R for Novice Programmers (1e)

First Steps with R and RStudio

William Okech

2024-01-01

Table of contents

List of Figures

List of Tables

Welcome

This is the website for the book "R for Novice Programmers" written by William Okech. The goal of this book is to introduce non-programmers or those with very little programming experience to the benefits of the R and RStudio software. The main prerequisites for learners are basic knowledge of computer applications and experience working with files and folders. This book will primarily focus on the basic R concepts that are hardly emphasized, but that may prove difficult for learners new to programming.

Cover image was designed using Canva.

This book was built with Quarto.

Introduction

Why did I write this book?

This book is primarily intended to cater to the needs of individuals who have a desire to learn the basics of programming. I focus on R and RStudio because their capabilities may be relevant to a wide variety of individuals and organizations seeking to perform basic statistical analysis and data visualization. Personally, my skills in R and RStudio were gained via classroom instruction, online tutorials and videos, as well as relevant blog posts. A significant disadvantage of some of these resources is the assumption of prior programming knowledge. To address this, I begin the book with instructions for downloading software, navigating the R and RStudio interfaces, and an overview of the basics of R to decrease the cognitive load on novices.

About the author

The author of this book is a Certified Carpentries instructor and a trainer with the Digital Research Academy. Additionally, the author holds a PhD in Biomedical Engineering and has completed postdoctoral fellowships in vascular biology and infectious diseases. Lastly, the author is passionate about using R and RStudio to generate data-driven visualizations to allow for a more in-depth understanding of public policy issues.

Syllabus

At the end of the book, the student should be able to perform the tasks listed in the syllabus below.

Table 1: Syllabus

Chapter	Title	Date Completed
	Introduction	
1	Overview of R and RStudio	
2	Download and Install R and RStudio	

Chapter	Title	Date Completed
3	Navigating the R and RStudio interfaces	
4	Managing your files and data	
5	Importing data and saving analysis outputs	
6	Basic arithmetic, arithmetic operators, and variables	
7	The primary types of operators in R	
8	Data Types	
9	Vectors	
10	Data Structures (Part I)	
11	Data Structures (Part II)	
12	Handling missing data	
	Conclusion	
	Appendix	

Sample chapter design

Each lesson will follow a pre-described format

i. Questions to be addressed

ii. Learning objectives

iii. Lesson content

iv. Practice exercises

v. Lesson summary

The learners are encouraged to work through each chapter sequentially. For each chapter, the learner should first review the questions to be addressed and learning objectives. Next, the student should read through the lesson content and ensure that the questions and learning objectives sections have been addressed. Finally, the learners should do the practice exercises to reinforce the newly learned concepts and review the lesson summary.

Feedback

Feedback can be provided using a GitHub pull request: Link

Summary

Overall, I believe that this book will increase both the knowledge and confidence levels of novice programmers and allow them to perform basic statistical analysis and simplify everyday computational tasks at home or in their workplaces. In the next chapter, I will provide a basic overview of the R and RStudio ecosystem.

1 Overview of R and RStudio

1.1 Questions

- What is R? How is it related to RStudio?
- Why is R considered a powerful language for statistical computing and data analysis?
- What are some common uses of R in various fields?
- What advantages does R offer over other programming languages for data science tasks?

1.2 Learning Objectives

- Learn about the historical background of R and RStudio.
- Understand the uses and primary advantages of R and RStudio.
- Explore the various applications of R across different industries.

1.3 Lesson Content

1.3.1 Why learn R and RStudio?

Both R and Studio are free, open-source software tools that are widely used for statistical analysis and data visualization. R is a programming language that enables the use of code to analyze data. The primary function of the R language is statistical analysis, and this can be performed directly in the R console. To ease the analysis process and enhance usability, an integrated development environment (IDE), such as RStudio is recommended. The RStudio IDE is a user-friendly interface that allows the learner to manage multiple script files, use the command-line terminal, easily access file inputs and outputs, and review file/analysis history.

The R programming language software was developed by Ross Ihaka and Robert Gentleman in 1993 (published as open-source in 1995) when they were based at the University of Auckland. *Fun fact: R represents the first letter of the first names of the creators.* The software is utilized by individuals working for various organizations, ranging from academic institutions

and healthcare organizations to financial services and information technology companies. In January 2024, the PopularitY of Programming Language (PYPL) Index, which is created by analyzing how often language tutorials are searched on Google, demonstrated that R was the 6th most popular programming language. However, in the same period, the TIOBE index indicated that R was the 23rd most popular language. This may result from different methodologies for developing the rankings. RStudio is an integrated development environment (IDE) for R that was developed by JJ Allaire. This software contains tools that make programming in R easier.

RStudio extends R's capabilities by making it easier to import data, write scripts, and generate visualizations and reports. The company RStudio (now Posit since 2022) was founded in 2009 with the main goal of "creating high quality open-source software for data scientists."

1.3.2 Uses of R and RStudio

i. The R and RStudio console can be used as a complex scientific calculator.

ii. The values of various data types can be assigned to variables using the symbol `<-` or `=`.

iii. Built-in functions can be used to manipulate variables.

iv. Built-in datasets can be accessed internally for analysis.

v. New datasets can be imported, and new functions can be created for custom analysis.

vi. To aid in computational analysis, there exists a large package library (CRAN), as well as a lot of software in development to aid in computational analysis.

1.3.3 Primary advantages of R and RStudio

i. R and RStudio are free and open-source software programs, which makes them accessible to anyone with a computer and an internet connection. This accessibility is key in enabling learners from all socioeconomic levels and geographic regions to have a chance to work with statistical software,

ii. Numerous user communities exist for the R/RStudio software. These communities (listed in the Appendix) provide learning support and assist with technical challenges,

iii. Numerous freely available packages/extensions have been developed by the R and RStudio user communities to facilitate all forms of computational analysis, visualization, and publication. The (CRAN) has packages that contain datasets as well as allow one to perform statistical analysis and data visualization,

iv. R and RStudio allow for reproducible analysis where scripts and workflows can be shared with fellow users, and,

v. The R/RStudio software is cross-platform, which means that it can be used on Linux, Windows, and Mac operating systems.

1.3.4 Applications of R in different industries

i. Bioinformatics and Healthcare: epidemiological studies, clinical trial analysis, and genetic data analysis.

ii. Financial Modelling and Risk Analysis: risk management, algorithmic trading, trading strategies and analysis, time series analysis, and portfolio optimization.

iii. Retail and Marketing: customer analytics, sales forecasting, market research, web analytics, and customer segmentation.

iv. Social Sciences and Humanities: text analysis, surveys and opinion research, social trend analysis, and policy analysis.

v. Statistics and Data Analysis: hypothesis testing, data visualization, regression modelling, and statistical inference.

vi. Environmental Science and Climate Change: forecasting weather patterns, modelling climate change, monitoring pollution levels, and ecological modelling.

1.4 Exercises

As you embark on your R/RStudio learning journey, I have listed (below) a few questions for you to think about before we get started with the lessons.

i. Why do you want to learn R and RStudio?

ii. Do you currently use any other software tools for data analysis and visualization? What are the limitations of these tools?

iii. What are some key differences between R and other statistical programming languages like SAS or SPSS?

iv. What tasks do you hope to accomplish after completing this training?

v. Explore the various R/RStudio communities listed in the appendix and consider joining any one of them. What is the role of the R community in the development and support of R?

vi. Browse some popular R packages (on CRAN or R-Universe) used for different tasks like data visualization and statistical analysis. Pick one package that interests you and read about its capabilities.

1.5 Conclusion

I hope you enjoyed learning about the history of R and RStudio, and have seen the advantages of using these tools for the diverse computational tasks in your fields of practice. Additionally, we discussed the numerous applications of R in various industries. In the next chapter, we will look at how to download and install both R and RStudio on your local computer.

2 Download and Install R and RStudio

2.1 Questions

- How does one install R and RStudio on their personal computer?
- Can RStudio be used online via a cloud-based service?
- Is it possible to work with alternative code editors when using R?
- What steps are involved in installing RStudio?

2.2 Learning Objectives

- Install the R programming language on your local machine.
- Install RStudio as an integrated development environment (IDE) for R on your local machine.
- Create an RStudio Cloud Account on the Posit Website.
- Learn how to use alternative code editors.

2.3 Lesson Content

2.3.1 Introduction

Both the R and RStudio software are required to make full use of the R programming environment. R is the programming language, while RStudio is the integrated development environment (IDE) that has an easy-to-use interface. Here, we will focus on downloading R and RStudio for Windows from the respective websites. The instructions for downloading these two programs on Mac/Linux operating systems are available on the download websites.

2.3.2 Install R

To install R on your personal computer, visit the R Project for Statistical Computing's Comprehensive R Archive Network CRAN. Follow the illustrated steps shown below:

2.3.2.1 Step 1

On the CRAN homepage, select the appropriate version of R for your operating system (Linux, macOS, or Windows) (Figure **??**).

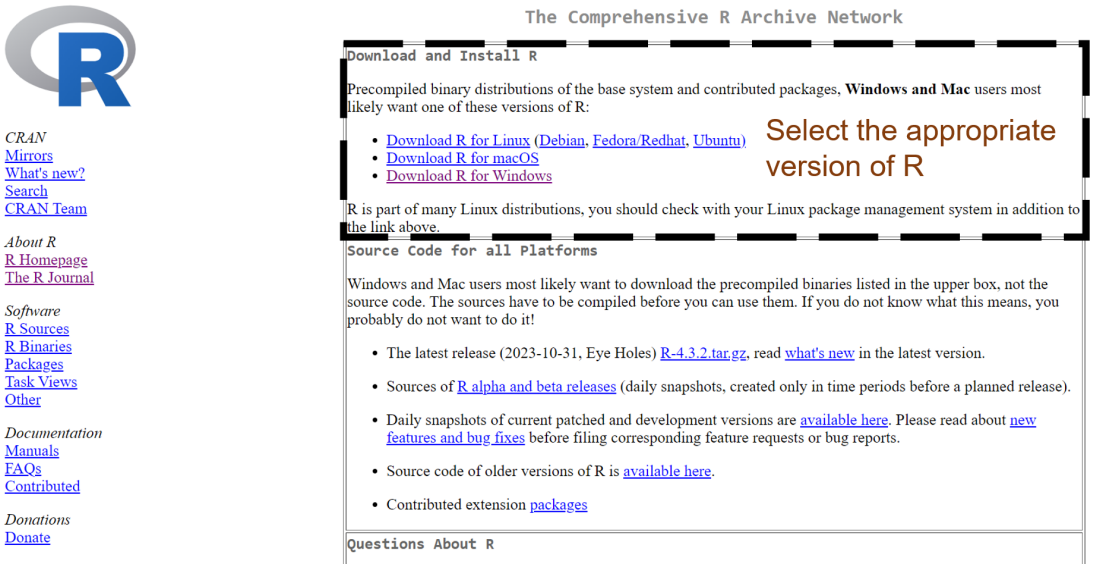


Figure 2.1: The CRAN Homepage (Part 1)

2.3.2.2 Step 2

If installing R for the 1st time, click on "base" (Figure **??**).

2.3.2.3 Step 3

Click on the "Download" link for the latest version of R that is currently available (at the time of this writing, it was R-4.3.2). Once downloaded, run the executable file and wait for the software to be installed (Figure **??**).

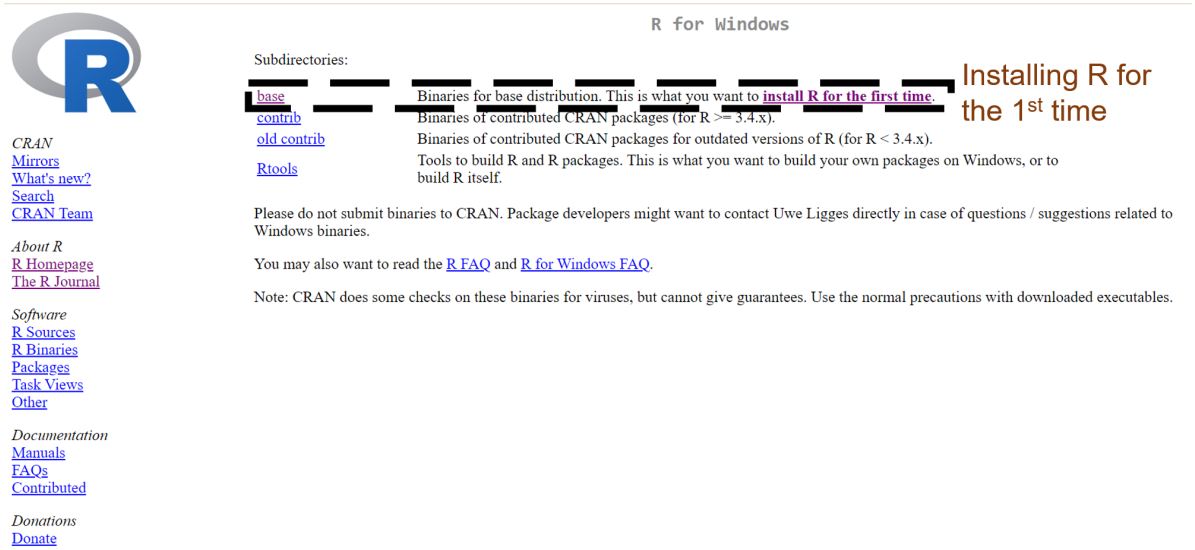


Figure 2.2: The CRAN Homepage (Part 2)

Figure 2.3: The CRAN Homepage (Part 3)

2.3.2.4 Step 4

Once R is installed, open up the R software. The R graphical user interface should be similar to what is depicted below (Figure **??**).

