Applied Statistics

R Sinuvasan

VIT-AP UNIVERSITY

Department of Mathematics

2024

Applied Statistics

R Sinuvasan

Department of Mathematics

2024

Preface

In today's data-driven world, the ability to analyze and interpret data is more critical than ever. Statistics provides the foundation for understanding data, while programming tools like R offer the computational power to perform complex analyses efficiently. This manual combines essential statistical concepts with hands-on R programming, offering students a comprehensive guide to both understanding and applying statistical techniques.

This manual is designed to introduce students to fundamental statistical methods and their implementation in R, enabling them to solve real-world problems using a practical, computational approach.

The first section focuses on descriptive statistics, covering measures of central tendency such as mean, median, and mode. These tools help summarize and interpret data, providing initial insights into datasets. R functions will be introduced to calculate these measures, allowing students to immediately see how statistical concepts translate into code.

We then explore probability theory, including key concepts like random events, conditional probability, and Bayes' theorem. With R, students will simulate probabilistic scenarios and learn how to model uncertainty using code. Bayes' theorem, in particular, will be implemented to demonstrate how new information updates probabilities in real time.

Next, the manual covers random variables and probability distributions, both discrete and continuous. In this section, students will learn how to work with common distributions (e.g., binomial, normal, Poisson) and use R to visualize and analyze these distributions. Simulations in R will help students gain a deeper understanding of how random variables behave and how distributions are used in practice.

Understanding relationships between variables is critical in data analysis, so we explore correlation and regression analysis. Using R, students will compute correlation coefficients and build regression models to investigate

the strength and direction of relationships between variables. R's graphical capabilities will also be employed to create scatter plots, residual plots, and other visualizations, making the results more interpretable.

In the final part of the manual, we focus on decision-making through hypothesis testing. Various tests, including the z-test, t-test, F-test, and chi-square test, will be covered. Each test will be demonstrated using R, guiding students through hypothesis formulation, execution of the tests, and interpretation of results. This hands-on approach ensures that students learn not only the theory but also the practical skills necessary for conducting statistical tests using real data.

Throughout this manual, R programming is integrated seamlessly with the statistical topics covered. By the end of the course, students will have a solid understanding of both statistical theory and how to use R to implement these techniques effectively. The focus on R not only equips students with a valuable skill for the job market but also enables them to work on data-driven projects with confidence.

Each chapter contains code snippets, exercises, and case studies to reinforce learning, ensuring that students can apply their knowledge to solve real-world problems. We hope this manual serves as a valuable resource, enabling students to harness the power of statistics and R programming in their academic and professional endeavors.

This version of the preface highlights how R programming is used throughout the manual to teach statistical concepts. It emphasizes the practical, hands-on approach that students will gain by combining theory with code, preparing them for real-world data analysis tasks. R Sinuvasan

October 2024

Contents

1 Introduction

In an age dominated by data, the ability to analyze and interpret information has become a critical skill across various fields, including business, healthcare, social sciences, and engineering. Statistics provides the essential framework for transforming raw data into meaningful insights, enabling informed decision-making. As we navigate through complex datasets, the integration of statistical theory with programming skills has proven invaluable. Among the tools available, **R** stands out as a powerful programming language specifically designed for statistical computing and data analysis.

R is an open-source programming language that has gained immense popularity among statisticians and data scientists for its robust capabilities and extensive ecosystem. It is renowned for its ability to handle complex statistical analyses, create compelling visualizations, and manage data effectively. With a vast array of packages available—such as `dplyr` for data manipulation, `ggplot2` for data visualization, and `tidyverse` for streamlined data science workflows—R empowers users to perform sophisticated statistical analyses with relative ease. Its flexibility and adaptability make it an essential tool for anyone working with data.

This manual is structured to provide a comprehensive overview of fundamental statistical concepts while employing R programming throughout. We will explore a variety of topics that are crucial for statistical analysis, including:

- **Descriptive Statistics**: We begin by examining measures of central tendency—mean, median, and mode. These statistics summarize data sets, allowing us to draw initial insights about distributions. Through R, students will learn how to calculate and visualize these measures, gaining a deeper understanding of their significance.

- **Probability Theory**: Understanding probability is essential for mak-

ing predictions based on data. This section introduces key concepts, including basic probability rules, conditional probability, and Bayes' theorem. By using R to simulate probabilistic scenarios, students will learn to model uncertainty and apply Bayes' theorem to real-world situations, enhancing their analytical skills.

