R4DataScience

A Course offered by Department of Statistics, University of Calicut

Mahesh Divakaran

2026-01-14

Table of contents

1 Introduction

1.1 R for Data Analysis: From Fundamentals to Advanced Applications

A Course Offered by the Department of Statistics, University of Calicut

---

1.2 About This Book

This book, *R for Data Analysis: From Fundamentals to Advanced Applications*, has been developed by Mahesh Divakaran as part of an certificate course offered by the **Department of Statistics, University of Calicut**. It is designed to serve as a **comprehensive learning resource** for students who wish to build strong practical and theoretical skills in data analysis using the R programming language.

The material is written in the form of a **Quarto book**, enabling seamless integration of explanations, executable R code, figures, and outputs in a single reproducible document. This approach reflects modern statistical practice, where analysis, interpretation, and reporting are inseparable.

1.3 Motivation for the Course

In today's data-driven world, statistics is no longer confined to theory alone. Statisticians are expected to:

- Handle real-world datasets
- Perform exploratory and confirmatory data analysis
- Apply appropriate statistical models
- Communicate results clearly and reproducibly

R has emerged as one of the most powerful and widely used tools to meet these demands. This course is motivated by the need to **bridge the gap between statistical theory and practical data analysis**, enabling students to translate classroom concepts into real analytical workflows.

1.4 Why R for Data Analysis?

R is a language developed by statisticians, for statisticians. Its design philosophy aligns closely with statistical thinking and data-centric problem solving.

Key reasons for choosing R as the primary tool in this course include:

- A rich ecosystem for statistical methods and modeling
- Strong support for data visualization and exploratory analysis
- Open-source nature and global community support
- Integration of computation, graphics, and reporting
- Wide acceptance in academia, research, and industry

By learning R, students gain a **transferable skill** that is relevant across disciplines such as economics, health sciences, social sciences, environmental studies, and machine learning.

1.5 Course Philosophy and Learning Approach

This course adopts a **hands-on, example-driven learning approach**. Rather than treating R as a collection of commands to memorize, the emphasis is on:

- Understanding data structures and workflows
- Writing clear, readable, and reusable code
- Interpreting results in a statistical context
- Developing reproducible analysis habits

Each concept is introduced with intuition, followed by practical examples and gradually extended to more complex applications. Errors, warnings, and unexpected outputs are treated as learning opportunities rather than obstacles.

1.6 Intended Audience

This book is primarily intended for:

- Undergraduate and postgraduate students of Statistics
- Students from allied disciplines with a quantitative background
- Beginners with little or no prior programming experience

A basic understanding of statistics will be helpful, but **no prior knowledge of R or programming is assumed**. The material progresses from fundamental concepts to advanced analytical techniques in a structured manner.

1.7 Course Structure and Progression

The book is organized to support **progressive learning**, moving from foundational concepts to advanced applications.

Broadly, the course covers:

1. Introduction to R, RStudio, and the computing environment
2. Core data structures and data handling in R
3. Data import, cleaning, and transformation
4. Exploratory data analysis and visualization
5. Statistical modeling and inference
6. Advanced topics and applied case studies
7. Reproducible reporting using Quarto

Each chapter builds on previous knowledge, ensuring continuity and conceptual clarity.

1.8 Reproducible Data Analysis with Quarto

A defining feature of this course is the emphasis on **reproducible research**. All analyses in this book are written using Quarto, allowing:

- Code and results to coexist in one document
- Automatic generation of tables and figures
- Easy export to PDF, HTML, and Word formats

This practice encourages transparency, accuracy, and professional documentation—skills essential for academic research and industry projects alike.

1.9 Learning Outcomes

By the end of this course, students will be able to:

- Use R confidently for data analysis tasks
- Understand and manipulate different data structures
- Perform exploratory and inferential statistical analysis
- Visualize data effectively
- Write reproducible and well-documented analysis reports
- Apply R to real-world datasets and research problems

1.10 How to Use This Book

To get the most out of this book:

- Read the explanations carefully and run the code yourself
- Modify examples and observe how results change
- Practice regularly using the exercises provided
- Focus on understanding workflows rather than isolated commands

Active engagement is essential for mastering data analysis with R.

1.11 Concluding Remarks

R for Data Analysis: From Fundamentals to Advanced Applications is more than a programming guide—it is an introduction to **modern statistical practice**. Through this course, students will develop the skills needed to analyze data rigorously, think statistically, and communicate findings effectively using R.

The chapters that follow begin with the basics of R and RStudio, laying the foundation for advanced analytical techniques explored later in the book.