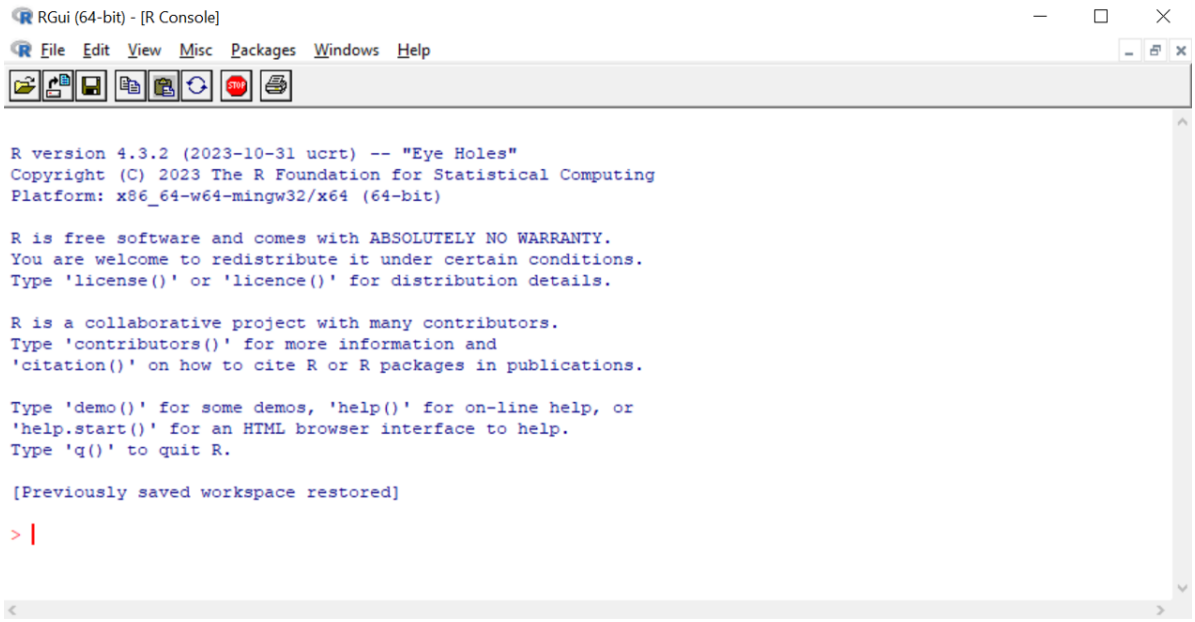


Figure 2.4: The R Console

2.3.3 Install RStudio

The RStudio IDE can be obtained from the "Download" section of the Posit website. Install the software by clicking on the "Download RStudio Desktop for Windows" button and running the downloaded program (Figure **??**).

Once installed, verify that the RStudio software has a similar graphical user interface to the one depicted in the image below (Figure **??**).

2.3.4 Create an RStudio Cloud Account on the Posit website

Posit (formerly RStudio) Cloud lets the user access the RStudio interface from their internet browsers (Figure **??**). Using this option does not require any installation or specific software configuration to be implemented. Posit Cloud offers a free plan for casual users (without the need for a paid plan) and there is no need for dedicated hardware. Additionally, Posit provides a comprehensive guide for first-time users.

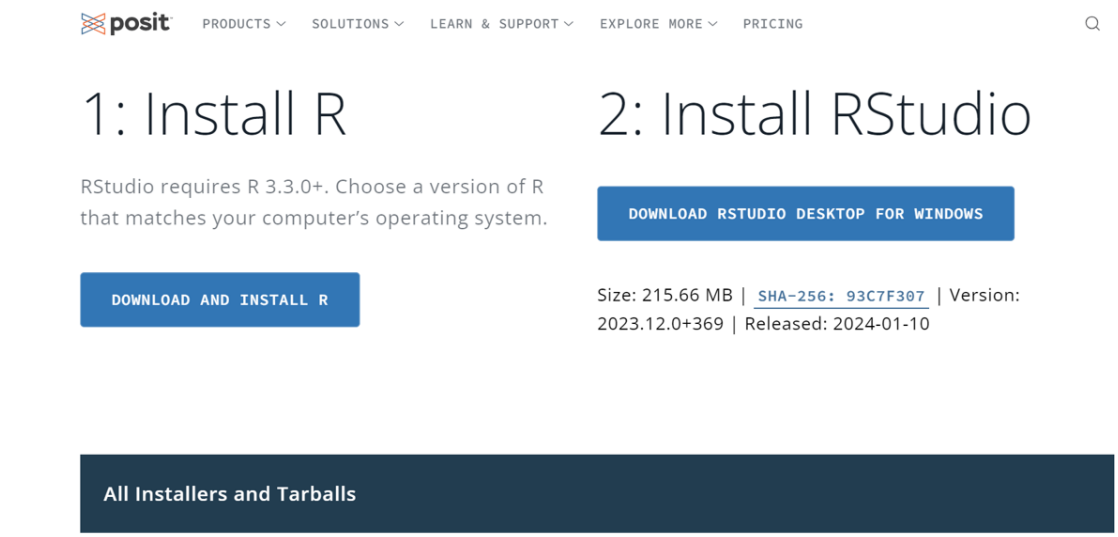


Figure 2.5: The Posit/RStudio Download Page

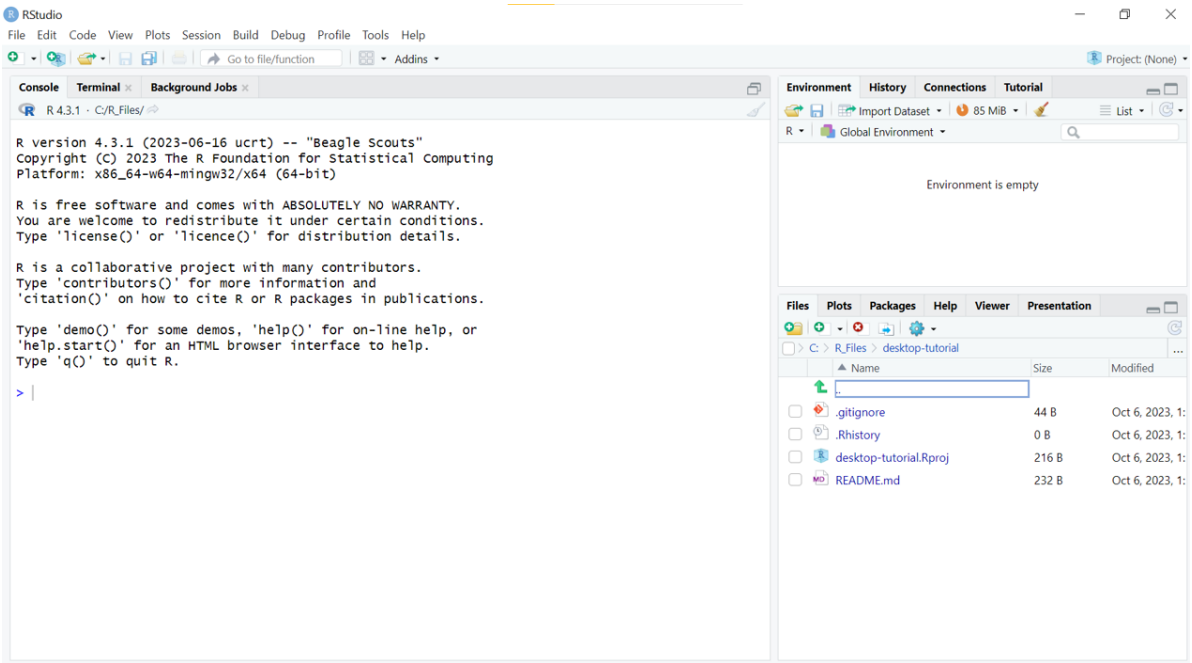


Figure 2.6: The RStudio Console

2.3.4.1 Step 1

Access the Posit Cloud Website (Figure **??**).

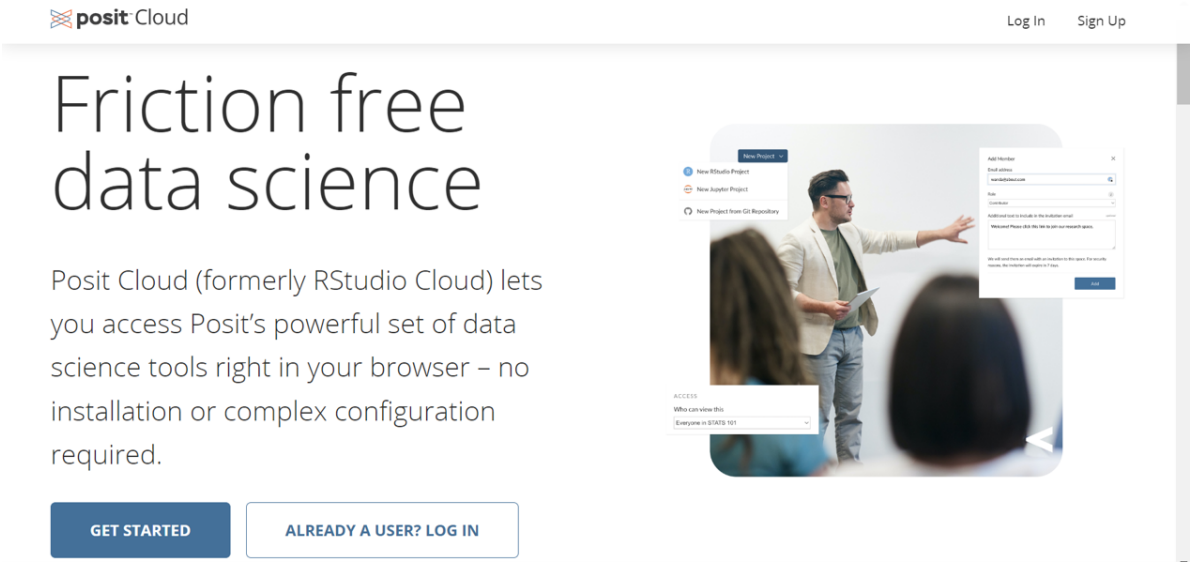


Figure 2.7: The Posit/RStudio Cloud Homepage

2.3.4.2 Step 2

Sign up for a new user account (Figure **??**).

2.3.4.3 Step 3

Log in to the new account, and you should see the page shown below (Figure **??**). It includes the workspace where all projects will be hosted. On the top right-hand side of the website is a button that allows one to create a "New Project." Additionally, the "Learn" and "Help" sections on the left panel can assist with troubleshooting.

2.3.4.4 Step 4

Create an R script to allow for code editing (Figure **??**).