- **Random Variables and Distributions**: We will delve into random variables and their associated probability distributions, exploring both discrete and continuous cases. This section will cover important distributions such as the binomial, normal, and Poisson distributions. R will be utilized to visualize these distributions and understand their applications in statistical analysis.

- **Correlation and Regression**: Understanding relationships between variables is crucial in data analysis. We will explore correlation coefficients and regression analysis, teaching students how to quantify relationships between variables. Using R, students will create regression models and visualizations, allowing them to make predictions based on their findings.

- **Hypothesis Testing and Decision Making**: The final section addresses hypothesis testing—a fundamental aspect of statistical analysis. Students will learn how to formulate and test hypotheses using z-tests, t-tests, F-tests, and chi-square tests. R will serve as a powerful tool for conducting these tests and interpreting the results, providing students with the skills to make informed decisions based on data.

By integrating R programming throughout the manual, we ensure that students not only grasp statistical theories but also develop practical coding skills essential for data analysis. Each chapter includes coding examples, exercises, and case studies that encourage students to apply their knowledge in real-world contexts.

The use of R in this manual is supported by a wealth of literature that showcases its capabilities and applications in statistics. For instance, *R for Data Science* by Hadley Wickham and Garrett Grolemund provides an excellent introduction to data science concepts using R, guiding readers through practical data analysis techniques [**wickham2017r**]. Furthermore, the R community continuously contributes to a growing repository of packages and

resources, ensuring that users have access to the latest developments in statistical methodologies.

By the end of this manual, students will have a solid foundation in both statistical concepts and R programming, equipping them with the skills necessary to tackle diverse data-driven problems. We hope this resource inspires a deeper appreciation for the power of statistics and the role of R in unlocking insights from data.

2 Introduction to R Programming

2.1 Introduction to R

R is a programming language and environment primarily used for statistical computing and data analysis. It was developed by statisticians Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and has gained popularity among data analysts and researchers in various fields, including social sciences, bioinformatics, and finance.

R provides a wide range of statistical techniques and is highly extensible, allowing users to create custom functions and packages. As an open-source language, R has a large and active community that continuously contributes to its development and improvement. This community-driven approach has resulted in a vast ecosystem of packages available through the Comprehensive R Archive Network (CRAN), which enhances R's capabilities for various data analysis tasks.

The language is particularly known for its powerful data visualization tools, which enable users to create high-quality graphs and plots. R's syntax and data structures are designed to facilitate data manipulation, making it a preferred choice for exploratory data analysis.

R is also widely used for reproducible research, enabling analysts to produce dynamic reports that combine code, output, and narrative text, thus making the research process transparent and verifiable.

Overall, R has become an essential tool for data scientists and statisticians, offering robust solutions for data analysis and visualization.

- History and development of R.: R was developed in the early 1990s by statisticians Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. It was designed as a free, open-source programming language for statistical computing and data analysis, in-

spired by the S programming language created by John Chambers and his colleagues at Bell Laboratories.

The first version of R was released in 1995, and it quickly gained traction within the statistical community. Its development was fueled by the growing need for powerful statistical tools that were accessible to researchers and analysts. In 2000, the R Project was formally established, and R became widely recognized as a leading language for statistical analysis and data visualization.

Over the years, R has evolved significantly, with contributions from a vibrant global community of developers and users. The Comprehensive R Archive Network (CRAN) was established as a repository for R packages, allowing users to share their work and access a vast array of additional functionalities.

Today, R is supported by a comprehensive ecosystem, including numerous packages for various statistical techniques, data manipulation, and visualization. Its adoption continues to grow in academia, industry, and government, making it one of the most popular tools for data analysis and research.

- Advantages of using R for statistics and data analysis.: R offers several advantages for statisticians and data analysts:
  - **Open Source:** R is free to use and distribute, which makes it accessible to a wide range of users and encourages community contributions.
  - **Comprehensive Statistical Packages:** R has a vast collection of packages available through CRAN, providing tools for a wide array of statistical analyses, from basic to advanced.
  - **Data Visualization:** R excels in data visualization, with packages like `ggplot2` that allow users to create high-quality, customizable graphics.
  - **Active Community:** A large and active community contributes to R's development, providing support, resources, and continuously updated packages.
  - **Reproducible Research:** R supports reproducible research through R Markdown, allowing users to combine code, analysis, and documentation in a single document.