2 About

2.1 About the Author

Mahesh Divakaran is a Data Scientist, Statistical Programmer, Trainer, and Public Speaker with over **eight years of professional experience** in statistics, data science, and clinical research. He is the author of this course material *R for Data Analysis: From Fundamentals to Advanced Applications*, offered by the **Department of Statistics, University of Calicut**.

He specializes in **clinical data analysis using R**, supported by a strong foundation in statistical theory and applied analytics. He is currently pursuing a **Ph.D. in Statistics at Amity University**, with a research focus on advanced data analysis techniques.

Mahesh has worked extensively in the clinical research and healthcare analytics domain. He is currently a **Senior Statistical Programmer at IQVIA**, where he leads the development and validation of R packages for clinical studies. Previously, he served as **Associate Lead – Clinical Data Analytics at Genpro Research**, contributing to the development of R Shiny dashboards and automated reporting systems for clinical trials.

His technical expertise includes **R, Python, SAS, R Shiny, Tableau, and Power BI**, with particular strength in ADaM dataset creation, statistical reporting, and data visualization. Alongside industry work, he is deeply committed to teaching and capacity building in **Statistics, Data Science, and Clinical Data Analytics**, and has delivered invited talks and presentations at national and international conferences.

2.2 About the Department of Statistics, University of Calicut

The **Department of Statistics, University of Calicut**, was established in **1988** with the objective of developing into a **centre of excellence** in statistics and its applications in the region. The department was founded under the leadership of **Dr. K. Kumaranakutty**, the founder professor, and is rooted in a philosophy that emphasizes the close relationship between the University and the community, promoting a scientifically motivated society.

Since its inception, the department has been actively involved in **teaching, training, and research**, with a strong emphasis on modeling real-life problems using rigorous statistical methods. The faculty members engage in collaborative research with scientists and institutions within India and abroad. Several faculty members have been recipients of **Commonwealth Scholarships and Fellowships** tenable in the United Kingdom.

The department also plays a vital service role within the University by advising and assisting faculty members and research scholars from other departments in the **statistical analysis of research data**.

As of 2010, **27 research scholars** from the department have been awarded the **Ph.D. degree**. Research output from the department is nationally and internationally recognized, as reflected in peer-reviewed publications, funded projects, and invited lectures at academic conferences.

2.2.1 Thrust Areas

The major thrust areas of the department include:

- Stochastic Processes and Applications
- Reliability and Survival Analysis
- Extreme Value Theory
- Time Series Analysis and Modelling
- Applied Probability
- Distribution Theory
- Sampling Theory
- Novelty Detection and Neural Networks

2.2.2 Courses Offered

The department offers the following academic programmes:

1. **M.Sc. Statistics** – 4 Semesters (Choice Based Credit Semester System – CCSS)
2. **M.Phil.** – 2 Semesters
3. **Ph.D. Programme**

2.2.2.1 M.Sc. Statistics Programme

The M.Sc. Statistics programme is designed to provide **intensive training** covering the core and applied areas of statistics. The curriculum equips students with technical and analytical skills required for immediate employment as statisticians, while also preparing them for advanced research.

- **Current Specialization Module**: Advanced Statistical Analysis
- **Total Intake**: 25 students
- **Admission Criteria**: Candidates with a B.Sc. degree securing at least 50% marks, with either (i) Statistics as the main subject, or (ii) Mathematics as the main subject with Statistics as a subsidiary. Admission is based on performance in the common university entrance examination.

Graduates of the programme find employment opportunities in **research organizations, government agencies, industry, healthcare, analytics, and education**.

2.2.3 Faculty

- **Dr. K. Jayakumar** – Senior Professor
- **Dr. Krishnarani S. D.** – Associate Professor & Head
- **Dr. Dileepkumar M.** – Assistant Professor

Former Faculty Members:

- Dr. C. Chandran
- Dr. M. Manoharan
- Dr. N. Raju
- Dr. K. Kumaranakutty

2.2.4 Infrastructure and Facilities

The department is well-equipped with academic and research infrastructure. It has its own **departmental library**, housing over **2,100 books** and several volumes of classical statistical journals. The **computer laboratory** is equipped with more than **twenty personal computers** and statistical software such as **SPSS and SYSTAT**, along with teaching aids including OHP and LCD projectors. The department also maintains an active **Alumni Association**.

2.2.5 Research and Collaborations

Research in the department is held in high regard at national and international levels. Faculty members have received prestigious fellowships, including Commonwealth Scholarships and research fellowships in Canada. The department maintains active research collaborations with academic institutions, industry partners, and government organizations.