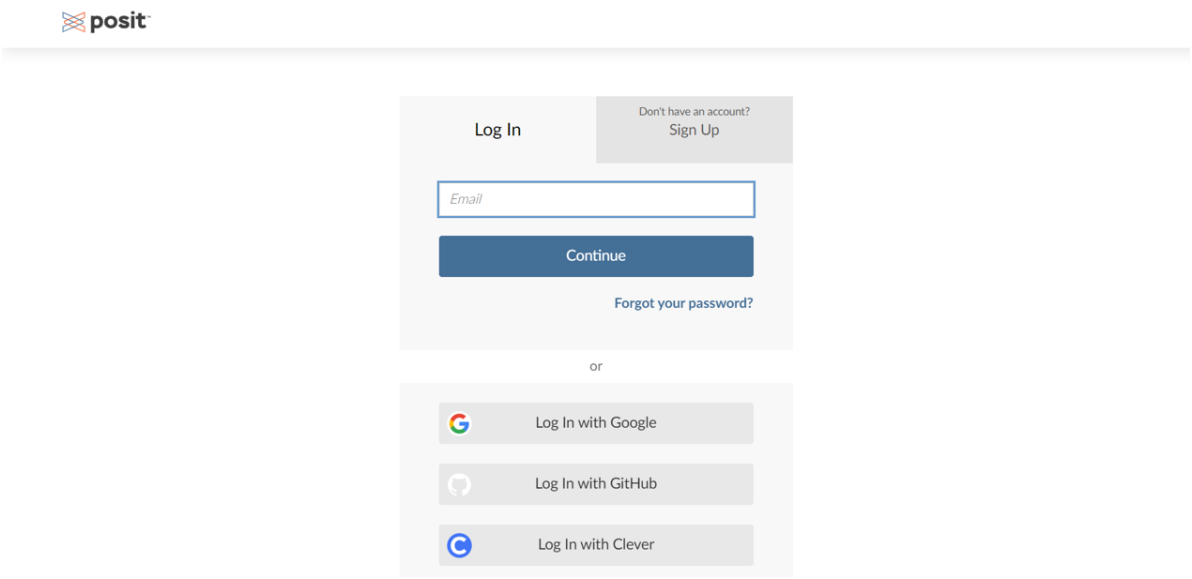


Figure 2.8: The Posit/RStudio Cloud Login Page

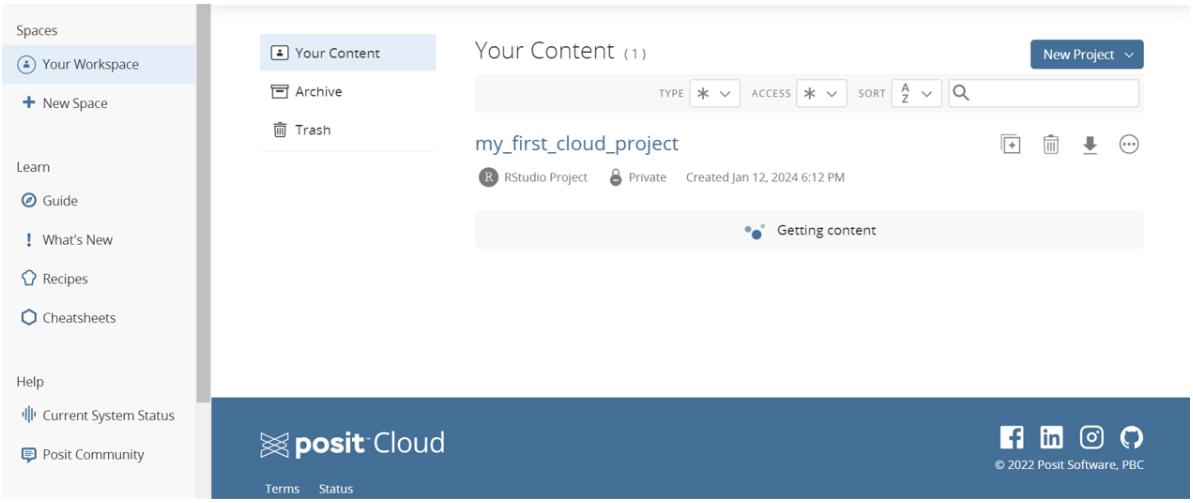


Figure 2.9: The Posit/RStudio Cloud Workspace

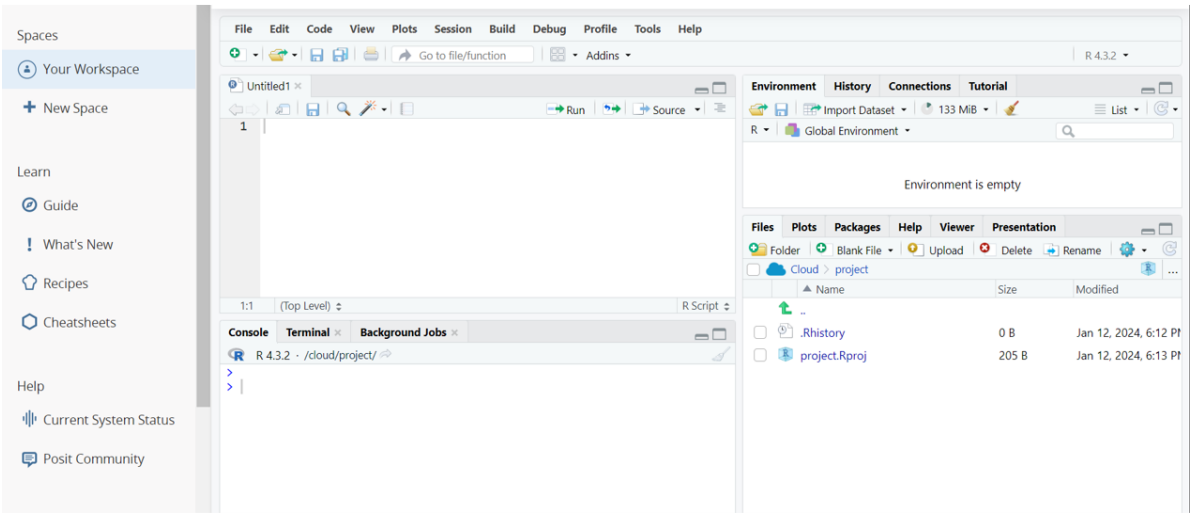


Figure 2.10: The Posit/RStudio Cloud Console

2.3.5 Using alternative code editors to work with R (Advanced Users)

Visual Studio (VS) Code is a code editor that can be used with various programming languages. For users that have previous experience with this code editor and would like to use it when reading this book, it is possible to use VS Code as an alternative to RStudio. After installing R, the "R Extension for Visual Studio Code" can be installed in the "Extensions" menu (Figure **??**).

2.4 Exercises

i. Download and install R and RStudio (*If you have not already done so*).
ii. Describe the steps involved in installing R on a Windows operating system.
iii. Are there any specific considerations when installing R on a macOS or Linux system?
iv. Set up a free RStudio Cloud Account on the Posit Website.
v. Compare the local RStudio with the cloud-based RStudio and list the potential benefits/disadvantages of both.
vi. Verify your R installation by running a simple R script that prints "Hello, R! My name is (fill in the blank)" to the console.
vii. Can you customize the appearance or behavior of RStudio according to your preferences?

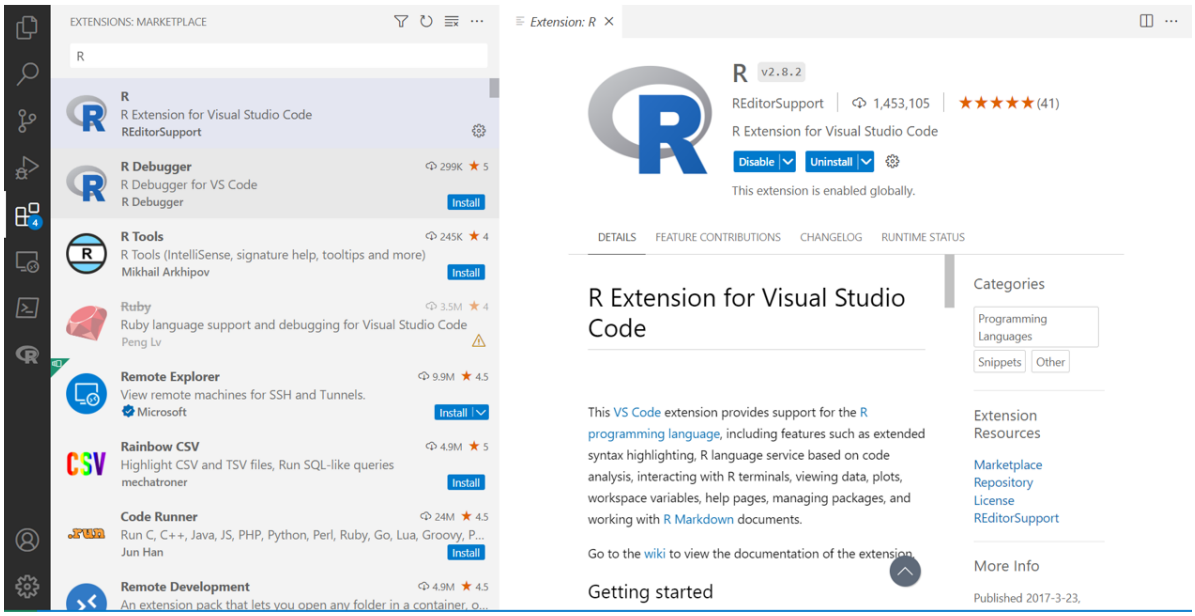


Figure 2.11: The R interface on Visual Studio Code

2.5 Summary

In this chapter, we have provided a step-by-step guide for downloading R and RStudio on the learner's personal computer. Additionally, the learner has been shown how to set up an RStudio Cloud account on the Posit website, which allows for the use of R/RStudio using a web-based browser. Lastly, the learner has been shown how to access R/RStudio using alternative code editors such as Visual Studio Code. Overall, having a functional R and RStudio environment is key to gaining the most out of your learning journey. In the next chapter, we will navigate the R/RStudio interfaces and explain the various panes and menu options.

3 Navigating the R and RStudio interfaces

3.1 Questions

- What are the main components of the R interface?
- How is the RStudio Integrated Development Environment (IDE) organized, and what are its key features?
- How does one navigate the R and RStudio interfaces?
- What are the functions of the various RStudio panes?

3.2 Learning Objectives

- Identify and understand the main components of the R interface.
- Navigate the RStudio IDE and comprehend its organizational structure.
- Customize your RStudio environment to suit your preferences.
- Understand the roles of the console and menu options in the R software.
- Review the various panes and menu options in the RStudio software.
- Learn the various keyboard shortcuts that can improve speed and efficiency.

3.3 Lesson Content

3.3.1 The R interface

3.3.1.1 Menu options

The R graphical user interface opens up with a console where you can start writing and executing code (Figure **??**). Additionally, there exists a toolbar that contains shortcuts to commonly used functions. The menu bar allows one to perform various operations within the R software. The various menu bar options and a brief summary of their main uses are listed below:

- File: Create new scripts and workspaces, as well as review the session history.
- Edit: Copy, paste, and edit, as well as set up the graphical user interface (GUI) preferences.
- View: Include the toolbar and status bar within the GUI.
- Misc: Control computations and review objects.
- Packages: Install, load, and update packages.
- Windows: Arrange multiple windows in the learner's preferred orientation.
- Help: Find links to FAQs, manuals, and help functions.

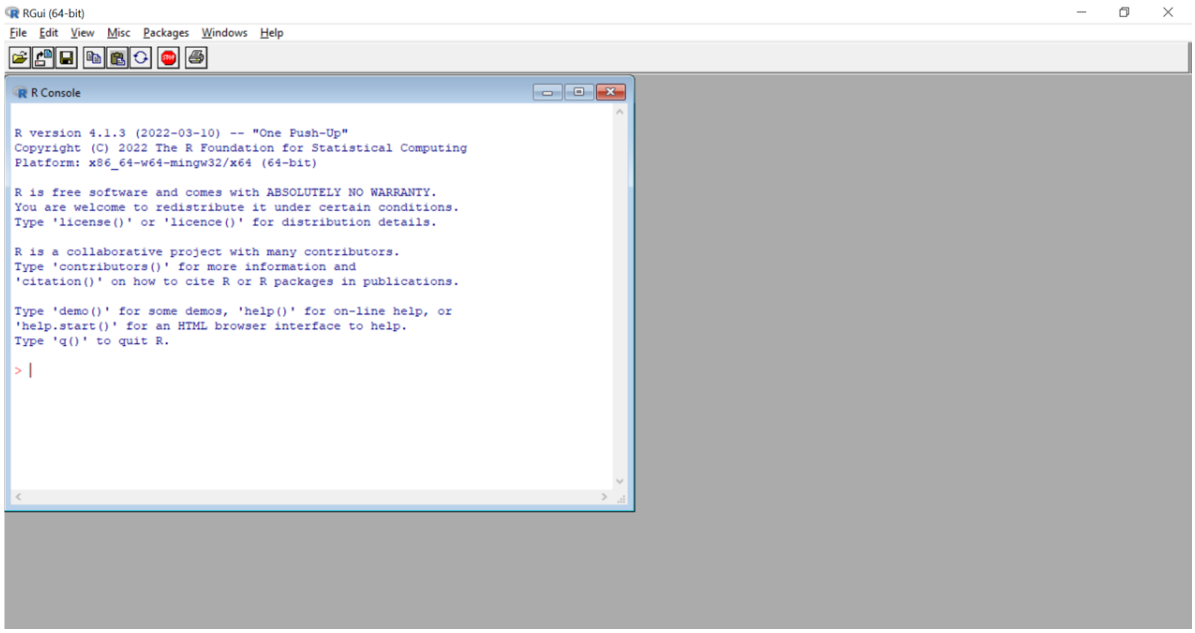


Figure 3.1: The R Console

3.3.2 RStudio interface (Panes)

When opened for the 1st time, the RStudio interface will have 3 panes: Console, Environment, and Navigation. To include the 4th required panel, click on "File" –> "New File" –> "R Script". Now that the 4 panes are open, let us review their functions.

3.3.2.1 Script Pane (Text Editor)

The script pane allows the user to write, edit, and save code. This code can then be run, and the output will be displayed in the console pane.

3.3.2.2 Console Pane

In the console pane, one can write and edit code, but not save. Within the console, one can run one line of code at a time. Additionally, one can access the terminal tab and run code via the command line. Lastly, we can access the "Background Jobs" tab to monitor the progress of R scripts that are running in a separate, dedicated R session.

3.3.2.3 Environment/History Pane

This pane has multiple uses. The "Environment" tab shows the values of objects loaded in the current R session. Additionally, from this tab, one can load workspaces, import datasets, and observe memory usage. The "History" tab allows you to see what has been run previously, and the "Connections" tab provides a method for connecting to various databases. Lastly, the "Tutorial" tab helps with training users on the basic concepts of R.

3.3.2.4 Navigation Pane

The navigation pane allows the user to manage files (create new folders and scripts), view generated plots, and review and update installed packages. Additionally, there is a "Help" tab for basic queries, and "Viewer" and "Presentation" tabs for reviewing analysis outputs.