  - **Integration with Other Tools:** R can easily integrate with other programming languages (e.g., Python, C++), databases, and tools, enhancing its capabilities for data analysis.
  - **User-Friendly Syntax:** The syntax of R is designed for data analysis, making it intuitive for statisticians and researchers, even those with limited programming experience.

- Overview of R's ecosystem, including RStudio and CRAN.: R's ecosystem is rich and diverse, facilitating a wide range of statistical computing and data analysis tasks. Key components include:

  - **R:** The core programming language, designed for statistical analysis and data manipulation, with a wide array of built-in functions and libraries.
  - **CRAN (Comprehensive R Archive Network):** A repository of R packages that extends R's capabilities. It hosts thousands of packages for various statistical methods, data visualization, and machine learning.
  - **RStudio:** An integrated development environment (IDE) that enhances the R programming experience. It offers features like syntax highlighting, debugging tools, and project management, making it easier for users to write and organize their R code.
  - **R Markdown:** A tool for creating dynamic reports that combine R code with narrative text, enabling the presentation of data analysis results in an easily shareable format.
  - **Bioconductor:** A project that provides tools for the analysis of genomic data, offering specialized packages for bioinformatics.
  - **Shiny:** A web application framework for R that allows users to create interactive web applications directly from R, making data analysis accessible to non-programmers.

  This ecosystem collectively empowers users to perform complex data analyses, create reproducible research, and visualize results effectively.

2.2 Installing R and RStudio

To get started with R, you first need to install R and RStudio.

- Step-by-step instructions for installing R and RStudio.
- Overview of R packages and how to install them using `install.packages()`.
- Basic configuration tips for RStudio.

2.3 Installing R

2.3.1 Windows

To install R on Windows:

- Visit the CRAN Windows page.
- Click on the download link for the latest version of R.
- Run the installer and follow the on-screen instructions, leaving most options at their default values unless you have specific needs.

2.3.2 Mac OS X

For Mac users:

- Go to the CRAN Mac OS page.
- Download the most recent version of the `.pkg` file (e.g., `R-4.3.0.pkg`).
- Open the file and follow the installer instructions to complete the installation.

2.3.3 Linux

On Linux-based systems, R can be installed using the terminal:

- Open a terminal window.
- For Debian-based distributions (e.g., Ubuntu), use the command:

```
sudo apt-get install r-base
```

- For Fedora-based distributions, use:

```
sudo dnf install R
```

- Ensure that any dependencies required by R are installed.

2.4 Installing RStudio

Once R is installed, you can install RStudio:

- Visit the RStudio download page.
- Choose your operating system (Windows, Mac, or Linux) and download the corresponding installer.
- Run the installer and follow the on-screen instructions to set up RStudio.

2.4.1 Windows and Mac

For both Windows and Mac, the installation is straightforward: download the executable file, open it, and proceed with the default options unless specific configurations are needed.

2.4.2 Linux

On Linux, after downloading the `.deb` or `.rpm` file, use the terminal to install RStudio:

- For Debian-based systems, run:

```
sudo dpkg -i rstudio-x.yy.zzz-amd64.deb
```

- For Fedora-based systems, run:

```
sudo dnf install rstudio-x.yy.zzz-x86_64.rpm
```

- Replace `x.yy.zzz` with the actual version number of RStudio you downloaded.

2.5 Overview of R Packages

R packages are collections of R functions, data, and compiled code that are stored in a well-defined format. They extend R's functionality by providing tools for specific tasks, such as data manipulation, visualization, or statistical analysis. Many packages are available through CRAN (Comprehensive R Archive Network), but others are hosted on platforms like GitHub.

2.6 Installing R Packages

To install an R package, use the `install.packages()` function. This function downloads and installs the specified package from CRAN or other repositories.

2.6.1 Example of Installing a Package

For example, to install the `ggplot2` package for data visualization, run the following command in your R console:

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

This command will download and install the package, including any dependencies, automatically.