Major research interests include **Stochastic Processes, Reliability and Survival Analysis, Distribution Theory, Time Series Modelling, Applied Probability, Extreme Value Theory, Novelty Detection, and Neural Networks**.

Further details about the department and its academic programmes are available at **http://www.cuonline.ac.in/**.

3 Introduction to R and RStudio

3.1 What is R?

R is a**programming language and software environment**designed primarily for**statistical computing, data analysis, and visualization**. It is widely used in academia, research institutions, government organizations, and industry for tasks ranging from simple data summaries to advanced machine learning and reproducible research.

Key characteristics of R include:

- **Open-source and free**: Anyone can use, modify, and distribute R.
- **Strong statistical foundation**: Built by statisticians for statistical work.
- **Extensive package ecosystem**: Thousands of user-contributed packages extend R's capabilities.
- **Excellent graphics**: R is known for high-quality, publication-ready visualizations.
- **Reproducibility**: Scripts and documents ensure analyses can be repeated and verified.

In this workshop, R will be used as a**tool for thinking with data**, not just a calculator. You will learn how to write clear, reusable code that performs data analysis systematically.

3.2 Why Use R for Data Analysis?

R is especially well-suited for data analysis because it combines**data manipulation, statistical modeling, and visualization**in one environment.

3.2.1 Advantages of R

- Handles**small to very large datasets**efficiently
- Supports**classical statistics, modern data science, and machine learning**
- Encourages**script-based analysis**, reducing manual errors
- Integrates well with reports (PDF, Word, HTML) through**Quarto and R Markdown**
- Strong community support and documentation

3.2.2 Typical Use Cases

- Exploratory Data Analysis (EDA)
- Statistical inference and hypothesis testing
- Regression and multivariate analysis
- Time series and forecasting
- Data visualization and dashboards
- Academic research and thesis work

3.3 Installing R

R itself is the**core engine**that performs computations.

3.3.1 Steps to Install R

1. Visit the official Comprehensive R Archive Network (CRAN):https://cran.r-project.org
2. Choose your operating system (Windows / macOS / Linux)
3. Download and install the latest stable version

After installation, R can be run from the system console. However, working directly in the R console is not ideal for beginners or large projects. This is where**RStudio**becomes essential.

3.4 What is RStudio?

RStudio is an**Integrated Development Environment (IDE)**for R. It makes working with R easier, faster, and more organized.

RStudio does**not replace R**. Instead:

> R does the computation, and RStudio provides a user-friendly interface to work with R.

3.4.1 Why Use RStudio?

- Code editor with syntax highlighting
- Easy file and project management
- Built-in help and documentation
- Integrated plotting and visualization
- Seamless support for Quarto documents

3.5 Understanding the RStudio Interface

When you open RStudio, you typically see**four panes**:

3.5.1 1. Script Editor (Top-Left)

- Where you write and save R code
- Supports`.R`scripts and`.qmd`Quarto files
- Code is written once and executed many times

3.5.2 2. Console (Bottom-Left)

- Where R executes commands
- Displays output, messages, and errors
- Useful for quick calculations and testing code

3.5.3 3. Environment / History (Top-Right)

- Shows objects currently in memory
- Displays datasets, variables, and functions
- Helps track what data is loaded

3.5.4 4. Files / Plots / Help / Viewer (Bottom-Right)

- File browser for project files
- Plot display window
- Help documentation for functions
- Viewer for Quarto and HTML outputs

Understanding these panes is crucial for efficient workflow in R.

3.6 Your First Interaction with R

Let us start with very simple commands.

3.6.1 Using R as a Calculator

```
2 + 3
```

```
[1] 5
```

```
10 / 2
```

```
[1] 5
```

```
5^2
```

```
[1] 25
```

R immediately evaluates expressions and returns results.

3.6.2 Assigning Values to Objects

```
x <- 10
y <- 5
x + y
```

```
[1] 15
```

Here:

- `<-`is the**assignment operator**
- `x`and`y`are objects stored in memory

Object-based thinking is fundamental in R.

3.7 Basic Data Types in R

R works with different types of data. The most common ones are:

- **Numeric**: Numbers (e.g., 10, 3.5)
- **Character**: Text (e.g., "Statistics")
- **Logical**: TRUE or FALSE

Examples:

```
age <- 21
name <- "Anita"
passed <- TRUE
```

You can check the type of an object using:

```
class(age)
```

```
[1] "numeric"
```

```
class(name)
```

```
[1] "character"
```

```
class(passed)
```

```
[1] "logical"
```

3.8 Vectors: The Core Data Structure

A vector is a**collection of values of the same type**.

```
marks <- c(78, 85, 90, 88)
names <- c("A", "B", "C")
```

Key points:

- `c()`stands for*combine*
- All elements in a vector must be of the same type
- Vectors are the building blocks of most R data structures

3.9 Working Directory and Projects

The**working directory**is the default location where R reads and saves files.

```
getwd()
setwd("path/to/your/folder")
```

However, changing directories manually is discouraged.