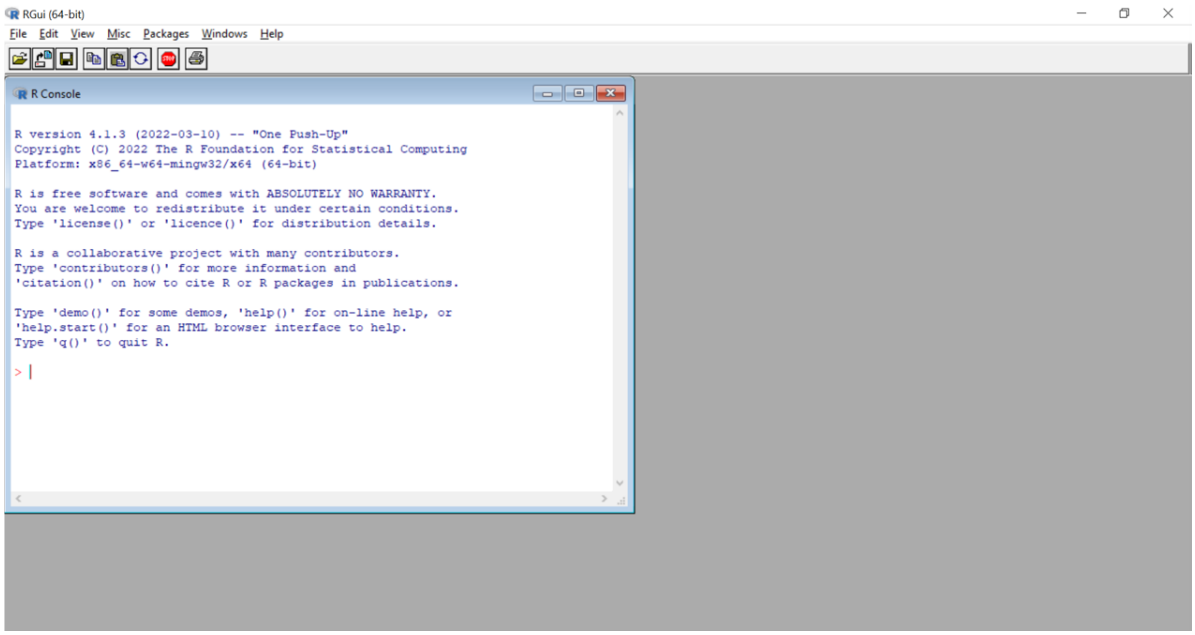


Figure 3.2: The R Console

3.3.3 RStudio interface (Menu Options)

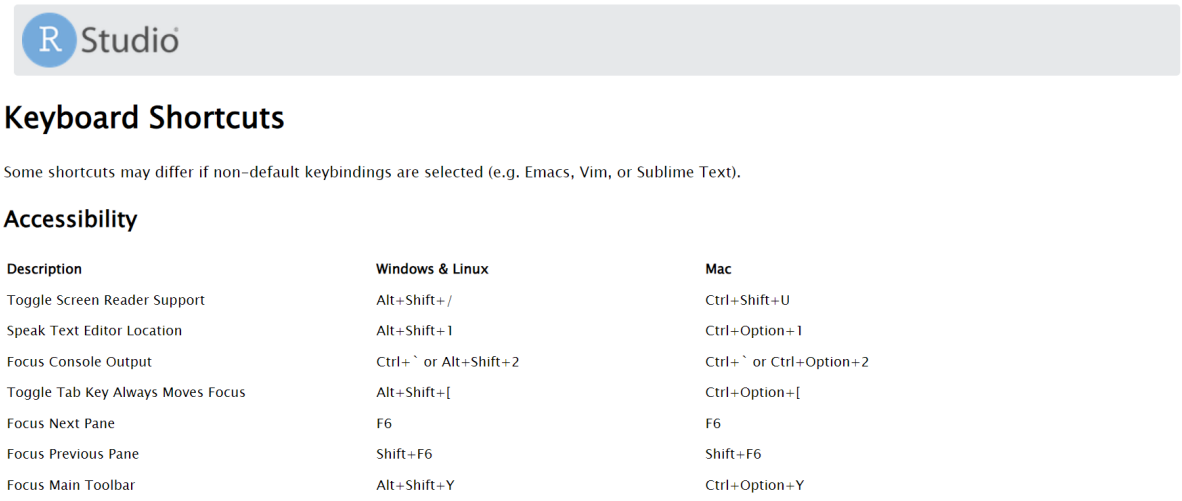
The RStudio interface has a number of menu bar options (Figure **??**). To orient the user, a brief description of each menu bar option has been provided below.

- File: Create new files and projects, save them, as well as open and close the projects. Additionally, importing of datasets is made much easier by the "Import Dataset" option.
- Edit: Using this option, one can cut, copy, and paste, as well as undo and redo any actions.
- Code: Edit code formats and run code.
- View: Change the appearance of the RStudio graphical user interface (GUI), adjust and view different panes.
- Plots: View plots and save the plot outputs to different formats.
- Session: Create sessions, load workspaces, and set the working directories.
- Build: Configure build tools and edit the project options.
- Debug: Debug your code.
- *Profile:*
- Help: The help section allows one to access tools for troubleshooting. Additionally, it contains documentation and cheatsheets that allow for to discover more about the software.
- Tools: Install packages and check for package updates. Additionally, one can review version control options, access the terminal, and look at keyboard shortcuts. A short snippet of the numerous shortcuts is shown below (Figure **??**).

Lastly, one access both "Project Options" and "Global Options" from this menu bar option. Here, the user can change the appearance and pane layout for individual projects and globally.

3.4 Exercises

i. Describe the main components of the RStudio interface.
ii. Open RStudio and explore the different panels of the interface, then provide a brief summary of the purpose of the Environment, History, Files, and Plots panes.
iii. What is the purpose of the Console pane in RStudio?
iv. Access the help documentation using the Help pane and find information on specific R functions/packages.



RStudio

Keyboard Shortcuts

Some shortcuts may differ if non-default keybindings are selected (e.g. Emacs, Vim, or Sublime Text).

Accessibility

Description	Windows & Linux	Mac
Toggle Screen Reader Support	Alt+Shift+/	Ctrl+Shift+U
Speak Text Editor Location	Alt+Shift+1	Ctrl+Option+1
Focus Console Output	Ctrl+` or Alt+Shift+2	Ctrl+` or Ctrl+Option+2
Toggle Tab Key Always Moves Focus	Alt+Shift+[	Ctrl+Option+[
Focus Next Pane	F6	F6
Focus Previous Pane	Shift+F6	Shift+F6
Focus Main Toolbar	Alt+Shift+Y	Ctrl+Option+Y

Figure 3.3: A list of shortcuts available in RStudio

v. Customize your RStudio environment by changing the theme and adjusting the appearance settings.
vi. Practice using keyboard shortcuts for common tasks in RStudio.
vii. How can you customize the layout of panes in RStudio?

3.5 Summary

This brief overview of the R/RStudio interface should make the learner more comfortable when using the software. In this chapter, we explored the main components of the R interface and delved into the organizational structure of the RStudio IDE. Understanding the different panes and tabs within RStudio is crucial for efficient and effective programming. We navigated through the Source, Console, Environment, Plots, and Help panes, as well as additional tabs for managing files, plots, packages, and the viewer. Customizing your RStudio environment enhances your overall experience, and practicing keyboard shortcuts can significantly improve your coding efficiency. As you proceed with this guide, having a solid grasp of navigating the R and RStudio interfaces will empower you in your journey as an R programmer. Few introductory courses provide this overview, and it is hoped that this will decrease the cognitive load on learners in future lessons.

4 Managing your files and data

4.1 Questions

- Why is good file and data management important in R?
- What is the concept of a project in R, and how does it help in organizing your work?
- What is the purpose of creating R project folders and scripts?
- Describe the significance of a working directory in R, and how it impacts your workflow?
- How do you create, save, and run R scripts for efficient code management?

4.2 Learning Objectives

- Understand the importance of properly organized files and datasets for efficient R projects.
- Understand the concept of R projects and their role in organizing your work.
- Write and run R code in scripts for better code organization and reproducibility.
- Master the concept of working directories and navigate your project files effectively.

4.3 Lesson Content

4.3.1 Introduction

Good file and data management allows the user to debug code, reduce the number of coding errors, and improve data quality. Additionally, by utilizing best practices, the learner can easily share their work, organize projects, and develop a reproducible workflow.

4.3.2 Projects

An R project enables the learner to keep all their work in one folder. In the R project folder, all scripts, data files, figures, tables, history, and output can be stored in sub-folders within the R project. The root folder of the project is also known as the working directory (described in the section below).

To create a new project folder:

File –> New Project

4.3.3 Scripts

To ensure that the learner has a record of all the code that they have written in the console, it is important that the learner also write the code in a script and saves it. Failure to do this will mean that the learner will have to write the code in the console every time they need to execute it. Code written in script files allows one to run it multiple times and edit when necessary.

To create a new script file:

File –> New File –> R Script

Save the file in your project folder

Ensure that the file names are readable for both the R software and collaborators.

Examples include:

- "my_model_1.R"

4.3.4 Working Directory

The working directory is a file path or location on your computer that R uses for reading and writing files. The function `getwd()` prints the current working directory, and `setwd()` allows the user to set the working directory. This is where R looks for files that you ask it to load, and where it puts any files that you ask it to save.

The paths to files or directories can be defined using relative or absolute paths. The relative path is where the file is found in the location where the user is, and absolute includes the entire path from the root directory.

4.3.5 Finding help when using packages and functions

- There is a built-in help system that can be accessed via the "Help" menu, or one can use the `?` symbol before the function, package, or dataset to get a brief description and instructions for use.
- Online R documentation is also available to allow the user to learn the basics of R.

4.4 Exercises

i. Create a new RStudio project called "novice_programmer_guide".

ii. Within the project folder, create a new R Script called "introduction.R" and save it.

iii. Set your working directory in the console to the root of this new project, and practice loading data from a file within it.

iv. Explain the significance of using relative and absolute file paths in R, and experiment with using relative paths to access different data files within your project directory.

v. Explain the role of the functions, `file.path()`, `list.files()`, and `dir.create()`, in R.

vi. Describe the purpose of the `.Rproj` file in an R project.

vii. How can you clear the R workspace? Explore functions like `rm()` and `rm(list=ls())` to remove unwanted objects.

4.5 Summary

Hopefully, this lesson has demonstrated to the learner the importance of good file and data management. We have reviewed the concepts of projects, scripts, and working directories and shown how this can help with file organization, saving/reusing code, and ultimately impact project reproducibility. In the next lesson, we will see how this is important when importing data and saving analysis outputs.

5 Importing data and saving analysis outputs

5.1 Questions

- Can users access and use internal datasets for analysis in R? How do I load the datasets?
- How does one import external datasets into R? What functions and packages are commonly used for this purpose?
- How do I handle different data formats such as CSV, Excel, or text files?
- Once the user completes their data analysis, how can their output(s) be saved?
- How can I create new data objects or modify existing ones within R?

5.2 Learning Objectives

- Learn how to load and access internal R datasets.
- Import external datasets into your R project folder.
- Demonstrate how to save analysis outputs from R such as tables and datasets.
- Understand methods for importing data stored internally in R and from external sources.
- Utilize built-in functions and packages to read various data formats like CSV, Excel, and text files.

5.3 Lesson Content

5.3.1 Introduction

A dataset is a collection of data, and R offers the user the opportunity to either use internal datasets or import external datasets for analysis. In this lesson, we will cover the basics of accessing internal datasets, the importation of external datasets, as well as the saving of analysis outputs, such as plots and tables.

5.3.2 Working with internal datasets

R has several built-in datasets. To access them, we use the `data()` function after loading in the datasets library.

```
#| eval: false
#| include: false
library(datasets)

print(data())
```

To find out more about a specific dataset, we use the `?` operator.

```
?dataset
```

Example:

Helpful functions when working with datasets

5.3.2.1 Loading datasets

```
datasets::dataset
```

OR

```
datasets("dataset")
```

Example:

```
datasets::airquality

data("airquality")
```

5.3.2.2 Print datasets

```
print(dataset)
```

Example:

```
print(airquality)
```

5.3.2.3 Inspect head and tail of datasets

`head(dataset)`

OR

`tail(dataset)`

Example:

```
head(airquality)
```

```
tail(airquality)
```

5.3.2.4 Inspect the column and row names in the dataset

Names

`names(dataset)`

Row names

`rownames(dataset)`

Column names

`colnames(dataset)`

Example:

```
names(airquality)
```

```
rownames(airquality)
```

```
colnames(airquality)
```

5.3.2.5 Dataset dimensions

Dimensions

`dim()`

Number of rows

`nrow()`

Number of columns

`ncol()`

Example:

```
dim(airquality)
```

```
nrow(airquality)
```

```
ncol(airquality)
```

5.3.2.6 Summary statistics for the dataset

Summary of the dataset

`summary()`

Example:

```
summary(airquality)
```

Access specific columns in a dataset

`dataset$column`

Example:

```
airquality$Ozone
```

5.3.3 Importing external datasets

As an R user, one of the tasks that you will often engage in is the importation of data from various sources. The commonly used data types include, but are not limited to CSV, TXT, JSON, SQL, SAS, STATA, SPSS, and XLSX. Additionally, R has two native data formats: Rdata (Rda) and Rds. Rdata is for multiple objects, while Rds is for a single R object.

When importing external data, first set the correct working directory. The function `getwd()` will show the current working directory, and `setwd(file_path_name)` will set the working directory to the file path name specified.

NOTE: External datasets can be accessed from multiple sources. Examples include can be found in the Appendix 1 (Appendix **??**). Both of these websites contain large repositories of datasets that can be used for the examples provided below.

Download the datasets from Appendix 2 (Appendix **??**) and save them to your working directory.

Data importation can use base R functions or the `readr` package, which is part of the tidyverse. We will review both sets of methods in this section.

5.3.3.1 Base R

1. Commonly used data importation formats

There is a wide variety of base R functions used for importing external data into R. The main format for importing data is shown below:

```
read.___(path to file, header = __, sep = "_", dec = "_")
```

- The argument "path to file" provides the location of the file the user intends to import.
- The argument header is set as either TRUE or FALSE, if TRUE, R assumes that the file has a header.
- The sep argument is a group of field separation characters which include , `,,` `;`, or `\t`.
- The dec argument allows one to set the different characters for decimal points, which include `.` or `,`.
- The stringsAsFactors argument, by default, allows strings to be read as factors. This can be set as TRUE or FALSE

The main functions for importing data using base R are listed below:

- `read.table()` is a general function to read in a file in table format as a data frame.

