Introduction to R for Data Analysis

Paul Riesthuis

2025-12-09

Table of contents

Preface **4**

1 R and Rstudio environment **6**
1.1 R and Rstudio . . . 6
1.2 Coding . . . 7
1.2.1 Example 1: Variable with single number . . . 8
1.2.2 Example 2: Variable with multiple numbers . . . 8
1.2.3 Example 3: row vs column vectors . . . 8
1.2.4 Example 4: Dataframes . . . 9
1.2.5 Example 5: **Accessing and Subsetting Data** . . . 9
1.2.6 Example 6: **Descriptive statistics** . . . 12
1.2.7 Summary and conclusion . . . 13

2 Packages **14**
2.1 Installing and loading packages . . . 14
2.1.1 **Installing a package from CRAN** . . . 14
2.2 **Installing packages from GitHub** . . . 15
2.2.1 **Step 1: Install and load devtools** . . . 15
2.2.2 **Step 2: Install a package from GitHub** . . . 15
2.3 **Loading a package** . . . 16
2.4 **Using a package** . . . 16
2.4.1 **Data manipulation and cleaning** . . . 16
2.4.2 **Data importing** . . . 16
2.4.3 **Data visualization** . . . 16
2.4.4 **Data analysis and statistics** . . . 17
2.4.5 Simulation . . . 17
2.4.6 Power analysis . . . 17
2.4.7 **Reporting and reproducibility** . . . 17
2.5 Calling functions from a package . . . 17
2.6 **Getting help for a package** . . . 18
2.6.1 Get help for the whole package: . . . 18
2.6.2 Help for a specific function: . . . 18
2.6.3 Search for function names or keywords: . . . 19
2.7 **Summary** . . . 19