3.9.1 RStudio Projects (Recommended)

- Create a project for each workshop, assignment, or research work
- Keeps scripts, data, and outputs organized
- Ensures reproducibility across systems

3.10 Introduction to Quarto

Quarto is a**next-generation scientific and technical publishing system**that allows you to combine:

- R code
- Text explanation
- Tables and figures

into a**single, reproducible document**.

3.10.1 Why Quarto for This Workshop?

- Generates**PDF, HTML, and Word**outputs
- Ideal for teaching, reports, and books
- Supports executable R code chunks

Example of an R code chunk in Quarto:

summary(c(10, 20, 30, 40))

3.11 How This Book Is Structured

This Quarto book is designed for**beginners**and progresses gradually:

1. Basics of R and RStudio
2. Data structures and data import
3. Data manipulation and visualization
4. Statistical analysis
5. Reproducible reporting with Quarto

Each chapter includes:

- Clear explanations
- Annotated examples
- Practice exercises

3.12 Best Practices for Learning R

- Type the code yourself instead of copy–paste
- Read error messages carefully
- Experiment with small examples
- Save scripts regularly
- Ask*why*a result occurs, not just*what*it is

Learning R is not about memorizing commands, but about developing**data analysis thinking**.

3.13 Getting Help

R has extensive built-in help and a supportive community. Here are some ways to get help: - **Help files**: Use `?function_name` or `help(function_name)` to access documentation for any function. For example:

```
?mean
```

- **RStudio Help Pane**: Use the Help pane in RStudio to search for functions and topics.
- **Online Resources**: Websites like Stack Overflow, R-bloggers, and the RStudio Community are great places to ask questions and find solutions.
- **Books and Tutorials**: There are many free and paid resources available for learning R. Remember, seeking help is a normal part of learning programming!

4 R Environment, Syntax, and Operators

4.1 Introduction

Before performing any data analysis, it is essential to ensure that the R environment is correctly installed and functioning. This chapter introduces the basic diagnostics for the R environment, followed by a detailed discussion of R syntax, assignment operators, variable naming conventions, and arithmetic operations. These concepts form the foundation for writing clear and correct R programs.

4.2 R Environment Diagnostic Checks

A properly configured R environment ensures smooth execution of scripts and reproducible results.

4.2.1 Checking the R Version

```
R.version.string
```

```
[1] "R version 4.5.2 (2025-10-31)"
```

This command displays the installed R version. Using an up-to-date version of R is recommended for compatibility with modern packages.

4.2.2 Testing Package Installation

```
install.packages("ggplot2")
```

```
The following package(s) will be installed:
- ggplot2 [4.0.1]
These packages will be installed into "~/Documents/Personal/R/R4Data Science_CU/R4DataScience_C
4.5/aarch64-apple-darwin20".

# Installing packages --------------------------------------------------------
```

```
- Installing ggplot2 ...                        OK [linked from cache]
Successfully installed 1 package in 3.6 milliseconds.
```

Successful installation confirms that R can access CRAN repositories and install external packages.

4.2.3 Verifying Basic Computation

```
(50 + 50) / 2
```

```
[1] 50
```

This simple arithmetic check confirms that the R interpreter is functioning correctly.

4.2.4 Checking TinyTeX Installation

```
tinytex::is_tinytex()
```

```
[1] TRUE
```

TinyTeX is required for generating PDF outputs from Quarto documents. This command verifies its availability.

4.3 Assignment Operators in R

R provides multiple ways to assign values to objects.

```
x <- 5
y = 10
11 -> z
```

- `<-` is the **recommended assignment operator** in R
- `=` is commonly used but should be avoided in complex expressions
- `->` assigns values from right to left

4.3.1 Global Assignment Operator

```
z <<- 15
```

The`<<-`operator assigns a value to a variable in the global environment. Its use should be limited, as it can make code harder to debug.

4.4 Variable Naming Conventions

Valid variable names in R:

```
var1 <- 10
var_2 <- 20
var.3 <- 30
Var4 <- 40
varFive <- 50
```

Key rules:

- Variable names are**case-sensitive**
- They may contain letters, numbers, dots (`.`), and underscores (`_`)
- They must not begin with a number

Invalid variable names contain spaces or special characters and will produce errors.