Example:

```
read.table("datasets/r4novice_datafile.csv", header = TRUE, sep = ",")
```

- `read.csv()` imports data with comma separated values. Alternatively, we use `read.csv2()` is used where a comma (“,”) indicates a decimal point and a semicolon (“;”) is a field separator.

Example:

```
read.csv("datasets/r4novice_datafile.csv", header = TRUE, sep = ",")
```

- `read.delim()` is used for files with tab-separated values, such as TXT, and where point “.” is used as a decimal point. Alternatively, `read.delim2()` is used for files with tab-separated values such as TXT and where comma “,” is used as a decimal point.

Example:

```
read.delim("datasets/r4novice_datafile.txt", header = TRUE)
```

- General data import

For beginners, one can use `read.___(file.choose())` for the interactive selection of files.

Example:

```
read.table(file.choose())
```

NOTE:Run this command in your R console and select the appropriate file from the working directory.

- Excel (XLS and XLSX) files are read in with `read.xlsx()`

Load the required library

```
library(xlsx)
```

Example:

```
read.xlsx("datasets/r4novice_datafile.xls", sheetIndex = 1)
```

```
read.xlsx("datasets/r4novice_datafile.xls", sheetIndex = 1)
```

2. Less frequently used data importation formats

a. Fixed-width formats: `read.fwf("file path", header = TRUE)`

b. SPSS/Stata/SAS Data Files: The foreign package reads SPSS SAV files, Stata DTA files, and SAS XPORT libraries.

Steps for use:

Install the library

```
install.packages("foreign")
```

Load the library

```
library(foreign)
```

SPSS: `read.spss("file path", to.data.frame = __, use.value.labels = __)`

- to.data.frame: indicates whether R should treat the loaded data as a dataframe (options are TRUE/FALSE).
- use.value.labels: indicates whether R should convert variables with value labels into R factors (options are TRUE/FALSE).

Stata: `read.dta("file", convert.dates = __, convert.factors = __)`

convert.dates: converts Stata dates to R's Date class

convert.factors: creates factors with Stata value labels

SAS: `read.xport("file path")`

Alternatively, one can utilize specific packages like `rio` or `haven` for more complex data formats like SPSS or SAS.

Install the library

```
install.packages("haven")
```

Load the library

```
library(haven)
```

SPSS: `read_sav("file path")`

SAS: `read_sas("file path")`

c. Native data formats in R

R Data Files: `load("file_name.rda")`

RDS Files: `readRDS("file_name.rds")`

Online Files

To download and import an online file, we use `read.html()` and `read.xml()`

Example:

```
read.html("file path")
```

```
read.xml("file path")
```

e. Miscellaneous file formats

JSON

Install the library

```
install.packages("rjson")
```

Load the library

```
library(rjson)
```

```
fromJSON(file = "file name")
```

SQL

Install the library

```
install.packages("RSQLite")
```

Load the library

```
library(RSQLite)
```

```
sql_connect <- dbConnect(RSQLite::SQLite(), "file name")
```

```
dbListTables(sql_connect)
```

Matlab

Install the library

```
install.packages("Rmatlab")
```

Load the library

```
library(Rmatlab)
```

```
readMat("file name")
```

5.3.3.2 Tidyverse and the `readr/readxl` packages

To enhance the speed and efficiency of data imports, the user can work with the **readr** and **readxl** packages that are part of the Tidyverse. This is because the functions in this package allows for faster data imports and work similarly despite the data type that is imported.

1. Commonly used data importation formats in the tidyverse

```
# install.packages("readr")

library(readr)
```

a. Import flat files

- `read_table()` is used to import whitespace-separated files.

Example:

```
read_table("datasets/r4novice_datafile.txt", col_names = TRUE)
```

- `read_csv()` is for comma-separated values (CSV), while `read_csv2()` is used for semicolon-separated values with `,` as the decimal mark.

Example:

```
read_csv("datasets/r4novice_datafile.csv")
```

- `read_tsv()` is for tab-separated values (TSV)

Example:

```
read_tsv("datasets/r4novice_datafile.txt")
```

- `read_delim()` is for all delimited files, such as CSV and TSV.

Example:

```
read_delim("datasets/r4novice_datafile.csv", delim = ",")
```

b. Import spreadsheets

```
# install.packages("readxl")

library(readxl)
```

Use excel_sheets() to read the names of the different worksheets in the Excel workbook.

```
excel_sheets("datasets/r4novice_datafile.xlsx")
```

```
read_excel("datasets/r4novice_datafile.xlsx", sheet = "Sheet1")
```

2. Less frequently used data importation formats

- Import Google Sheets

```
# install.packages("googlesheets4")

library(googlesheets4)
```

Use `read_sheet("link to file")` to read in the file via the link.

- `read_fwf()` is used to read in fixed-width files.
- `read_log()` is the standard format for reading in web log files.
- `readRDS()` is used to read data stored as a single R object.
- `read_lines()` is used to read data up to a specified number of lines in a file.

NOTE: Missing data is a major challenge in data analysis. Strategies for dealing with this are discussed in Chapter 12 "Handling missing data" (Chapter **??**).

5.3.3.3 Inspecting the imported data

Once the data is imported, the user can inspect the data with `str()`, `head()`, and `summary()` functions. Additionally, one can check various aspects of the data such as `names()`, `dim`, and `class` to validate that the import was successful.

Example:

Import a specific dataset

```
import_inspect <- read_csv("datasets/r4novice_datafile.csv")

head(import_inspect)

tail(import_inspect)

summary(import_inspect)

str(import_inspect)

names(import_inspect)

dim(import_inspect)

class(import_inspect)
typeof(import_inspect)
```

5.3.4 Saving data and analysis outputs

Both the base R functions and the `readr` package can be used to save data. This data can be in the form of cleaned/rearranged tables, or it can be other analysis outputs such as plots.

5.3.4.1 Base R

Before we explore the functions used to save data, I will create a dataframe. A dataframe is one of the most common data objects used to store tabular data in R. A more in-depth look at dataframes will be provided in the chapter "Data Structures (Part I)" (Chapter **??**).

```
df_to_save <- data.frame(one=c(1, 3, 2, 9, 5),
                         two=c(7, 7, 3, 8, 2),
                         three=c(3, 3, 9, 7, 1),
                         four=c(5, 2, 2, 8, 9))
```

1. Commonly used R functions

- `write.csv()` saves data to a CSV file.

Example:

```
write.csv(df_to_save, file = "saved_datasets/r4novice_saved_data.csv")
```

- `write.table()` saves data to a specified file type.

Example:

```
write.table(df_to_save, file = "saved_datasets/r4novice_saved_data.txt", sep = ",")
```

- `fwrite()` saves data to a specified file type. It is obtained from the `data.table` package.

```
library(data.table)
```

Example:

```
fwrite(df_to_save, file = "saved_datasets/r4novice_saved_data_2.csv", sep = ",")
```

5.3.4.2 Tidyverse and the `readr/readxl` packages

The **readr** and **readxl** also offer complementary functions that allow the user to save data.

a. Flat files

- `write_csv()` and `write_csv2()` can be used to write comma-delimited and semicolon-delimited files, respectively.

Example:

```
write_csv(df_to_save, file = "saved_datasets/r4novice_saved_data_3.csv")
```

- `write_tsv()` is used to write a tab delimited file.

Example:

```
write_tsv(df_to_save, file = "saved_datasets/r4novice_saved_data_2.txt")
```

- `write_delim()` is used to write files with any delimiter.

Example:

```
write_delim(df_to_save, file = "saved_datasets/r4novice_saved_data_4.csv", delim = ";")
```

b. Excel files

```
# install.packages("writexl")

library(writexl)
```

Create an XLS and XLSX file

```
#write_xlsx(df_to_save, "saved_datasets/r4novice_saved_data_4.xls")
write_xlsx(df_to_save, "saved_datasets/r4novice_saved_data_4.xlsx")
```

5.4 Exercises

i. Access the `mtcars`, `iris`, and `airquality` datasets and read the documentation using `?`. Additionally, explore the different characteristics of the data, such as names, dimensions, and summary statistics.

ii. Import datasets from different formats like CSV, Excel, and text files into your R environment using the base R functions such as read.*().

iii. Save your file using a new name with write.*(), save(), or .RData?

iv. How can you check the structure and summary statistics of an imported dataset in R?

v. What options do you have when importing data from non-local sources like URLs or databases? Explore functions like url() and dbConnect() for remote data access.

vi. How can you specify import options for different file formats? Learn about arguments like sep, header, and na.strings for customized data reading.

vii. What is the difference between overwriting a dataset and appending new data to it in R?

5.5 Summary

In this lesson, the learner has been guided through the process of accessing internal datasets. Additionally, methods for importing external datasets have been demonstrated for various data formats. Lastly, the learner has been shown how to save the outputs of the data analysis. With this knowledge, the learner sis now ready to perform basic arithmetic operations, which will be covered in the next chapter.

6 Basic arithmetic, arithmetic operators, and variables

6.1 Questions

- How can we perform basic arithmetic operations in R?
- What arithmetic operators can be used in R?
- How do I assign values to variables?
- Are there naming conventions/variable style guides for creating new variables?

6.2 Learning Objectives

- Learn how to use R to perform basic arithmetic.
- Declare and manipulate variables to store and retrieve data in R.
- Master basic arithmetic operations like addition, subtraction, multiplication, and division in R.
- Understand the concept of variables and assign numeric values in your R code.
- Use appropriate naming conventions for variables.
- Differentiate between different variable types (numeric, integer, character) and choose appropriate ones.

6.3 Lesson Content

6.3.1 Basic Arithmetic

To get started, first, we will open R or RStudio (Figure **??**). In R, go to the console, and in RStudio, head to the console pane. Next, type in a basic arithmetic calculation such as `1 + 1` after the angle bracket (`>`) and hit "Enter."

NOTE: According to the Tidyverse Style Guide, there should be a space on either side of the operator.

Examples

- `a <- 2`
- `b <- 4 + c`

```
1 + 1
```

The output will be observed next to the square bracket containing the number 1 (`[1]`). The 1 indicates that the indexing begins at position 1. Here, as opposed to Python, which uses zero-based indexing, R uses one-based indexing.

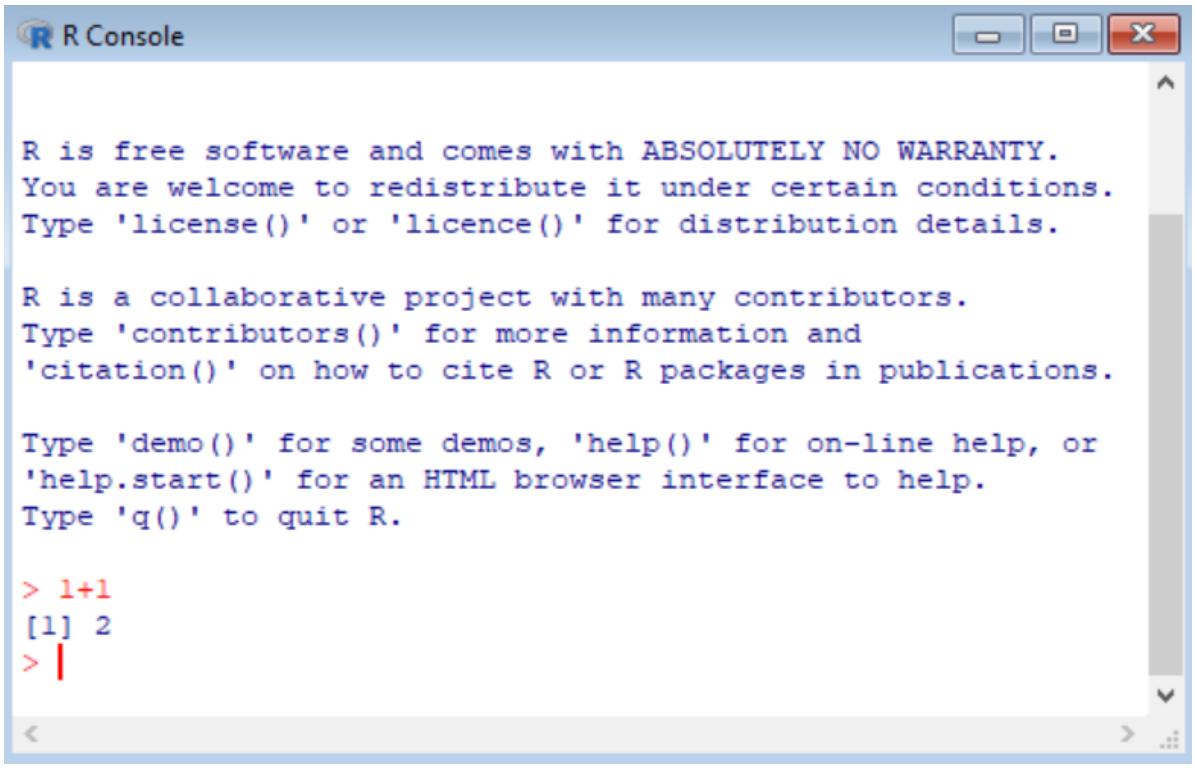


Figure 6.1: A basic arithmetic calculation in the R Console

Additionally, to include comments into the code block, we use the hash (`#`) symbol. Anything written after the code block will be commented out and not run.

```
# A simple arithmetic calculation (which is not run because of the hash symbol)

1 + 1
```

6.3.2 Arithmetic operators

Various arithmetic operators (listed below) can be used in R/RStudio.