2.6.2 Loading an Installed Package

Once a package is installed, load it into your R session using the `library()` function:

```
library(ggplot2)
```

2.7 Commonly Used Packages

Some widely-used packages in R include:

- `dplyr` – for data manipulation
- `ggplot2` – for data visualization
- `tidyverse` – a collection of data science tools
- `shiny` – for building interactive web applications

2.8 Basic Configuration Tips for RStudio

- **Appearance:** Go to `Tools > Global Options > Appearance` to change the editor theme and font size for better visibility.
- **Pane Layout:** Customize the layout of RStudio panes (Source, Console, Environment) under `Tools > Global Options > Pane Layout`.

2.9 Code Execution

- Enable line numbers, auto-complete, and code folding under `Tools > Global Options > Code`.
- Set keyboard shortcuts for running code chunks (e.g., `Ctrl + Enter`).

2.10 Packages

- Manage installed packages via the `Packages` tab or use `install.packages()` to add new packages.

2.11 Basic R Syntax

Understanding the basic syntax of R is crucial for efficient programming.

- Variables, data types (numeric, character, factor, logical).

- Basic operators (arithmetic, relational, logical).
- Introduction to vectors and data frames.

2.12 Variables in R

Variables in R are used to store data that can be manipulated during the execution of a program. A variable can be assigned a value using the assignment operator `<-` or `=`. For example:

```
x <- 10
name <- "John"
```

In R, variables do not need explicit declaration, and their type is determined dynamically when a value is assigned.

2.13 Data Types in R

R provides several basic data types that are essential for handling different kinds of data. These include numeric, character, factor, and logical types.

2.13.1 Numeric

The numeric data type is used to store numbers, both integers and real numbers (decimals). For example:

```
x <- 42
y <- 3.14
```

R treats all numbers as double precision by default, but integers can also be explicitly declared using the `as.integer()` function.

2.13.2 Character

Character data represents text or strings. Characters are enclosed in double or single quotes. For example:

```
name <- "Alice"
greeting <- 'Hello'
```

2.13.3 Factor

Factors are used to handle categorical data in R. They store both the values and the possible categories. Factors are particularly useful for statistical modeling:

```
colors <- factor(c("red", "blue", "green", "red"))
```

Factors can be ordered or unordered and are often used in data frames for analysis.

2.13.4 Logical

Logical data types store boolean values: `TRUE` or `FALSE`. They are often the result of comparisons or logical operations:

```
is_sunny <- TRUE
result <- 5 > 3  # result will be TRUE
```

Understanding variables and data types is crucial for working effectively with R. The dynamic nature of R allows for flexible handling of different types of data, making it easier to perform statistical analysis and data manipulation.

2.14 Introduction

In R, operators allow you to perform various computations on data. These include arithmetic operations for numerical calculations, relational operators for comparisons, and logical operators for evaluating boolean expressions. Understanding these operators is fundamental for programming in R.

2.15 Arithmetic Operators

Arithmetic operators are used for basic mathematical computations. The main arithmetic operators in R are:

- `+` : Addition
- `-` : Subtraction

- `*` : Multiplication
- `/` : Division
- `^` : Exponentiation
- `%/%` : Integer division (quotient)
- `%%` : Modulo (remainder)

2.15.1 Examples

```
x <- 10
y <- 3

sum <- x + y        # sum = 13
diff <- x - y       # diff = 7
prod <- x * y       # prod = 30
quot <- x / y       # quot = 3.33
exp <- x ^ y        # exp = 1000
int_div <- x %/% y  # int_div = 3
mod <- x %% y       # mod = 1
```

2.16 Relational Operators

Relational operators compare values and return `TRUE` or `FALSE`. The main relational operators are:

- `==` : Equal to
- `!=` : Not equal to
- `>` : Greater than
- `<` : Less than
- `>=` : Greater than or equal to
- `<=` : Less than or equal to

2.16.1 Examples

```
x <- 5
y <- 10

result1 <- x == y    # result1 = FALSE
result2 <- x != y    # result2 = TRUE
result3 <- x > y     # result3 = FALSE
result4 <- x <= y    # result4 = TRUE
```

Relational operators are often used in conditions for `if` statements, loops, and other control structures.