4.5 Fundamental R Syntax

R syntax is flexible and generally ignores whitespace.

```
x <- 10
y <- 20
z <- x + y

a = 5; b = 15; c = a * b
d=x-y
e = x * y; f = x / y
```

Although whitespace is optional,**consistent formatting**improves readability and maintainability.

4.6 Basic Arithmetic Operations

R supports all standard arithmetic operations.

```
5 + 3      # Addition
```

```
[1] 8
```

```
10 - 4     # Subtraction
```

```
[1] 6
```

```
6 * 7      # Multiplication
```

```
[1] 42
```

```
20 / 5     # Division
```

```
[1] 4
```

```
2 ^ 3      # Exponentiation
```

```
[1] 8
```

```
10 %% 3    # Modulus
```

```
[1] 1
```

```
10 %/% 3   # Integer division
```

```
[1] 3
```

These operations are frequently used in data transformation and statistical computation.

4.7 Summary

In this chapter, you learned how to:

- Verify and diagnose the R environment
- Use different assignment operators
- Apply valid variable naming conventions
- Understand fundamental R syntax
- Perform arithmetic operations in R

These basics are essential for writing correct and efficient R programs. The next chapter explores**data types and type conversion**, which are crucial for understanding how R stores and processes data.

5 Data Types, Type Conversion, and Basic Functions

5.1 Introduction

Understanding data types is fundamental to effective data analysis in R. This chapter introduces the core data types supported by R, explains how R stores and interprets data, and demonstrates type conversion and commonly used built-in functions.

5.2 Numeric and Integer Data Types

5.2.1 Numeric

```
num_var <- 42.5
class(num_var)
typeof(num_var)
```

By default, numeric values in R are stored as **double-precision numbers**.

5.2.2 Integer

```
int_var <- 42L
class(int_var)
typeof(int_var)
```

Appending `L` explicitly creates an integer. Although integers and numerics behave similarly, they differ in memory representation.

5.3 Character Data Type

```
char_var <- "Hello, R!"
class(char_var)
typeof(char_var)
```

R supports both single and double quotes for character strings. Escape characters allow quotes to be included within strings.

5.4 Logical Data Type

```
log_var <- TRUE
log_var2 <- FALSE
```

Logical values are essential for conditional operations and filtering data. Shorthand forms `T` and `F` exist but are not recommended in professional code.

5.5 Complex and Raw Data Types

5.5.1 Complex

```
comp_var <- 3 + 4i
class(comp_var)
typeof(comp_var)
```

Complex numbers include an imaginary component and are mainly used in specialized mathematical computations.

5.5.2 Raw

```
raw_var <- charToRaw("Hello")
rawToChar(raw_var)
```

Raw data represents bytes and is rarely used in routine data analysis.

5.6 Type Conversion in R

R provides explicit functions for converting between data types.

```
as.character(123.45)
as.numeric("678.90")
as.integer(99.99)
as.logical(1)
```

Type conversion is common when importing data from external sources.

5.7 Date, Factor, and NULL

5.7.1 Date

```
date_var <- as.Date("2024-01-01")
class(date_var)
```

Dates are crucial for time-based analysis.

5.7.2 Factor

```
factor_var <- factor(c("Red", "Blue", "Green"))
levels(factor_var)
```

Factors are used to represent categorical variables and play an important role in statistical modeling.

5.7.3 NULL

```
null_var <- NULL
is.null(null_var)
```

`NULL` represents the absence of a value and is commonly used in programming logic.

5.8 Special Numeric Values

```
Inf
-Inf
NaN
```

These values represent infinity and undefined numerical results. Functions such as `is.nan()` and `is.infinite()` help identify them.

5.9 Basic Functions in R

Frequently used built-in functions include:

```
length(vec)
class(vec)
summary(vec)
str(vec)
mean(vec)
sum(vec)
sd(vec)
```

These functions are essential for exploratory data analysis.

5.10 Sequences and Repetition

```
seq(1, 10, by = 2)
rep(1:3, times = 2)
```

Sequences and repetition are commonly used for simulations and indexing.

5.11 Accessing Help and Documentation

```
?mean
help.search("regression")
vignette()
```

R provides extensive documentation that should be consulted regularly.

5.12 Working Directory Management

```
getwd()
list.files()
```

Managing the working directory ensures that data files and outputs are correctly located.