Arithmetic Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
** or ^	Exponentiation
%%	Modulus (remainder after division)
%/%	Integer division

Example

Create an R script called "my_arithmetic_1.R"

Run the following calculations in the script and save the file to the working directory.

6.3.2.1 Addition

```
10 + 30
```

6.3.2.2 Subtraction

```
30 - 24
```

6.3.2.3 Multiplication

```
20 * 4
```

6.3.2.4 Division

```
93 / 4
```

6.3.2.5 Exponentiation

```
3^6
```

6.3.2.6 Modulus (remainder with division)

```
94 %% 5
```

6.3.2.7 Integer Division

```
54 %/% 7
```

6.3.2.8 Slightly more complex arithmetic operations

```
5 - 1 + (4 * 3) / 16 * 3
```

6.3.3 Variables

Variables are instrumental in programming because they are used as "containers" to store data values. To assign a value to a variable, we can use `<-` or `=`. However, most R users prefer to use `<-`.

- Variables store values for later use in your code. This results in improved readability and efficiency.
- Assign values to variables using the `<-` operator (e.g., `x <- 5`).
- Variable names should be descriptive and avoid special characters.

6.3.3.1 Variable assignment

1. Using `<-`

```
variable_1 <- 5
variable_1
```

2. Using `=`

```
variable_2 <- 10
variable_2
```

3. Reverse the value and variable with `->`

```
15 -> variable_3
variable_3
```

4. Assign two variables to one value

```
variable_4 <- variable_5 <- 30
variable_4
variable_5
```

The output of the variable can then be obtained by:

1. Typing the variable name and then pressing "Enter,"
2. Typing "print" with the variable name in brackets, `print(variable)`, and,
3. Typing "View" with the variable name in brackets, `View(variable)`.

Both `print()` and `View()` are some of the many built-in functions available in R.

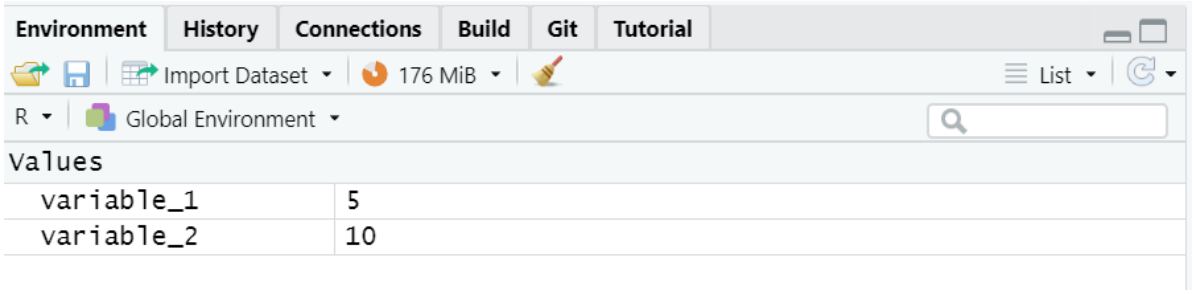


Figure 6.2: The environment pane in RStudio with stored variables

```
print(variable_1)

View(variable_2)
```

Output of `View()` will be seen in the script pane.

6.3.3.2 The `assign()` and `rm()` functions

In addition to using the assignment operators (`<-` and `=`), we can use the `assign()` function to assign a value to a variable.

```
assign("variable_6", 555)
variable_6
```

To remove the assignment of the value to the variable, either delete the variable in the "environment pane" or use the `rm()` function.

```
variable_7 <- 159

rm(variable_7)
```

After running `rm()` look at the environment pane to confirm whether variable_7 has been removed.

6.3.3.3 Naming variables

At this point, you may be wondering what conventions are used for naming variables. First, variables need to have meaningful names such as current_temp, time_24_hr, or weight_lbs. However, we need to be mindful of the variable style guide (link) which provides us with the appropriate rules for naming variables.

```
object_name <- value
```

Objects must start with a letter and only contain letters and numbers

Some rules to keep in mind are: 1. R is case-sensitive (variable is not the same as Variable), 2. Names similar to typical outputs or functions (TRUE, FALSE, if, or else) cannot be used, 3. Appropriate variable names can contain letters, numbers, dots, and underscores. However, you cannot start with an underscore, number, or dot followed by a number.

6.3.3.4 Valid and invalid variable names

Types of variable names:

- snake_case
- CamelCase
- use.periods
- snake.case.CamelCase_snake_case

Valid names:

- time_24_hr
- .time24_hr

Invalid names:

- _24_hr.time
- 24_hr_time
- .24_hr_time

6.4 Exercises

i. In R, calculate `3 + 5` and then `4 * 6`.
ii. Assign the value 10 to a variable called x. Then calculate `x ^ 2`. Next, calculate the expression `(3 + x) * (x - 1)` using the x from before.
iii. Declare a variable to store your age and assign a value to it. Print the variable value.
iv. Create variables for the length and width of a rectangle and calculate its area.
v. Explain the rules for naming variables in R. Provide examples of valid and invalid variable names.
vi. What is operator precedence in R? How does it work for arithmetic operators?
vii. What data types can be used for arithmetic operations in R?
viii. What is the difference between `<-` and `=` for assignment in R?

6.5 Summary

In this chapter, I have demonstrated how to perform basic arithmetic operations and how to use the various arithmetic operators available in R/RStudio. Additionally, the concept of variables and values has been explained, and conventions for appropriately naming variables have been discussed. In the next chapter, we will look at the other types of primary operators in R/RStudio.

7 The primary types of operators in R

7.1 Questions

- What are the primary operator types in R?
- How can one use the various operator types in data analysis?
- How do arithmetic, comparison, and logical operators work in R?
- When should I use assignment operators versus comparison operators?

7.2 Learning Objectives

- Identify the primary operator types in R.
- Learn how to use various operators for data manipulation.
- Utilize arithmetic operators for efficient numerical calculations in your R scripts.
- Apply comparison operators to evaluate conditions and filter data based on logical comparisons.
- Employ logical operators to combine multiple conditions and control the flow of your R code.

7.3 Lesson Content

7.3.1 Introduction

R has many types of operators that can perform different tasks. Here we will focus on 5 major types of operators. The major types of operators are:

- Arithmetic,
- Relational,
- Logical,
- Assignment, and

- Miscellaneous.

7.3.2 Arithmetic Operators

Arithmetic operators are used to perform mathematical operations. These operators were reviewed in the previous lesson on "Basic arithmetic, arithmetic operators, and variables" (Chapter **??**)

7.3.3 Relational Operators

Relational operators are used to find the relationship between 2 variables and compare objects. The output of these comparisons is Boolean (TRUE or FALSE). The table below describes the most common relational operators.

Relational Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not Equal to

7.3.3.1 Assign values to variables

```
x <- 227
y <- 639
```

a. Less than

```
x < y
```

b. Greater than

```
x > y
```

c. Less than or equal to

```
x <= 300
```

d. Greater than or equal to

```
y >= 700
```

e. Equal to

```
y == 639
```

f. Not Equal to

```
x != 227
```

7.3.4 Logical Operators

Logical operators are used to specify multiple conditions between objects. Logical operators work with basic data types such as logical, numeric, and complex data types. These operators generally return `TRUE` or `FALSE` values. Numbers greater that 1 are `TRUE` and 0 equals `FALSE`. The table below describes the most common logical operators.

Logical Operator	Description
!	Logical NOT
|	Element-wise logical OR
&	Element-wise logical AND

7.3.4.1 Assign vectors to variables

NOTE: Full discussion in the chapter on "Vectors" (Chapter **??**)

```
vector_1 <- c(0,2)
vector_2 <- c(1,0)
```

a. Logical NOT

```
!vector_1
```

```
!vector_2
```

b. Element-wise Logical OR

```
vector_1 | vector_2
```

c. Element-wise Logical AND

```
vector_1 & vector_2
```

7.3.5 Assignment Operators

These operators assign values to variables. A more comprehensive review can be obtained in the lesson on "Basic arithmetic, arithmetic operators, and variables" (Chapter **??**)

7.3.6 Miscellaneous Operators

These are helpful operators for working in that can perform various functions. A few common miscellaneous operators are described below.

Miscellaneous Operator	Description
`%*%`	Matrix multiplication (to be discussed in subsequent chapters)
`%in%`	Does an element belong to a vector
`:`	Generate a sequence

a. Sequence

```
a <- 1:8
a

b <- 4:10
b
```

b. Element in a vector

```
a %in% b

9 %in% b

9 %in% a
```

7.4 Exercises

- List the basic operator types in R and provide an example for each.
- What operators allow you to subset vectors, lists, and data frames in R?
- Explain how the modulus operator (`%/%`) differs from the regular division operator (`/`).
- What are the membership operators `%in%` and `!%in%` used for in R?
- Assign 10 to x using `<-`. Then compare if `x == 10`.
- Calculate `3^4` using arithmetic operators.

- Compare `3 > 5`. Is the result TRUE or FALSE?
- When would you use the sequence operator (`:`) in R? Give an example.

7.5 Summary

The focus of this chapter has been the five major types of operators that can be used in R/RStudio. The operators allow one to perform numerous tasks including basic arithmetic including basic arithmetic, comparisons, subsetting, matrix multiplication, and vector generation. In the next chapter, we will discuss the available data types in R and how to utilize/handle them.

8 Data Types

8.1 Questions

- What are the primary data types in R?
- Can one convert between different data types?
- How does R handle different types of data?
- How can I determine variable type in R?
- What are some helpful functions for identifying and manipulating data types?
- How do data types impact my analysis and coding in R?

8.2 Learning Objectives

- Identify and differentiate between common data types in R.
- Learn how to convert between different data types.
- Use functions like `class()`, `typeof()`, and `str()` to inspect types.
- Convert between data types using the `as.___()` functions.

8.3 Lesson Content

8.3.1 Introduction

R and RStudio utilize multiple data types to store different kinds of data.

The most common data types in R are listed below.

Data Type	Description
Numeric	The most common data type. The values can be numbers or decimals (all real numbers).

Data Type	Description
Integer	Special case of numeric data without decimals.
Logical	Boolean data type with only 2 values (`TRUE` or `FALSE`).
Complex	Specifies imaginary values in R.
Character	Assigns a character or string to a variable. The character variables are enclosed in single quotes ('character') while the string variables are enclosed in double quotes ("string").
Factor	Special type of character variable that represents a categorical such as gender.
Raw	Specifies values as raw bytes. It uses built-in functions to convert between raw and character (`charToRaw()` or `rawToChar()`).
Dates	Specifies the date variable. Date stores a date and POSIXct stores a date and time. The output is indicated as the number of days (Date) or number of seconds (POSIXct) since 01/01/1970.

8.3.2 Examples of data types in R

1. Numeric

```
89.98
```

```
55
```

2. Integer

```
5L
```

```
5768L
```

3. Logical

```
TRUE
```

```
FALSE
```

4. Complex

```
10 + 30i
```

```
287 + 34i
```

5. Character or String

```
'abc'
```

```
"def"
```

```
"I like learning R"
```

NOTE: A comprehensive overview of handling characters with the `stringr` package will be provided in a subsequent series on the Tidyverse.

6. Dates

```
"2022-06-23 14:39:21 EAT"
```

```
"2022-06-23"
```

7. Raw

8. Factor

8.3.3 Inspecting various data types

Several functions exist to examine the features of the various data types. These include:

1. `typeof()` – what is the data type of the object (low-level)?
2. `class()` – what is the data type of the object (high-level)?
3. `length()` – how long is the object?
4. `attributes()` – any metadata available?

Let's look at how these functions work with a few examples:

```
a <- 45.84
b <- 858L
c <- TRUE
d <- 89 + 34i
e <- 'abc'
```

1. Examine the data type at a low-level with `typeof()`

```
typeof(a)
```

```
typeof(b)
```

```
typeof(c)
```

```
typeof(d)
```

```
typeof(e)
```

2. Examine the data type at a high-level with `class()`

```
class(a)

class(b)

class(c)

class(d)

class(e)
```

3. Use the `is.____()` functions to determine the data type. To test whether the variable is of a specific type, we can use the `is.____()` functions.

First, we test the variable a which is numeric.

```
is.numeric(a)

is.integer(a)

is.logical(a)

is.character(a)
```

Second, we test the variable c which is logical.

```
is.numeric(c)

is.integer(c)

is.logical(c)

is.character(c)
```

8.3.4 Converting between various data types

To convert between data types, we can use the `as.___()` functions. These include: `as.Date()`, `as.numeric()`, and `as.factor()`. Additionally, other helpful functions include `factor()` which adds levels to the data and `nchar()` which provides the length of the data.

8.3.4.1 Implicit vs. Explicit conversion

Examples

```
as.integer(a)
```

```
as.logical(0)
```

```
as.logical(1)
```

```
nchar(e)
```

Order of precedence when converting between data types is:

Character > Numeric > Integer > Logical

8.4 Exercises

- What are the basic data types in R? Give examples of each.
- How can you check the data type of a variable in R?
- Use the `class()`, `typeof()`, and `str()` functions to check the data type of objects.
- Describe the process of coercion in R.
- Provide an example of implicit and explicit coercion between data types.
- How can you coerce or convert between different data types in R?
- What does it mean for an object in R to have attributes? What attributes are commonly used?

8.5 Summary

In this chapter, I have demonstrated the primary data types in R. Additionally, the learner has been trained on how to identify and convert between the different data types. Next, we will review vectors, which are key data structures in R/RStudio.