2.17 Logical Operators

Logical operators evaluate logical expressions and are primarily used to combine relational operations. The main logical operators are:

- `&` : Logical AND
- `|` : Logical OR
- `!` : Logical NOT

2.17.1 Examples

```
a <- TRUE
b <- FALSE

result1 <- a & b     # result1 = FALSE (both must be TRUE)
result2 <- a | b     # result2 = TRUE (one must be TRUE)
result3 <- !a        # result3 = FALSE (logical negation)
```

Logical operators are especially useful when you need to check multiple conditions.

Arithmetic, relational, and logical operators are crucial in R for data manipulation, decision-making, and computations. They provide a foundation for writing effective R scripts and performing complex analyses.

2.18 Vectors in R

A vector is one of the most basic data structures in R. It is a sequence of data elements of the same type, such as numeric, character, or logical. Vectors can be created using the `c()` function, which combines values into a single vector.

2.18.1 Example of a Numeric Vector

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

Vectors in R are 1-dimensional and are fundamental to most data manipulation operations.

2.18.2 Example of a Character Vector

```
names <- c("Alice", "Bob", "Charlie")
```

In this chapter, we will cover vectors. A vector is a data structure in R. A data structure can be defined as a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. A vector can either be an atomic vector or a list. An atomic vector contains only one data type, whereas a list may contain multiple data types.

An atomic vector could be Character, Logical, Double, Integer, or Complex.

2.18.3 Integer

```
4
## [1] 4
```

2.18.4 Character

```
"Pune"
## [1] "Pune"
```

You can use the `typeof` function to check the type of each of these:

```
typeof("Pune")
## [1] "character"
typeof(4)
## [1] "double"
typeof(4.3)
## [1] "double"
typeof(2+1i)
## [1] "complex"
```

2.18.5 Creating a Vector

Let's create a vector containing three numeric values. This will be an example of a Numeric Vector:

```
c(1, 4, 7)
## [1] 1 4 7
```

To check the length of the vector, we can use the length function:

```
length(c(1, 4, 7))
## [1] 3
```

Let's assign this to a variable for future use:

```
Num_variable <- c(1, 4, 7)
```

Let's create a vector `Names` that contains the names of persons appearing in an Exam. This will be a kind of character vector. Please note that here `Names` is a vector, and `<-` is an operator that assigns the value on the right-hand side of the operator to the variable name on the left-hand side. The character `c` is used to create a vector:

```
Names <- c("Arvind", "Krishna", "Rahul", "Saurabh", "Venkat", "Sucharitha")
```

Let's extract the first element of the `Names` vector. To extract the first element, we need to use `[` and the value 1:

```
Names[1]
## [1] "Arvind"
```

Similarly, to extract the third value from the `Names` vector:

```
Names[3]
## [1] "Rahul"
```

Let's create another vector and see if we can combine two vectors:

```
Missed_names <- c("Ajay", "Amit")
```

Here we will be updating the `Names` vector by including `Missed_names` in the `Names` vector:

```
Names <- c(Names, Missed_names)
Names
## [1] "Arvind"     "Krishna"    "Rahul"      "Saurabh"    "Venkat"
## [6] "Sucharitha" "Ajay"       "Amit"
```

2.18.6 Class of Vector

We can also check the types of vectors using different functions such as:

```
class(Names)
## [1] "character"
class(Num_variable)
## [1] "numeric"
typeof(Num_variable)
## [1] "double"
```

What is the difference between `typeof`, `mode`, and `class`? I will leave it to the reader to explore it on their own. Remember the reader can use the `help` function to check the definitions in detail. For example, just type `?class` in the console to see the details of the class function.

2.18.7 Adding Vectors of Different Types

Let's add a character and numeric vector and check the result:

```
mix_vector <- c(1, 2, "Amish")
class(mix_vector)
## [1] "character"
```

The `mix_vector` we created had multiple data types (Numeric and Character); however, R converts the multiple data types to a single data type through a process called coercion. Logical values are converted to numbers: TRUE is converted to 1 and FALSE to 0.

Values are converted to the simplest type required to represent all information.

The ordering is roughly logical < integer < numeric < complex < character < list.

Objects of type `raw` are not converted to other types.

