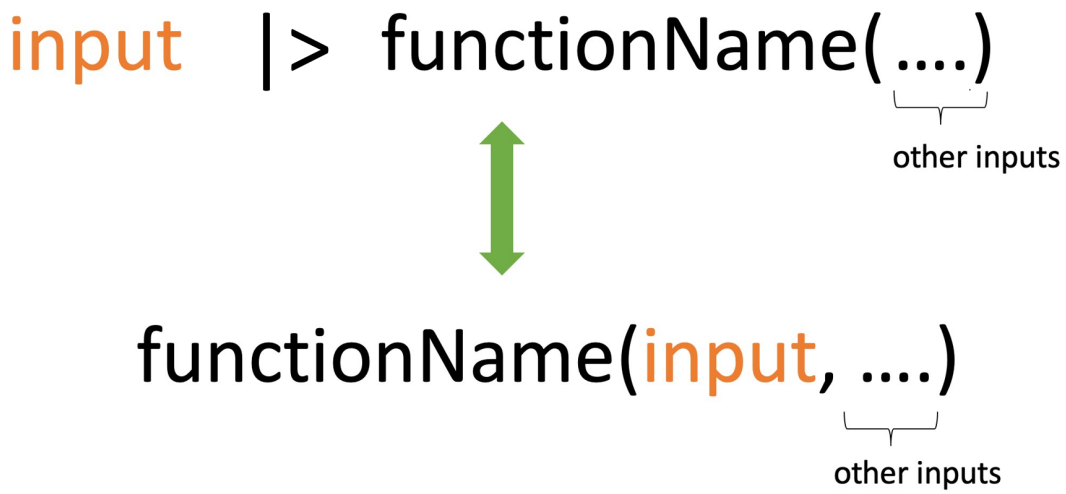

Figure 2.3: Pipe Operator

- Therefore, using the pipe operator allows you to chain multiple operations together in a way that is easier to read and understand compared to nested function calls.

```r
# Nested function call
mean(final$Value, na.rm = TRUE)
```

```
[1] 3506.971
```

```r
# Using pipe operator)
final$Value |> mean(na.rm=TRUE)
```

```
[1] 3506.971
```

3 Scripts and Rstudio Projects

So far, we've been using the console to run code. One problem with the R console is that once you close the project, you won't be able to access the previous code you ran unless you save the workspace. This is where the script editor comes in.

3.1 Working with R Scripts

- To open a script file, go to File -> New File -> R Script.
- You can use a script file to save your code so you don't have to re-type everything when you start a new R session.
- Just typing a command in the script file isn't enough. It must be sent to the Console before R can execute it.
- To run a line of code from the script, place your cursor on the line (or select multiple lines), then click Run or press CTRL + ENTER to send them to the Console.
- Remember to save your script with the *.R extension and give it a suitable name, not as 'Untitled1'.