3 Real dataset **20**

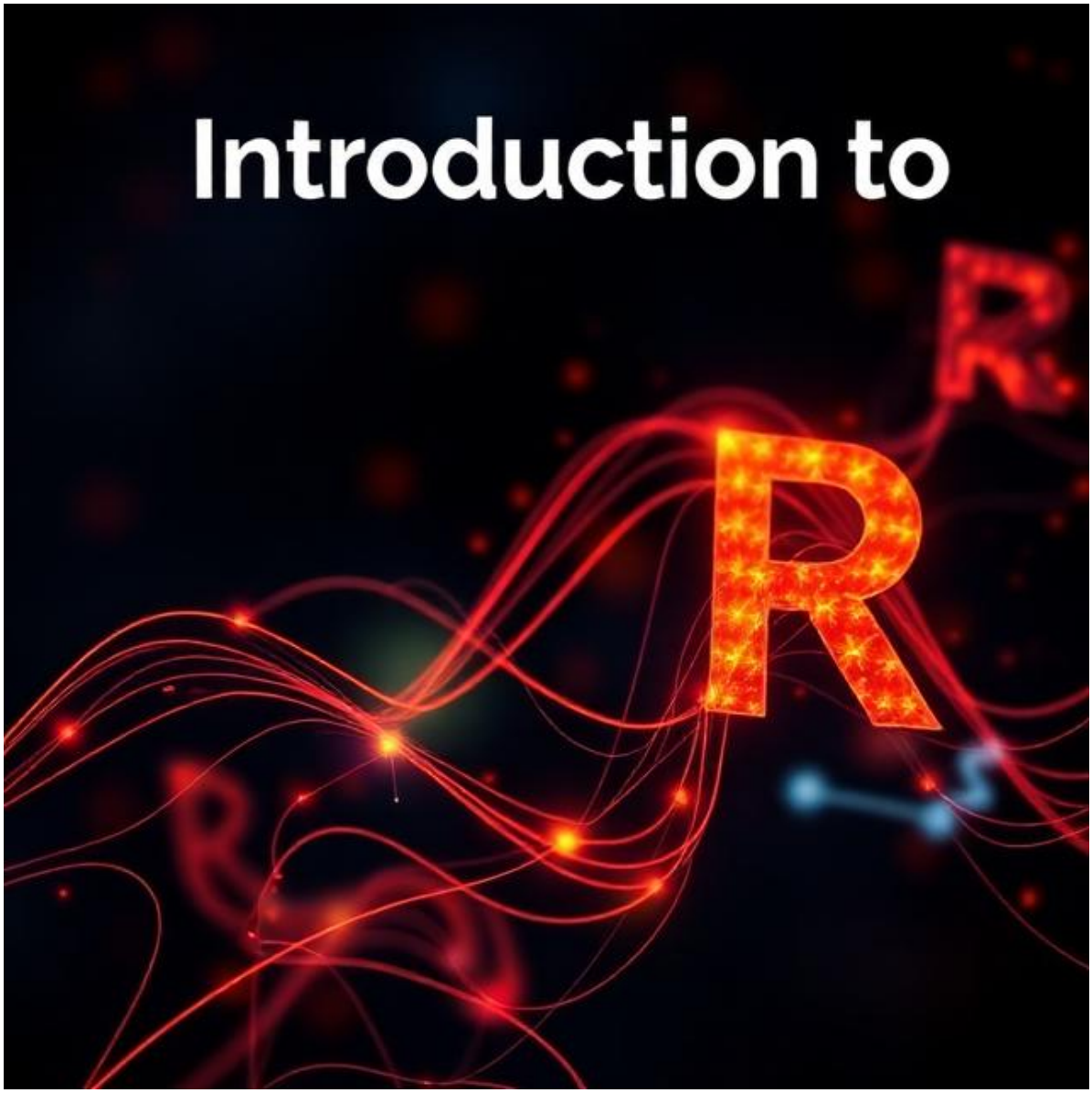
References **21**

Preface

This book is designed to serve as an accessible and practical introduction to **R**, aimed primarily at researchers and students who are new to the language. It begins with the fundamentals, providing clear guidance on how to get started, and gradually introduces more applied examples that are particularly relevant to psychological research.

With the recent rise of **large language models** such as ChatGPT, many coding tasks have become easier and more efficient. Throughout this book, I offer practical recommendations on how researchers can responsibly and effectively use these tools to support coding, data analysis, and problem-solving, without sacrificing scientific rigor or understanding. This book is also made with help of large language models.

A central focus of the book is to equip readers with the skills to conduct analyses that are frequently encountered in psychological research. Each chapter provides step-by-step examples, best practices, and insights into common challenges, so that readers can confidently apply R to their own research questions.



I hope this book serves as a helpful guide for anyone looking to develop a solid foundation in R, while also embracing modern tools that enhance productivity and reproducibility in research.

You can cite this resource as:

Riesthuis, P. (2025). ***Introduction to data analyses in R.***

Paul Riesthuis

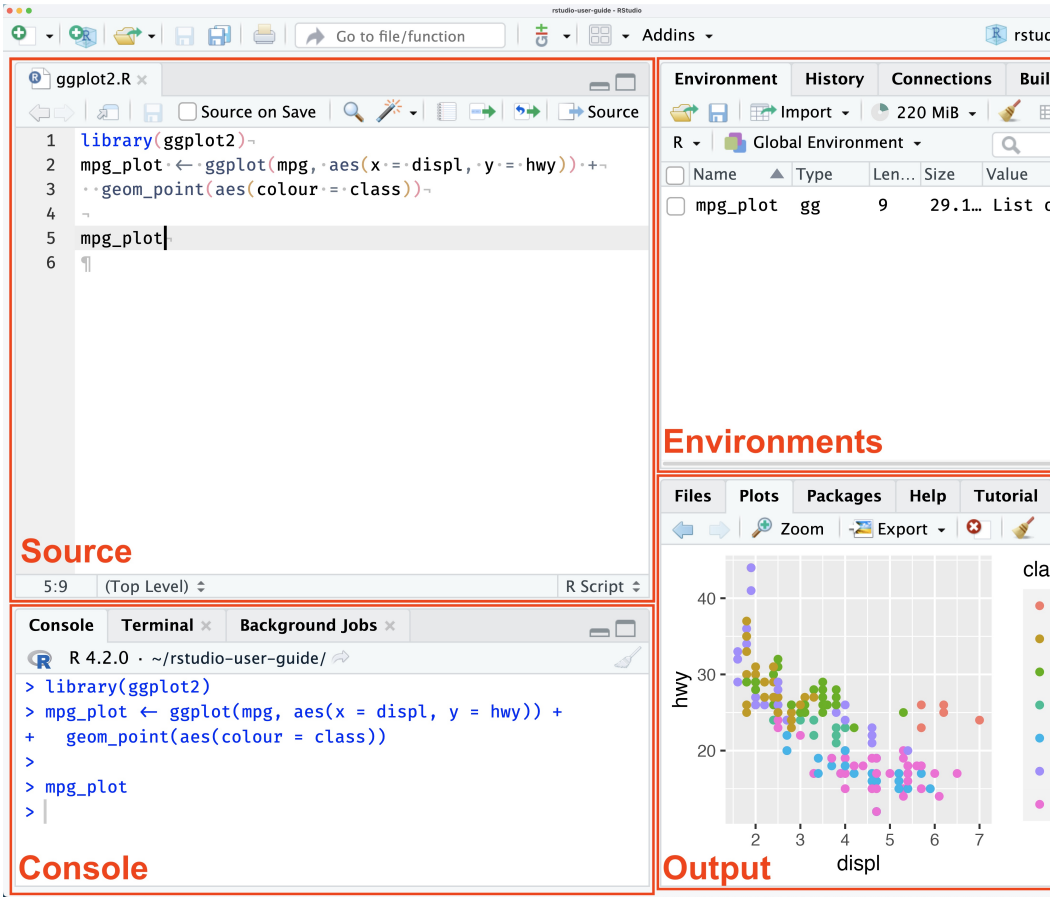
1 R and Rstudio environment

1.1 R and Rstudio

When I first started working with R, I remember being confused by the fact that many people recommended downloading both R and RStudio. I didn't fully understand why RStudio was necessary.

The simplest way to think about it is that RStudio is a convenient interface (an IDE) that makes it easier to write, run, and manage your R code. You use R as the underlying language, and RStudio as the environment that helps you work with that language more effectively.

The Rstudio looks as follows:



The RStudio interface is divided into **four main sections** that help you write, run, and manage your R code efficiently:

1) **Source**

- This is where you write and edit your R scripts or Quarto documents.
- Think of it like a "text editor" for your code.
- You can save your scripts, organize your work, and run lines or sections of code from here.

2) **Console**

- This is where R actually executes your code.
- You can type commands directly and see immediate results.
- It's perfect for testing small pieces of code or quickly checking something.

3) **Environment**

- This section shows all the variables, data frames, and objects currently stored in your R session.
- It helps you keep track of what data and results you have available.
- Some tabs also show your command history, so you can easily reuse previous commands.

4) **Output**

- This section displays plots, graphs, and other outputs generated by your code.
- It also includes tabs for files, packages, help, and viewer windows.
- You can inspect your results visually and navigate your project files from here.

Summarized:

A simple way to remember the sections:

1. **Source** → write code
2. **Console** → run code
3. **Environment** → see your variables
4. **Output** → see results and plots

1.2 Coding

So, in the source section we can actually code variables. The possibilities here are endless. That is, we can code a single number to be assigned to a variable name "y" but also conduct entire structural equation models, multilevel models, etc. Let's start with some basics and most essential.

1.2.1 Example 1: Variable with single number

- In R we use "<-" instead of "=" to assign variables.
- You can copy code always by clicking the icon in the upper right of the code chunk.

```
x <- 7
x
```

```
[1] 7
```

Tip: You can run code by selecting a line and pressing **Ctrl + Enter** (Windows) or **Cmd + Enter** (Mac).

1.2.2 Example 2: Variable with multiple numbers

This is also known as a **single row vector**

```
row_vec <- c(1,2,3,4,5)
row_vec
```

```
[1] 1 2 3 4 5
```

1.2.3 Example 3: row vs column vectors

In R, the vectors we create are **row vectors** by default. Sometimes, it's useful to create a **column vector** (a vector arranged vertically), especially when working with matrices or data frames.

```
col_vec <- matrix(c(1, 2, 3, 4, 5), ncol = 1)
col_vec
```

```
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
```

Explanation:

- **Row vector:** A simple vector created with `c()`; elements are arranged horizontally when printed.
- **Column vector:** Created using `matrix()` with `ncol = 1`; elements are arranged vertically.
- Vectors are the **building blocks** for matrices and data frames in R.
- **Tip**: In psychological research, a **column often represents a variable** (e.g., Age, Score) and a **row represents a participant**.