Objects can also be explicitly coerced using the `as.` function. For example, to coerce the `mix_vector` to numeric use:

```
mix_vector <- as.numeric(mix_vector)
## Warning: NAs introduced by coercion
mix_vector
## [1]  1  2 NA
class(mix_vector)
## [1] "numeric"
```

We can also change the elements of the vector. Let's change the first element of the `Names` vector using the following expression:

```
Names[1] <- "Arun"
Names
## [1] "Arun"       "Krishna"    "Rahul"      "Saurabh"    "Venkat"
## [6] "Sucharitha" "Ajay"       "Amit"
```

2.18.8 Accessing Elements of Vectors

Let's try to access elements of the vector using negative indexing:

```
Names[-1] # All elements except the first element will be printed
## [1] "Krishna"    "Rahul"      "Saurabh"    "Venkat"     "Sucharitha"
## [6] "Ajay"       "Amit"
Names[c(-1, -2)] # All elements except the first and second elements will be printed
## [1] "Rahul"      "Saurabh"    "Venkat"     "Sucharitha" "Ajay"
## [6] "Amit"
```

Alternatively, we can also use the following code to remove the first two elements of the `Names` vector:

```
Names[-c(1, 2)] # All elements except the first and second elements will be pr
## [1] "Rahul"      "Saurabh"    "Venkat"     "Sucharitha" "Ajay"
## [6] "Amit"
```

2.18.9 Creating Vector Using In-Built Functions

```
seq(1, 10, by = 1) # Create a sequential vector from 1 to 10 which increases b
##  [1]  1  2  3  4  5  6  7  8  9 10
seq(1, 10, by = 0.5)
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0
```

2.18.10 Let's Try to Repeat Vectors

```
rep(c(1, 2), times = c(5, 5))
##  [1] 1 1 1 1 1 2 2 2 2 2
```

`typeof(Vector)`: This method tells you the data type of the vector.

`Sort(Vector)`: This method helps us to sort the items in ascending order.

`length(Vector)`: This method counts the number of elements in a vector.

`head(Vector, limit)`: This method returns the top six elements (if you omit the limit). If you specify the limit as 4, then it returns the first 4 elements.

`tail(Vector, limit)`: It returns the last six elements (if you omit the limit). If you specify the limit as 2, then it returns the last two elements.

2.18.11 Arithmetic Operators in Vectors in R

We have performed arithmetic operations in previous chapters. Let's see how we can perform arithmetic operations in vectors.

```
V1 <- c(1, 2, 3, 4) # created a vector V1
V2 <- c(5, 6, 7, 8) # created a vector V2
```

Let's perform addition:

```
V12 <- V1 + V2
V12 # Print result of V12, which is sum of V1 and V2 vector.
## [1]  6  8 10 12
```

As you may have guessed, the addition operation above was performed elementwise. That means that the first element of `V1` was added to the first element of `V2`.

Let's perform multiplication:

```
V12_mult <- V1 * V2
V12_mult
## [1]  5 12 21 32
```

Let's perform division:

```
V12_division <- V2 / V1
V12_division
## [1] 5.000000 3.000000 2.333333 2.000000
typeof(V12_division)
## [1] "double"
```

Let's convert the double to numeric:

```
V12_division <- as.integer(V12_division)
V12_division
## [1] 5 3 2 2
typeof(V12_division)
## [1] "integer"
```

2.19 Introduction to Data Frames in R

Data frames in R language are a type of data structure used to store data in a tabular form, which is two-dimensional. The data frames are a special category of list data structures in which the components are of equal length. R language supports the built-in function `data.frame()` to create data frames and assign data elements. R also supports using the data frame name to modify and retrieve data elements from data frames. Data frames in R are structured with column names as component names, and rows structured by component values. Data frames in R are a widely used data structure when developing machine learning models in data science projects.

2.19.1 Characteristics of Data Frames

- The column name is required.
- Row names should be unique.
- The number of items in each column should be the same.

2.20 Steps For Creating Data Frames in R

Let's start with creating a data frame, which is explained below:

2.20.1 Step 1: Create a Data Frame of a Class in a School

Code:

```
tenthclass = data.frame(roll_number = c(1:5),
Name = c("John","Sam","Casey","Ronald","Mathew"),
Marks = c(77,87,45,68,95), stringsAsFactors = FALSE)
print(tenthclass)
```

When we run this code, we will get a data frame like this:
Output:

```
  roll_number   Name Marks
1           1   John    77
2           2    Sam    87
3           3  Casey    45
4           4 Ronald    68
5           5 Mathew    95
```

Figure 2.1: Data frame output example

Here, in our example, the data frame is very small, but in real life, while dealing with problems, we have lots of data. To understand the structure of the data, we use the `Str()` function.