5.13 Summary

In this chapter, you learned about:

- Core data types in R
- Differences between `class()` and `typeof()`
- Explicit type conversion
- Special numeric values
- Essential built-in functions
- Help and file management utilities

These concepts are critical for data handling and analysis. The next chapter will focus on **data structures such as vectors, lists, and data frames**, which build upon the data types introduced here.

6 Vectors, Lists, Matrices, and Data Frames

6.1 Introduction

Data structures are the backbone of data analysis in R. While Chapter 5 introduced basic data types, this chapter focuses on **how data is organized and stored** using R's fundamental data structures: **vectors, lists, matrices, and data frames**. Understanding these structures is essential for efficient data manipulation, analysis, and modeling.

6.2 Vectors

A **vector** is the most basic data structure in R. It is a collection of elements of the **same data type**.

6.2.1 Creating Vectors

```
num_vec <- c(10, 20, 30, 40)
char_vec <- c("A", "B", "C")
log_vec <- c(TRUE, FALSE, TRUE)
```

The function `c()` stands for *combine*.

6.2.2 Properties of Vectors

- All elements must be of the **same type**
- Vectors are **one-dimensional**
- R performs **type coercion** if mixed types are used

```
mix_vec <- c(1, "A", TRUE)
class(mix_vec)
```

6.3 Accessing Vector Elements

Elements in a vector are accessed using **indexing**, which starts at 1 in R.

```
num_vec[1]
num_vec[2:4]
num_vec[c(1, 3)]
```

6.3.1 Logical Indexing

```
num_vec[num_vec > 20]
```

6.4 Vector Operations

Vectors support element-wise operations.

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)

x + y
x * y
```

6.4.1 Recycling Rule

```
x <- c(1, 2, 3, 4)
y <- c(10, 20)
x + y
```

R recycles the shorter vector. A warning appears if lengths are incompatible.

6.5 Lists

A **list** is a flexible data structure that can contain elements of **different types and lengths**.

6.5.1 Creating Lists

```
my_list <- list(
  numbers = c(1, 2, 3),
  text = "Statistics",
  logical = TRUE,
  data = c("A", "B")
)
```

6.5.2 Accessing List Elements

```
my_list[[1]]
my_list$numbers
```

- `[[ ]]` extracts a single element
- `$` accesses elements by name

6.6 Matrices

A **matrix** is a two-dimensional data structure where all elements are of the **same type**.

6.6.1 Creating a Matrix

```
mat <- matrix(1:6, nrow = 2, ncol = 3)
mat
```

6.6.2 Matrix by Row

```
mat_row <- matrix(1:6, nrow = 2, byrow = TRUE)
mat_row
```

6.6.3 Accessing Matrix Elements

```
mat[1, 2]
mat[, 1]
mat[2, ]
```

6.7 Matrix Operations

Matrices support mathematical operations.

```
A <- matrix(1:4, nrow = 2)
B <- matrix(5:8, nrow = 2)

A + B
A * B        # Element-wise multiplication
A %*% B     # Matrix multiplication
```

6.8 Data Frames

A **data frame** is the most commonly used data structure in data analysis. It is a table-like structure where:

- Columns can have **different data types**
- All columns must have the **same length**

6.8.1 Creating a Data Frame

```
df <- data.frame(
  ID = 1:4,
  Name = c("A", "B", "C", "D"),
  Marks = c(78, 85, 90, 88),
  Passed = c(TRUE, TRUE, TRUE, FALSE)
)
```

6.9 Accessing Data Frame Elements

```
df$Marks
df[1, ]
df[, 2]
df[ , "Name"]
```

6.9.1 Structure of a Data Frame

```
str(df)
summary(df)
```

6.10 Adding and Removing Columns

```
df$Grade <- c("B", "B", "A", "B")
df$Passed <- NULL
```

6.11 Converting Between Data Structures

```
as.list(num_vec)
as.matrix(df)
as.data.frame(mat)
```

Understanding conversions helps avoid common errors in analysis.

6.12 Summary

In this chapter, you learned:

- How vectors store homogeneous data
- How lists store heterogeneous data
- How matrices represent two-dimensional numeric data
- How data frames organize real-world datasets
- Methods to access, modify, and convert data structures

These data structures form the foundation for data manipulation and analysis in R. In the next chapter, we will focus on **importing external data and preparing it for analysis**.

7 Data Import in R (CSV, Excel, SPSS, SAS)

7.1 Introduction

In real-world data analysis, data rarely originates inside R. Instead, datasets are typically stored in external files such as spreadsheets, text files, or statistical software formats. This chapter introduces the most common methods for **importing data into R**, with a focus on formats widely used in statistics, research, and industry: **CSV, Excel, SPSS, and SAS**.

The ability to correctly import data is a critical skill, as errors at this stage can propagate throughout the analysis.

7.2 Working Directory and File Paths

Before importing data, it is important to understand where R looks for files.