9 Vectors

9.1 Questions

- What are vectors, and why are they essential in R?
- What are the different types of vectors and their key characteristics?
- How does one create a vector?
- What operations can be performed on a vector?
- How do I create and access different elements within a vector?
- Can I combine and manipulate vectors to achieve specific tasks?
- What are some common functions for working with vectors?

9.2 Learning Objectives

- Define and create vectors in R.
- Understand how to perform basic operations on vectors.
- Understand the concept of vectors as fundamental data structures in R.
- Master the creation and manipulation of vectors using various techniques.
- Identify and differentiate between different types of vectors based on their elements.
- Utilize functions and operators to combine, subset, and transform vectors with ease.
- Apply your understanding of vectors to enhance your R coding and data analysis efficiency.

9.3 Lesson Content

9.3.1 Introduction

A vector is a collection of elements of the same data type, and they are a basic data structure in R programming.

Vectors cannot be of mixed data type. The most common way to create a vector is with `c()`, where "c" stands for combine. In R, vectors do not have dimensions; therefore, they cannot be defined by columns or rows. Vectors can be divided into atomic vectors and lists (discussed in the "Data Structures" {Chapter **??**} section). The atomic vectors include logical, character, and numeric (integer or double).

Additionally, R is a vectorized language because mathematical operations are applied to each element of the vector without the need to loop through the vector.

Examples of vectors made up of different data types are shown below:

- Numbers

```
c(2, 10, 16, -5)
```

- Characters

```
c("R", "RStudio", "Shiny", "Quarto")
```

- Logicals

```
c("TRUE", "FALSE", "TRUE")
```

9.3.2 Sequence Generation

To generate a vector with a sequence of consecutive numbers, we can use: `sequence()`, or `seq()`.

Generate a sequence using `:`

```
a <- 9:18
a
```

```
a_rev <- 18:9
a_rev
```

```
a_rev_minus <- 5:-3
a_rev_minus
```

Generate a sequence using `sequence()`

```
b <- sequence(7)
b

c <- sequence(c(5,9))
c
```

Generate a sequence using `seq()`

The `seq()` function has four main arguments: `seq(from, to, by, length.out)`, where "from" and "to" are the starting and ending elements of the sequence. Additionally, "by" is the difference between the elements, and "length.out" is the maximum length of the vector.

```
d <- seq(2,20,by=2)
d

f <- seq(2,20, length.out=5)
f

h <- seq(20,2,by=-2)
h

j <- seq(20, 2, length.out=3)
j
```

The `seq()` function can also be used with the arguments first, by, and length.

```
seq(first, by = ___, length = ____)
```

Example

```
seq(3, by = 4, length = 30)
```

Repeating vectors

To create a repeating vector, we can use `rep()`.

```
rep(number, times)

k <- rep(c(0,3,6), times = 3)
k

rep(range, times)

l <- rep(2:6, each = 3)
l

rep(number or range, length.out)
```

```
m <- rep(7, length.out = 20)
m
n <- rep(7:10, length.out = 20)
n
```

Random number generation

The `runif()` function can be used to generate a specific number of random numbers between a minimum and maximum value.

```
runif(n, min = 0, max = 1)
```

```
runif(5, min = 3, max = 100)
```

NOTE: If you run the previous code block multiple times you will get different answers. Use `set.seed()` to get a similar sequence every time you run the calculation.

```
set.seed(12345)
runif(5, min = 3, max = 100)
```

9.3.3 More functions to manipulate vectors

Generate a random vector with 50 elements

```
set.seed(54321)
sam <- runif(50, min = 3, max = 100)
sam
```

Sample

```
sample(sam, size = 15)
```

To get the same sample repeatedly

```
set.seed(1234)
sample(sam, size = 15)
```

```
sort(sam)
```

```
rev(sort(sam))
```

```
quantile(sam)
```

Create two vectors and look at the similarities/differences between the two

```r
set.seed(2468)
vec_top_1 <- seq(2, by = 6, length = 100)
vec_top_2 <- seq(4, by = 4, length = 100)

vec_top_1

vec_top_2

union(vec_top_1, vec_top_2)

intersect(vec_top_1, vec_top_2)

setdiff(vec_top_1, vec_top_2)
```

9.3.4 Vector Operations

Vectors of equal length can be operated on together. If one vector is shorter, it will get recycled, as its elements are repeated until it matches the elements of the longer vector. When using vectors of unequal lengths, it would be ideal if the longer vector is a multiple of the shorter vector.

1. Basic Vector Operations

```r
vec_1 <- 1:10

vec_1 * 12 # multiplication

vec_1 + 12 # addition

vec_1 - 12 # subtraction

vec_1 / 3 # division

vec_1^4 # power

sqrt(vec_1) # square root
```

2. Operations on vectors

Operations on vectors of equal length Additionally, we can perform operations on two vectors of equal length.

a. Create two vectors

```r
vec_3 <- 5:14
vec_3
```

```
vec_4 <- 12:3
vec_4
```

b. Perform various arithmetic operations

```
vec_3 + vec_4
```

```
vec_3 - vec_4
```

```
vec_3 / vec_4
```

```
vec_3 * vec_4
```

```
vec_3 ^ vec_4
```

3. Functions that can be applied to vectors

The functions listed below can be applied to vectors:

a. `any()`

b. `all()`

c. `nchar()`

d. `length()`

e. `typeof()`

Examples

```
any(vec_3 > vec_4)
```

```
any(vec_3 < vec_4)
```

```
all(vec_3 > vec_4)
```

```
all(vec_3 < vec_4)
```

```
length(vec_3)
```

```
length(vec_4)
```

```
typeof(vec_3)
```

```
typeof(vec_4)
```

Determine the number of letters in a character

```
vec_5 <- c("R", "RStudio", "Shiny", "Quarto")
```

```
nchar(vec_5)
```

3. Vectors and mathmatical/statistical operations

A variety of mathematical operations can be performed on vectors. These operations together with examples are listed below.

a. Create a new vector

```
stat_math_vec_1 <- seq(2, by = 6, length = 120)
stat_math_vec_1
```

b. Perform mathematical operations on the vector

- Length

```
length(stat_math_vec_1)
```

- Sum

```
sum(stat_math_vec_1)
```

- Minimum

```
min(stat_math_vec_1)
```

- Maximum

```
max(stat_math_vec_1)
```

- Mean

```
mean(stat_math_vec_1)
```

- Median

```
median(stat_math_vec_1)
```

- Quantile

```
quantile(stat_math_vec_1)
```

- Standard Deviation

```
sd(stat_math_vec_1)
```

- Inter-Quartile Range (IQR)

```
IQR(stat_math_vec_1)
```

- Cumulative Sum

```
cumsum(stat_math_vec_1)
```

- Range

```
range(stat_math_vec_1)
```

- Cut This function cuts a range of values into bins and specifies labels for each bin.

The argument "breaks" can be the number of bins, or it can be specific break points.

```
cut(stat_math_vec_1, breaks = 6)
```

```
cut(stat_math_vec_1, breaks = c(0, 120, 240, 360, 480, 600, 720))
```

- Pretty

This function creates a sequence of equally spaced values that are rounded to the closest integer.The argument "n" lists the number of intervals.

```
pretty(stat_math_vec_1)
pretty(stat_math_vec_1, n = 20)
```

- Trigonometric functions

```
sin(stat_math_vec_1)
```

```
cos(stat_math_vec_1)
```

```
tan(stat_math_vec_1)
```

- Logarithmic functions

```
log(stat_math_vec_1)
```

```
log2(stat_math_vec_1)
```

```
log10(stat_math_vec_1)
```

4. Miscellaneous functions

a. Changing the case of a character vector

```
char_vec <- c("a", "b", "c", "d")
```

```
char_vec_upper <- toupper(char_vec)
char_vec_upper
```

```
char_vec_lower <- tolower(char_vec_upper)
char_vec_lower
```

4. Recycling of vectors

```
vec_3 + c(10, 20)
```

```
vec_3 + c(10, 20, 30)
# Will result in a warning as the longer vector is not a multiple of the shorter one
```

D) Accessing elements of a vector and subsetting

To access the elements of a vector, we can use numeric-, character-, or logical-based indexing.

Examples

1. Name the columns of a vector with names().

a) Create the vector.

```
vec_name <- 1:5
vec_name
```

b) Name the individual elements.

```
names(vec_name) <- c("a", "c", "e", "g", "i")
vec_name
```

2. Use the vector index to filter

```
vec_index <- 1:5
vec_index
```

a) Logical vector as an index

```
vec_index[c(TRUE, FALSE, TRUE, FALSE, TRUE)]
```

b) Filter vector based on an index

```
vec_index[1:3]
```

c) Access a vector using its position

```
vec_index[4]
```

With negative indexing, every element except the one specified is returned.

```
vec_index[-2]
```

```
vec_index[c(2,4)]
```

d) Modify a vector using indexing

```
vec_index

vec_index[5] <- 1000

vec_index
```

E) Create matrices by combining vectors by row and column

1. Create four new vectors

```
vec_mat_df_1 <- seq(2,20, length.out=10)
vec_mat_df_2 <- rep(2:6, each = 2)
vec_mat_df_3 <- runif(10, min = 3, max = 100)
vec_mat_df_4 <- sequence(10)

vec_mat_df_1
vec_mat_df_2
vec_mat_df_3
vec_mat_df_4
```

2. Create a new matrix by combining each vector as a column, `cbind()`.

```
new_matrix <- cbind(vec_mat_df_1, vec_mat_df_2, vec_mat_df_3, vec_mat_df_4)
new_matrix
str(new_matrix)
class(new_matrix)
```

3. Create a new matrix by combining each vector as a row, `rbind()`.

```
new_matrix <- rbind(vec_mat_df_1, vec_mat_df_2, vec_mat_df_3, vec_mat_df_4)
new_matrix
str(new_matrix)
class(new_matrix)
```

9.4 Exercises

i. Create a numeric vector with values 1 through 5 using `c()` and extract the middle element from the vector using the described subsetting methods.
ii. Generate a sequence of even numbers from 2 to 10 using the `seq()` function, and calculate the total sum and average of this new vector.

iii. Create a vector of student ages (between 16 and 20, n = 10), calculate their average and standard deviation, and identify students younger than 18.
iv. Use logical indexing to subset elements from a vector based on a condition.
v. Perform element-wise addition and multiplication on two newly created numeric vectors of the same length.
vi. Concatenate two newly created character vectors to create a longer vector.
vii. Create a named vector with elements representing days of the week.
viii. Use vector functions to find the length, sum, and mean of a numeric vector.
ix. Use the functions `sort()` or `rev()` to arrange elements in ascending, descending, or reversed order.
x. Use functions like `which()` or `match()` to locate elements based on conditions or matching values.
xi. How can you append elements to an existing vector?
xii. When binding vectors together, what is the difference between `cbind()` and `rbind()`?
xiii. Explore other vector capabilities like recycling, naming elements, and using vector lengths in your R code.

9.5 Summary

Vectors are an essential data structure for data analysis purposes. In this chapter, I have demonstrated the different types of vectors, how to generate vectors, and the various types of vector operations that can be performed. Now that we have a deeper understanding of vectors, it is time to explore other data structures (that are useful for data analysis) in the subsequent chapters.

10 Data Structures (Part I)

10.1 Questions

- List the data structures that can be used in R?
- How do we define the various data structures?
- What operations can be performed on the different data structures?
- How do these data structures differ from one another in terms of storage and functionality?

10.2 Learning Objectives

- Learn about data structures in R, specifically vectors, lists, and dataframes.
- Define and manipulate vectors, lists, and dataframes.
- Perform common operations on each data structure.
- Identify and differentiate between the diverse data structures offered by R.
- Choose the appropriate data structure based on the type and organization of your data.
- Understand the strengths and limitations of each structure for efficient analysis.

10.3 Lesson Content

10.3.1 Introduction

Data structures in R are the formats used to organize, process, retrieve, and store data. They help to organize stored data in a way that the data can be used more effectively. Data structures vary according to the number of dimensions and the data types (heterogeneous or homogeneous) contained.

The primary data structures are:

- Vectors
- Lists
- Dataframes
- Matrices
- Arrays
- Factors

In this lesson, we will review the 1st set of data structures: vectors, lists, and dataframes.

10.3.2 Vectors

Discussed in the previous chapter on "Vectors" (Chapter **??**)

10.3.3 Lists

Lists are objects/containers that hold elements of the same or different types. They can contain strings, numbers, vectors, dataframes, matrices, functions, or other lists. Lists are created with the list() function.

Examples:

a. Three element list

```
list_1 <- list(10, 30, 50)
```

b. Single element list

```
list_2 <- list(c(10, 30, 50))
```

c. Three element list

```
list_3 <- list(1:3, c(50,40), 3:-5)
```

d. List with elements of different types

```
list_4 <- list(c("a", "b", "c"), 5:-1)
```

e. List which contains a list

```
list_5 <- list(c("a", "b", "c"), 5:-1, list_1)
```

f. Set names for the list elements

```
names(list_5)
```

```
names(list_5) <- c("character vector", "numeric vector", "list")
```

```
names(list_5)
```

g. Access elements within a list

Return a list within a list

```
list_5[1]
```

```
list_5[[1]]
```

Return an individual element within a list of lists

```
list_5[[1]][1]
```

```
list_5[["character vector"]]
```

h. Length of list

```
length(list_1)
```

```
length(list_5)
```

Flatten out into a single vector using unlist()

```
unlist(list_5)
```

10.3.4 Dataframes

A data frame is one of the most common data objects used to store tabular data in R. Tabular data has rows representing observations and columns representing variables. Dataframes contain lists of equal-length vectors. Each column holds a different type of data, but within each column, the elements must be of the same type. The most common data frame characteristics are listed below:

- Columns should have a name,
- Row names should be unique,
- Various data can be stored (such as numeric, factor, and character), and,
- The individual columns should contain the same number of data items.