1.2.4 Example 4: Dataframes

A **data frame** is like a table in R. It can store **columns of different types** (numeric, character, logical) and is one of the most common ways to organize data for analysis.

```
df <- data.frame(
Name = c("Alice", "Bob", "Charlie"),
Age = c(25, 30, 22),
Passed = c(TRUE, FALSE, TRUE)
)

df
```

```
     Name Age Passed
1   Alice  25   TRUE
2     Bob  30  FALSE
3 Charlie  22   TRUE
```

Explanation:

- `data.frame()` creates a table-like structures
- Columns can have **different types** (numbers, text, logical values).

1.2.5 Example 5: Accessing and Subsetting Data

- You can access columns of a dataframe with `$`, e.g., `df$Age`.

```
df$Age
```

```
[1] 25 30 22
```

```
df$Passed
```

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

Each column is treated as a vector, so you can perform operations on it just like a normal vector.

- You can access rows using **row indices**, e.g., `df[1, ]`.

```
df[1, ]
```

```
   Name Age Passed
1 Alice  25   TRUE
```

```
df[2, ]
```

```
  Name Age Passed
2  Bob  30  FALSE
```

The syntax is df[row, column]. Leaving the column blank selects all columns (or vice versa).

- You can filter rows using logical conditions. For example, to select all participants who passed:

```
df[df$Passed == TRUE, ]
```

```
     Name Age Passed
1   Alice  25   TRUE
3 Charlie  22   TRUE
```

Some useful logical operators in R:

- == → equal to
- != → not equal to
- > → greater than
- < → less than
- >= → greater than or equal to
- <= → less than or equal to
- & → AND
- | → OR

Example: Select participants who passed or are older than 25:

```
df[df$Passed == TRUE & df$Age > 24, ]
```

```
   Name Age Passed
1 Alice  25   TRUE
```

There are also packages in R that others have created that also perform these data manipulations but also more advanced such as "**dplyr**".

- For example, we can **select** Age or Passed columns:

```
# install.packages(dplyr) # Run the code before without "#" to install the package
library(dplyr) # This is to call the packages

df %>% select(Age)
```

```
  Age
1  25
2  30
3  22
```

```
df %>% select(Passed)
```

```
  Passed
1   TRUE
2  FALSE
3   TRUE
```

- Or we can **filter** only participants who passed the test:

```
df %>% filter(Passed == TRUE)
```

```
     Name Age Passed
1   Alice  25   TRUE
2 Charlie  22   TRUE
```

- Or **select** and **filter** simultaneously for which we need "%>%" (pipe) operator:

```
df %>%
filter(Passed == TRUE) %>%
select(Name, Age)
```

```
      Name Age
1    Alice  25
2 Charlie  22
```

Why dplyr is useful:

- Syntax is **more readable** than base R, especially for beginners.
- You can **chain multiple operations** with %>%, which makes your code look like a step-by-step pipeline.
- Works well for **larger datasets** and more complex filtering.

There are many powerful functions in dplyr, and we'll explore them gradually throughout the book. However, tools like large language models can be extremely helpful when you want to automate your workflow. You can simply describe—in plain language—what you want to do (e.g., "filter rows where Age is above 30 and select only the Name column"), and the model can generate the corresponding R code for you.