```
getwd()
```

The working directory is the default location from which R reads files and saves outputs. Using **RStudio Projects** is strongly recommended to manage file paths consistently.

Files can be referenced using:

- **Relative paths** (recommended)
- **Absolute paths** (system-specific, less portable)

7.3 Importing CSV Files

Comma-Separated Values (CSV) files are one of the most common data formats.

7.3.1 Using Base R

```r
data_csv <- read.csv("data/sample.csv")
head(data_csv)
```

Important arguments:

- `header = TRUE` – first row contains column names
- `stringsAsFactors = FALSE` – prevents automatic factor conversion

```r
data_csv <- read.csv("data/sample.csv", stringsAsFactors = FALSE)
```

7.3.2 Using `readr` Package (Recommended)

```r
library(readr)
data_csv <- read_csv("data/sample.csv")
```

Advantages of `readr`:

- Faster for large datasets
- Better handling of column types
- Clearer warnings and messages

7.4 Importing Excel Files

Excel files are widely used in academic and administrative settings.

7.4.1 Using `readxl`

```r
library(readxl)
data_excel <- read_excel("data/sample.xlsx")
```

To read a specific sheet:

```r
data_excel <- read_excel("data/sample.xlsx", sheet = "Sheet1")
```

To view available sheets:

```
excel_sheets("data/sample.xlsx")
```

Excel files do not require external dependencies, making `readxl` suitable for most systems.

7.5 Importing SPSS Files

SPSS files (`.sav`) are commonly used in social sciences and health research.

7.5.1 Using `haven`

```
library(haven)
data_spss <- read_sav("data/sample.sav")
```

SPSS value labels are preserved and imported as labelled variables.

```
str(data_spss)
```

To convert labelled variables to factors:

```
data_spss[] <- lapply(data_spss, as_factor)
```

7.6 Importing SAS Files

SAS datasets are widely used in clinical research and regulatory environments.

7.6.1 Using `haven`

```
data_sas <- read_sas("data/sample.sas7bdat")
```

SAS transport files (`.xpt`) can also be imported:

```
data_sas_xpt <- read_xpt("data/sample.xpt")
```

Metadata such as variable labels and formats are preserved during import.

7.7 Checking Imported Data

After importing data, it is essential to inspect its structure and content.

```
str(data_csv)
summary(data_csv)
head(data_csv)
```

Key checks include:

- Correct variable types
- Missing values
- Unexpected factor levels
- Column names and labels

7.8 Common Data Import Issues

Some common problems encountered during data import include:

- Incorrect file paths
- Encoding issues
- Automatic type conversion
- Missing or malformed values

Careful inspection and cleaning immediately after import is considered best practice.

7.9 Saving Imported Data in R Format

Once data is imported and cleaned, it can be saved in R's native formats.

```
save(data_csv, file = "data/sample.RData")
saveRDS(data_csv, "data/sample.rds")
```

R-specific formats load faster and preserve object structure.

7.10 Summary

In this chapter, you learned how to:

- Set and manage file paths
- Import CSV and Excel files
- Read SPSS and SAS datasets using `haven`
- Inspect and validate imported data
- Save datasets in R-native formats

Accurate data import is the foundation of reliable analysis. The next chapter will focus on **data cleaning and transformation**, preparing imported datasets for statistical analysis.

8 Data Manipulation with dplyr

8.1 Introduction

Once data has been imported into R, the next crucial step is **data manipulation**. Real-world datasets are rarely analysis-ready. They often contain unnecessary variables, missing values, inconsistent coding, or require transformation before statistical analysis can be performed.

The `dplyr` package provides a powerful, consistent, and readable set of tools for data manipulation. It is part of the **tidyverse**, a collection of R packages designed to make data science tasks intuitive and efficient. This chapter focuses on understanding the philosophy of `dplyr`, its core verbs, and their practical application through extensive examples.

8.2 Why Use dplyr?

Traditional base R functions are powerful but can become complex and difficult to read when performing multiple operations. `dplyr` improves clarity by:

- Using **simple, consistent verbs**
- Allowing operations to be expressed as a sequence of steps
- Producing readable and maintainable code
- Integrating seamlessly with data frames and tibbles

The guiding idea behind `dplyr` is:

> *Each function performs one clear data manipulation task.*

8.3 Installing and Loading dplyr

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

Most modern R installations already include `dplyr` as part of the tidyverse.

8.4 The Pipe Operator %>%

One of the defining features of `dplyr` is the **pipe operator %>%**. It allows the output of one function to be passed directly as the input to the next.

8.4.1 Without Pipe

```
summary(head(data))
```

8.4.2 With Pipe

```
data %>%
  head() %>%
  summary()
```

The pipe makes code easier to read by expressing operations **from left to right**, similar to how we describe analysis steps in words.

8.5 Selecting Variables with `select()`

The `select()` function is used to choose specific columns from a data frame.