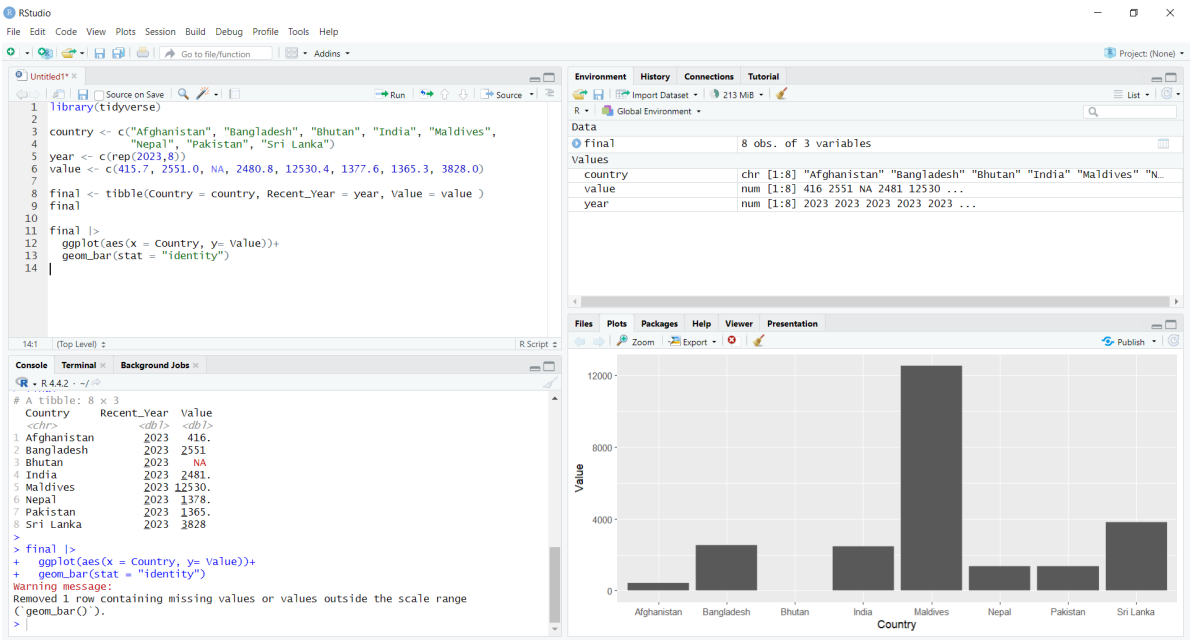

Figure 3.1: Working with R script files

3.2 Working with RStudio Projects

When you start working on large projects, you'll need to manage multiple files, such as input data, script files, and figures. Being organized is very important to manage your work efficiently. This is where RStudio projects help. RStudio projects allow you to keep all the documents related to a project in one folder.

To create an RStudio project, go to `File -> New Project -> New Directory -> New Project`, and then choose a suitable name for the folder to store your work. You also need to pick a location to save your project by setting the "Create project as subdirectory of" option. In the following example, I named my project "Test" and selected the Desktop to store it.

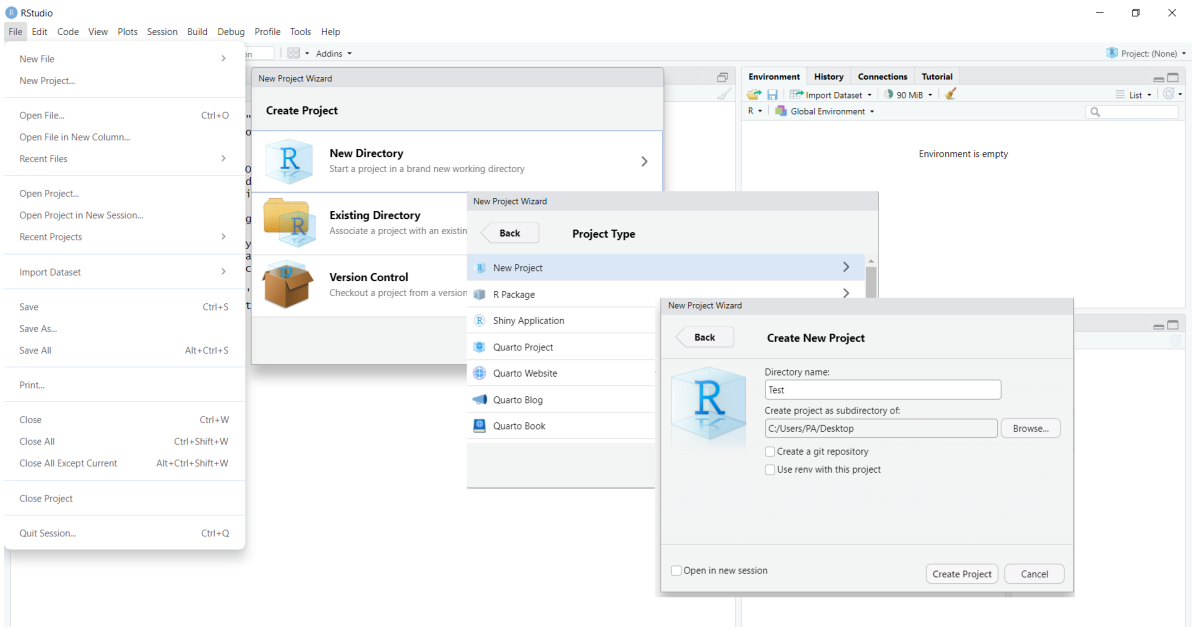

Figure 3.2: Working with RStudio Projects

Once this is done, I now have a new RStudio project called "Test" on my Desktop. This project looks like a regular folder where I can save all my work. If you open the folder, you will see a `.Rproj` file (R Project file). In my example, since I saved my project as "Test," the R project file is called "Test.Rproj." Inside this folder, you can create subfolders, like one for figures called `figures`, another for raw data files called `data`, and another for your R script files called `scripts`.

Let's say you're done working on the project for the day. Now, close the project and quit RStudio.

The next day, when you come back to the project, locate the folder (in my case, it's on the Desktop), open it, and double-click the .Rproj file. This will reopen the project, and you'll pick up right where you left off. You'll have access to all the files you've saved inside the RStudio Project.

The only things you can't retrieve are the commands you typed directly into the console and any unsaved outputs, like summary statistics or figures that were not saved.

But script files at least allow you to save time by preventing you from retyping commands or complex code you've already written in previous sessions. They're better than typing directly into the console.

4 Reproducible Reporting with Quarto

5 Data Import and Export

6 Data Wrangling

7 Data Visualization

8 Introduction to Statistical Modelling

References

R for Data Science (2e) : https://r4ds.hadley.nz/workflow-scripts.html

Advanced R: https://adv-r.hadley.nz/