1.2.6 Example 6: Descriptive statistics

It is also quite easy to compute **mean, median, standard deviation** or other statistics:

```
mean(df$Age)
```

```
[1] 25.66667
```

```
median(df$Age)
```

```
[1] 25
```

```
sd(df$Age)
```

```
[1] 4.041452
```

```
summary(df)
```

```
    Name                Age           Passed
 Length:3           Min.   :22.00   Mode :logical
 Class :character   1st Qu.:23.50   FALSE:1
 Mode  :character   Median :25.00   TRUE :2
                    Mean   :25.67
                    3rd Qu.:27.50
                    Max.   :30.00
```

There are also packages that will automatically summarise your data and provide additional information such as "**psych**".

```
# install.packages(psych) # Run the code before without "#" to install the package

library(psych)
describe(df)
```

```
Warning in FUN(newX[, i], ...): no non-missing arguments to min; returning Inf
```

```
Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning -
Inf
```

```
       vars n  mean   sd median trimmed  mad min  max range skew kurtosis   se
Name*     1 3  2.00 1.00      2    2.00 1.48   1    3     2 0.00    -2.33 0.58
Age       2 3 25.67 4.04     25   25.67 4.45  22   30     8 0.16    -2.33 2.33
Passed    3 3   NaN   NA     NA     NaN   NA Inf -Inf  -Inf   NA       NA   NA
```

1.2.7 Summary and conclusion

In this introductory chapter, we explored the basic structure and functionality of R and RStudio. You learned how the main interface is organized, how to write and run simple pieces of code, and how to create and manipulate some of the most fundamental data structures in R, such as vectors and data frames. We also introduced essential tools for inspecting and filtering data, both using base R and the more readable dplyr syntax. These skills form the foundation of nearly all data analysis work in R. In the next chapter, we will build on this foundation by working with a real dataset—importing it, exploring it, and preparing it for analysis—so you can see how these tools apply to actual research scenarios.

2 Packages

In the previous chapter, we explored the RStudio interface, learned how to write and run simple pieces of R code, and practiced creating and manipulating basic data structures such as vectors and data frames. These skills form the foundation for working in R.

In this chapter, we take the next step: we will learn how to **install packages**, **load them**, and **use them in your analyses**. Packages are essential in R because they extend the functionality of the language and make data manipulation, visualization, and analysis much easier.

By the end of this chapter, you will be able to:

- Install and load R packages
- Navigate packages and explore their functions
- Get help and documentation for packages

The skills you learn here will be applied throughout the rest of the book, as you work with more complex datasets and analyses.

2.1 Installing and loading packages

Base R already includes many useful functions, such as creating variables, working with vectors, and building data frames. However, much of R's power comes from the thousands of **additional packages** created by researchers and developers. These packages make tasks like data manipulation, visualization, modeling, or reporting much easier.

Earlier, we briefly introduced the `dplyr` package for data manipulation. Let's now formally look at how to install and load packages.

2.1.1 Installing a package from CRAN

CRAN (the Comprehensive R Archive Network) is the main place where R packages are stored. To install a package from CRAN, you run:

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

- You only need to install a package **once** on your computer (unless you update or reinstall R).

2.2 Installing packages from GitHub

Not all packages are available on CRAN. Many developers share their newest or experimental work on **GitHub**. To install these packages, you first need the `devtools` package.

2.2.1 Step 1: Install and load devtools

```r
install.packages("devtools")
library(devtools)
```

2.2.2 Step 2: Install a package from GitHub

The syntax is:

```r
devtools::install_github("username/repository")
```

For example, I created a package called *ROCpower* and uploaded it to GitHub, you could install it like this

```r
devtools::install_github("PaulRiesthuis/ROCpower")
```

GitHub installations are useful when:

- a package is very new
- a feature is only available in the development version
- or the author hasn't uploaded it to CRAN yet

2.3 Loading a package

After installation, you need to *load* the package each time you start a new R session:

```
library(dplyr)
```

Once a package is loaded, all of its functions become available for use. Riesthuis (2024)

2.4 Using a package

Once a package is loaded with `library()`, you can immediately start using its functions. Most packages are organized around a specific purpose. Some of the most used packages are:

2.4.1 Data manipulation and cleaning

- **dplyr** – grammar of data manipulation (filtering, selecting, grouping)
- **tidyr** – reshaping and organizing data (pivoting, separating, unnesting)
- **stringr** – working with text data (string detection, replacement, cleaning)
- **tidyverse** – Loading `tidyverse` loads `dplyr`, `tidyr`, `stringr`, and other core packages at once.