NOTE: Tibbles are the modern versions of dataframes, that have improved printing and subsetting features. The tibble package is a main component of the tidyverse; however, a review of the tidyverse is beyond the scope of this book and will be discussed in subsequent lessons.

10.3.4.1 Creation of data frames

```
level <- c("Low", "Mid", "High")
language <- c("R", "RStudio", "Shiny")
age <- c(25, 36, 47)

df_1 <- data.frame(level, language, age)
```

Use `stringsAsFactors = FALSE` to prevent R from converting string columns to factors.

```
df_2 <- data.frame(level, language, age, stringsAsFactors = FALSE)
```

10.3.4.2 Functions used to manipulate data frames

a. Number of rows

```
nrow(df_1)
```

b. Number of columns

```
ncol(df_1)
```

c. Dimensions

```
dim(df_1)
```

d. Class of data frame

```
class(df_1)
```

e. Column names

```
colnames(df_1)
```

f. Row names

```
rownames(df_1)
```

g. Top and bottom values

```
head(df_1, n=2)

tail(df_1, n=2)
```

h. Access columns / subset

```
df_1$level
```

i. Access individual elements / subset

```
df_1[3,2]
```

```
df_1[2, 1:2]
```

j. Access columns with index

```
df_1[, 3]
```

```
df_1[, c("language")]
```

k. Access rows with index

```
df_1[2, ]
```

NOTE: To view the internal structure of various data types described above, the learner can use the `str()` function.

Example

```
str(list_3)
```

```
str(df_1)
```

10.4 Exercises

i. Explain the characteristics and use cases of lists in R.

ii. Create a list containing a numeric vector, character vector, and logical vector.

iii. How can you add elements to a list in R?

iv. Explain what a data frame is and why it is widely used in R.

v. Write code to create a data frame with columns for "Name," "Age," and "Gender" for three individuals.

vi. How can you combine two data frames horizontally in R?

vii. Discuss various methods for subsetting elements in data structures.

viii. How do you check the structure of a data frame? What function is used?

ix. How do you access a specific column of a data frame?

x. What are rownames and colnames in a data frame? How are they useful?

xi. How do you add a new column to an existing data frame?

10.5 Summary

We have now covered half of the key data structures used in R. Knowledge of how to identify and manipulate vectors, lists, and dataframes will help the learner perform numerous computational tasks in R. Next, to complete our review of data structures, we will focus on matrices, arrays, and factors.

11 Data Structures (Part II)

11.1 Questions

- List the data structures that can be used in R?
- How do we define the various data structures?
- What operations can be performed on the different data structures?
- How do these data structures differ from one another in terms of storage and functionality?

11.2 Learning Objectives

- Learn about data structures in R, specifically matrices, arrays, and factors.
- Define and manipulate matrices, arrays, and factors.
- Perform common operations on each data structure.
- Identify and differentiate between the diverse data structures offered by R.
- Choose the appropriate data structure based on the type and organization of your data.
- Understand the strengths and limitations of each structure for efficient analysis.

11.3 Lesson Content

11.3.1 Introduction

Data structures in R are the formats used to organize, process, retrieve, and store data. They help to organize stored data in a way that the data can be used more effectively. Data structures vary according to the number of dimensions and the data types (heterogeneous or homogeneous) contained.

The primary data structures are:

- Vectors
- Lists
- Data frames
- Matrices
- Arrays
- Factors

In this lesson, we will review the 2nd set of data structures: matrices, arrays, and factors.

11.3.2 Matrices

A matrix is a rectangular two-dimensional (2D) homogeneous data set containing rows and columns. It contains real numbers that are arranged in a fixed number of rows and columns. Matrices are generally used for various mathematical and statistical applications.

a. Creation of matrices

Using the `matrix()` function

Format: `matrix(range, nrow = _, ncol = _)`

```
m1 <- matrix(1:9, nrow = 3, ncol = 3)
m2 <- matrix(21:29, nrow = 3, ncol = 3)
m3 <- matrix(1:12, nrow = 2, ncol = 6)
m1
m2
m3
```

Using `cbind()` and `rbind()`

b. Obtain the dimensions of the matrices `m1` and `m3`

```
nrow(m1)
ncol(m1)
dim(m1)

nrow(m3)
ncol(m3)
dim(m3)
```

c. Arithmetic with matrices

```
m1 + m2
m1 - m2
m1 * m2
m1 / m2
m1 == m2
```

d. Matrix multiplication

```
m5 <- matrix(1:10, nrow = 5)
m6 <- matrix(43:34, nrow = 5)
m5
m6

m5 * m6
```

Knowledge of dimensions is very important when working with matrices. Here, we observe that we can't multiply m5 by m6 because of the matrix dimensions

```
dim(m5)
dim(m6)

# m5 %*% m6 This will not execute because the dimensions are not appropriate
```

The vector m6 needs to be transposed before multiplication.

Transpose

```
dim(m5)
dim(t(m6))
```

Now we have a 5 by 2 matrix that can be multiplied by a 2 by 5 matrix.

```
m5%*%t(m6)
```

e. Generate an identity matrix

```
diag(5)
```

f. Column and row names

Dimensions of `m5`

```
dim(m5)
```

Current column names of `m5`

```
colnames(m5)
```

Set column names

```
colnames(m5) <- c("AA", "BB")
```

Display the matrix `m5` and new column names

```
m5
colnames(m5)
```

Set row names

Get the dimensions of `m6`

```
dim(m6)
```

Display the current row names of `m6`

```
rownames(m6)
```

Set the row names of `m6`

```
rownames(m6) <- c("ABC", "BCA", "CAB", "ACB", "BAC")
```

```
m6
rownames(m6)
```

g. Arithmetic within matrices

```
colSums(m5)
```

```
rowSums(m6)
```

h. Subsetting matrices

```
subset_m5 <- m5[1:4, 1:2]
subset_m5
```

```
subset_m6 <- m6[3:5, 1:2]
subset_m6
```

i. Matrix division

When you divide a matrix by a vector, the operation is row-wise.

```
m5 / m6
```

11.3.3 Arrays

An array is a multidimensional vector that stores homogeneous data. It can be thought of as a stacked matrix and stores data in more than 2 dimensions (n-dimensional). An array is composed of rows by columns by dimensions.

Example: an array with dimensions, dim = c(2,3,3), has 2 rows, 3 columns, and 3 matrices.

b. Creating arrays

```
arr_1 <- array(1:12, dim = c(2,3,2))
```

```
arr_1
```

b. Filter array by index

```
arr_1[1 ,  , ]
```

```
arr_1[1, ,1]
```

```
arr_1[, , 1]
```

11.3.4 Factors

Factors are used to store integers or strings which are categorical. They categorize data and store the data in different levels. This form of data storage is useful for statistical modelling. Examples include TRUE or FALSE and male or female. Useful for handling qualitative datasets.

NOTE: R has a more efficient method for handling categorical variables using the `forcats`package. This will be discussed in a subsequent series focusing on the `tidyverse`.

```
vector <- c("Male", "Female")
```

```
factor_1 <- factor(vector)
factor_1
```

OR

```
factor_2 <- as.factor(vector)
factor_2
```

```
as.numeric(factor_2)
```

NOTE: To view the internal structure of various data types described above, the learner can use the `str()` function.

Example

```
str(m5)
```

```
str(arr_1)
```

```
str(factor_1)
```

11.4 Exercises

- How do you create a matrix in R? Give an example.
- What are the column and row dimensions of a matrix called in R?
- How do you access elements in a matrix?
- Generate a 3x3 identity matrix using matrix().
- Write code to create a 3x3 matrix with sequential numeric values.
- Convert a character vector to a factor with 3 levels.
- Describe the purpose of factors in R.
- How do you create a factor variable with specified levels?
- What is an array, and how does it differ from a matrix in R?
- Provide an example of creating a three-dimensional array.

11.5 Summary

In this chapter, we have completed our review of data structures. These data structures have different properties that influence how they are used in various computational tasks. Additionally, we have looked at the strengths and limitations of each structure for data analysis. However, we haven't discussed an important aspect of data structures: what do we do with missing data? In the final chapter, we will tackle this issue and provide solutions.

12 Handling missing data

12.1 Questions

- What is missing data?
- Why is missing data a common challenge in data analysis?
- How can we identify and handle missing data effectively in R?
- What are the consequences of ignoring or mishandling missing data?
- What are the best practices for imputing missing values?
- When should I choose deletion, imputation, or alternative approaches?

12.2 Learning Objectives

- Define the main types of missing data.
- Identify missing data in R and assess its impact on analyses.
- Work with datasets that contain missing data.
- Understand the importance of addressing missing data in data analysis.
- Apply best practices for handling missing data in your R projects.
- Apply functions like `is.na()` and `complete.cases()` to identify `NA` values.

12.3 Lesson Content

12.3.1 Introduction

One of the most common tasks/activities in data analysis is handling missing values. Missing data are found everywhere; therefore, it is important to learn how to identify and summarize them. Missing data can arise from different sources, including nonresponses in surveys or technical difficulties during data collection.

- Unanswered survey questions
- Data entry errors
- Errors in measurements
- Data merge issues

Missingness presents itself in various ways in R. The most common way to represent a missing value in R is with the value `NA`. Other reserved values include:

- `NULL` = neither TRUE or FALSE (`is.null()` identifies a null value and `as.null()` converts to a null value).
- `NaN` = impossible values
- `Inf` = infinite value obtained by dividing by zero (`is.infinite()` identifies an infinite value and `as.infinite()` converts to an infinite value).

Examples of the missing values

`NA = 1/NA`

`NaN = 0/0`

`Inf = 10/0`

`NULL = 5/Inf`

Common strategies for handling missing data

When investigating a dataset, there are several functions that can assist with the process. To find missing and non-missing values, the user can work with `is.na()` and `!is.na()`, respectively. If any rows contain missing values, one can use `na.omit()` or `drop_na()`. If there are missing values in the dataset, the arguments `na.rm = TRUE`, `remove_na = TRUE`, or the function `complete.cases()` can be used to remove missing values.

- Deletion: Remove rows or columns with missing values (simplest method but it can lead to bias).
- Imputation: Replace missing values with estimated values based on statistical methods (can introduce artificial data).

R has two types of missing data, `NA` and `NULL`.

Missing values can represent a fixed value like 0 (zero) or a negative number.

Missing can be `NA` or `NaN` and represented by `is.na()` or `is.nan()`

12.3.2 NA

R uses `NA` to represent missing data. The `NA` appears as another element of a vector. To test each element for missingness we use `is.na()`. Generally, we can use tools such as `mi`, `mice`, and `Amelia` (which will be discussed later) to deal with missing data. The deletion of this missing data may lead to bias or data loss, so we need to be careful when we choose this option. In subsequent blog posts, we will look at the use of imputation to deal with missing data.

12.3.3 NULL

NULL represents nothingness or the "absence of anything". 2 It does not mean missing but represents nothing. NULL cannot exist within a vector because it disappears.

12.4 Exercises

i. Identify missing values in a dataset using the `is.na()` and `complete.cases()` functions.

ii. Practice removing missing data using the `na.omit()` function.

iii. How can you handle missing values when calculating summary statistics like mean or median?

iv. If you take the mean of a vector with `NA` values, what will the result be?

v. How are missing values represented in R?

vi. What is `NA` in R? How is it different from `NULL`?

vii. What are some common reasons you may get `NA` values in a dataset?

viii. How can you check if a value is `NA` in R?

12.5 Summary

The learner should now have a firm grasp of what missing data is and why it is a common challenge in data analysis. Additionally, after completing this chapter, the learner should know how to identify and handle missing data with various R functions. This is the final chapter in the book, and I hope you have enjoyed your learning journey so far. In the next chapter, I provide a conclusion to the book and some next steps you can take to continue on your R learning journey.

Conclusion

We have finally come to the end of the "R for Novice Programmers" book. Congratulations on completing this learning journey!

The overall goal of this book was to introduce non-programmers or those with very little programming experience to R and RStudio and their use in basic statistical programming. It is hoped that the learners gained new skills and discovered how this software could be used to simplify everyday tasks in their workplaces. The learner should now be able to understand all the topics listed below based on the previously described learning objectives.

Table 12.1: Syllabus

Chapter	Title	Date Completed
	Introduction	
1	Overview of R and RStudio	
2	Download and Install R and RStudio	
3	Navigating the R and RStudio interfaces	
4	Managing your files and data	
5	Importing data and saving analysis outputs	
6	Basic arithmetic, arithmetic operators, and variables	
7	The primary types of operators in R	
8	Data Types	
9	Vectors	
10	Data Structures (Part I)	
11	Data Structures (Part II)	
12	Handling missing data	
	Conclusion	
	Appendix	

To maintain mastery of the course material, the learner is encouraged to practice consistently, explore freely available resources, get involved in real-world projects, and work with the large R community. In future sessions, the learner will focus on specific topics such as data wrangling/manipulation, data visualization, and programming. Therefore, I would encourage the learner to review some of the links I have provided in the Appendix and look out for more books in the future which will cover:

i. Data visualization
ii. Tidyverse
iii. Programming: Functions, Loops, and Control Statements
iv. Data Science
v. Statistics
vi. Machine Learning and Artificial Intelligence
vii. Publishing with R Markdown and/or Quarto
viii. R Shiny

Thanks for your participation and for completing this book!