```
data %>%
  select(Name, Age, Salary)
```

8.5.1 Selecting by Position

```
data %>%
  select(1:3)
```

8.5.2 Excluding Variables

```
data %>%
  select(-Salary)
```

8.5.3 Helper Functions

```
data %>%
  select(starts_with("Age"))
```

Common helpers include `starts_with()`, `ends_with()`, `contains()`, and `matches()`.

8.6 Filtering Observations with `filter()`

The `filter()` function extracts rows that satisfy logical conditions.

```
data %>%
  filter(Age > 30)
```

8.6.1 Multiple Conditions

```
data %>%
  filter(Age > 30 & Gender == "Male")
```

8.6.2 Using Logical OR

```
data %>%
  filter(Department == "HR" | Department == "Finance")
```

8.6.3 Handling Missing Values

```
data %>%
  filter(!is.na(Salary))
```

8.7 Creating and Transforming Variables with `mutate()`

The `mutate()` function creates new variables or modifies existing ones.

```
data %>%
  mutate(Bonus = Salary * 0.10)
```

8.7.1 Multiple Transformations

```
data %>%
  mutate(
    Bonus = Salary * 0.10,
    AnnualSalary = Salary * 12
  )
```

8.7.2 Conditional Variables with `ifelse()`

```
data %>%
  mutate(Performance = ifelse(Salary > 50000, "High", "Average"))
```

8.8 Arranging Rows with `arrange()`

The `arrange()` function sorts data.

```
data %>%
  arrange(Salary)
```

8.8.1 Descending Order

```
data %>%
  arrange(desc(Salary))
```

8.9 Summarising Data with `summarise()`

The `summarise()` function reduces multiple values into summary statistics.

```
data %>%
  summarise(
    AverageSalary = mean(Salary, na.rm = TRUE),
    MaxSalary = max(Salary, na.rm = TRUE)
  )
```

8.10 Grouped Operations with group_by()

The true power of dplyr lies in grouped analysis.

```
data %>%
  group_by(Department) %>%
  summarise(
    AvgSalary = mean(Salary, na.rm = TRUE),
    Count = n()
  )
```

Grouped operations are fundamental in statistical reporting and exploratory analysis.

8.11 Counting Observations

```
data %>%
  count(Gender)
```

Equivalent to grouping and counting in one step.

8.12 Renaming Variables with rename()

```
data %>%
  rename(
    MonthlySalary = Salary,
    EmployeeAge = Age
  )
```

8.13 Removing Duplicate Rows with `distinct()`

```
data %>%
  distinct(EmployeeID, .keep_all = TRUE)
```

8.14 Combining Multiple dplyr Verbs

A typical real-world workflow:

```
data %>%
  filter(!is.na(Salary)) %>%
  mutate(AnnualSalary = Salary * 12) %>%
  group_by(Department) %>%
  summarise(
    AvgAnnualSalary = mean(AnnualSalary),
    Employees = n()
  ) %>%
  arrange(desc(AvgAnnualSalary))
```

This example demonstrates how complex analysis can be expressed in a clear, step-by-step pipeline.

8.15 Working with Missing Values

```
data %>%
  summarise(MissingSalary = sum(is.na(Salary)))
```

Handling missing data explicitly is a critical step before any statistical modeling.

8.16 Converting to Tibble

```
data_tbl <- as_tibble(data)
```

Tibbles provide improved printing and stricter behavior compared to base data frames.

8.17 Best Practices for Data Manipulation

- Always inspect data before and after manipulation
- Avoid overwriting original datasets
- Use clear variable names
- Break complex pipelines into readable steps
- Comment your code when logic is not obvious

8.18 Summary

This chapter introduced the principles and practical usage of `dplyr` for data manipulation. You learned how to:

- Select and filter data
- Create and transform variables
- Summarise and group data
- Combine multiple operations using pipes

Mastering `dplyr` is essential for efficient data analysis in R. The next chapter will focus on **data visualization using `ggplot2`**, where manipulated data is transformed into meaningful graphical representations.