2.4.2 Data importing

- **readr** – fast reading of CSV and text files
- **readxl** – import Excel files (e.g., `.xls` and `.xlsx`)
- **haven** – import SPSS, Stata, and SAS data
- **rvest** – downloading or scraping data from the web

2.4.3 Data visualization

- **ggplot2** – the most widely used visualization system in R
- **patchwork** – combine multiple ggplots into one figure
- **ggthemes** – additional themes and styles for ggplot2
- **plotly** – make interactive plots based on ggplot2

2.4.4 Data analysis and statistics

- **lme4** – mixed-effects models
- **psych** – descriptive statistics and psychometric tools
- **negligible** – equivalence tests
- **car** – common regression diagnostics and hypothesis tests
- **metafor** – meta-analyses
- **broom** – convert model outputs into tidy data frames

2.4.5 Simulation

- **faux** – simulate data for typical experimental designs
- **simstudy** – flexible framework to simulate data for studies, including complex dependencies
- **mirt** – simulate item response theory (IRT) data for psychometrics
- **MASS** – includes functions for statistical methods and simulation for multivariate normal data

2.4.6 Power analysis

- **Spower** – conduct simulation based power analyses based on G*Power
- **ROCpower** – power analyses for Receiver Operating Characteristic curves
- **pwr** – traditional analytical power calculations for t-tests, ANOVA, correlations, proportions, etc.

2.4.7 Reporting and reproducibility

- **knitr** – knitting R Markdown and Quarto documents
- **rmarkdown** / **quarto** – reproducible reports and documents
- **janitor** – cleaning column names and simple frequency tables

These are just a few of the most commonly used packages—R has thousands more, and we will introduce additional packages throughout the book as they become useful.

2.5 Calling functions from a package

If a package is loaded, you can simply call its functions by name:

```r
library(dplyr)

df <- data.frame(
  Name  = c("Alice", "Bob", "Charlie"),
  Age   = c(25, 30, 22),
  Passed = c(TRUE, FALSE, TRUE)
)

df %>%
  filter(Passed == TRUE)
```

```
     Name Age Passed
1   Alice  25   TRUE
2 Charlie  22   TRUE
```

If you want to use a function **without loading the entire package**, you can call it with `:::`

```r
dplyr::filter(df, Passed == TRUE)
```

```
     Name Age Passed
1   Alice  25   TRUE
2 Charlie  22   TRUE
```

2.6 Getting help for a package

R includes extensive documentation for every package and every function.

2.6.1 Get help for the whole package:

```r
help(package = "dplyr")
```

2.6.2 Help for a specific function:

```r
?filter
```

2.6.3 Search for function names or keywords:

```
??"mean"
```

2.7 Summary

In this chapter, we learned how to install and load R packages, explored some of the most commonly used packages for data manipulation, visualization, analysis, and reporting, and learned how to access help and call functions. Packages are a core part of R's power, and you will continue to encounter and use many more throughout the book. With these tools in hand, you are ready to move on to working with **real datasets** and applying your skills in practical research analyses.

3 Real dataset

In the previous chapter, we learned how to install, load, and use R packages. These tools will be essential as we begin working with actual datasets.

In this chapter, we take the next step: we will **work with a real dataset in R**, explore and inspect it, and perform some initial data cleaning and analysis. Working with real data is an important milestone because it allows you to apply the skills you've learned so far—creating variables, manipulating data structures, and using packages—in a practical research context.

By the end of this chapter, you will be able to:

- Import data from various sources (e.g., CSV, Excel)
- Inspect and explore the dataset to understand its structure
- Perform basic data cleaning and preparation
- Apply simple data analysis and summaries

This chapter lays the foundation for all subsequent analyses in the book. Once you are comfortable importing and working with real datasets, you will be ready to conduct more advanced analyses and visualizations in later chapters.

References

Riesthuis, Paul. 2024. "Simulation-Based Power Analyses for the Smallest Effect Size of Interest: A Confidence-Interval Approach for Minimum-Effect and Equivalence Testing." *Advances in Methods and Practices in Psychological Science* 7 (2): 25152459241240722. https://doi.org/10.1177/25152459241240722.