2.20.2 Step 2: Add the following line to our code

Code:

```
Str(tenthclass)
```

When we run the whole code, we will get the following output:

Output: `Str(tenthclass)`

The above output means we have 5 observations of 3 variables. It also explains the data type of each variable. For example, the roll number is an integer, the name is a character, and Marks are numeric.

Once we understand the structure of the data, we can pass the following code to understand the data more statistically.

2.20.3 Step 3: Use the `summary()` Function

Code:

```
summary(tenthclass)
```

Output: Data Frames in R 1-3

The `summary()` function provides a better understanding of our data. It tells us the mean, median, quartiles, maximum, and minimum values, helping us make better decisions.

2.21 Matrices in R

Matrices are two-dimensional data structures in R and are arranged in a rectangular layout. Matrices can contain only one data type. We can create matrices of any of the six data types we discussed before. A matrix can also be thought of as a vector in two dimensions.

We can use the `matrix` function to create a matrix in R programming. I will suggest using `? matrix` for detailed documentation of the matrix function. Let's create a matrix with 3 rows and 2 columns:

```
matrix(1:6, nrow = 3, ncol = 2)
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

What if we don't specify the number of rows and columns?

```
matrix(1:6, 3, 2)
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

As you have guessed from the above output, R will automatically pick the number of rows and columns based on the order of input. Let's specify one of the dimensions (either column or row) and see the output:

```
matrix(1:6, ncol = 3)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

So far we have not assigned any column names and row names to the matrix. Let's create a matrix M and assign column names and row names:

```
M <- matrix(1:20, ncol = 4)
colnames(M) <- c("A", "B", "C", "D")
rownames(M) <- c("E", "F", "G", "H", "I")
```

Let's check the matrix M:

```
M
##   A  B  C  D
## E 1  6 11 16
## F 2  7 12 17
## G 3  8 13 18
## H 4  9 14 19
## I 5 10 15 20
```

As you can see from the output, we have successfully assigned names to the columns. We can also call specific columns now. For example, `Matrix[rowname, colname]` will call the required row and column by name:

```
M["F", "D"]
## [1] 17
```

We can leave the row name field blank in `Matrix[rowname, colname]` to call the complete column:

```
M[, "D"]
##  E  F  G  H  I
## 16 17 18 19 20
```

As you can see in the above example, we have kept the row blank. This will cause the above expression to pick all the values from the corresponding columns. We can also access specific elements of vectors by specifying the row number and column number. Let's say we want to pick a specific column and vector from matrix, say the 2nd row and 3rd column:

```
M[2, 3]
## [1] 12
```

2.21.1 Extracting Row/Column from Matrices

We can also extract more than one row or column at a time. For example, below we extracted the first and third column and the second row:

```
M[2, c(1, 3)]
##  A  C
##  2 12
```

So what else can we do with matrices?

2.21.2 Operations in Matrices

Let's perform some operations with matrices. Let's create two matrices M_1 and M_2 and perform some operations:

```
M1 <- matrix(1:15, nrow = 5)
M1
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
```

```
## [5,]    5   10   15
```

```
M2 <- matrix(1:15, nrow = 5)
M2
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

Arithmetic Operations in Matrices

What if we add two matrices? Matrix operations can be performed similarly to how arithmetic operations are performed on numbers. However, we should be careful with the dimensions of the matrices that we are working on.

Let's create three matrices M_1, M_2, M_3:

```
M1 <- matrix(1:15, nrow = 5)
M1
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
M2 <- matrix(2:16, nrow = 5)
M2
##      [,1] [,2] [,3]
## [1,]    2    7   12
## [2,]    3    8   13
## [3,]    4    9   14
## [4,]    5   10   15
## [5,]    6   11   16
```

```
M3 <- matrix(2:10, nrow = 5)
## Warning in matrix(2:10, nrow = 5): data length [9] is not a sub-multiple or
```

```
## multiple of the number of rows [5]
M3
##      [,1] [,2]
## [1,]    2    7
## [2,]    3    8
## [3,]    4    9
## [4,]    5   10
## [5,]    6    2
```

Arithmetic Operations in Matrices

Addition operation on matrices of the same dimension:

```
M1 + M2
##      [,1] [,2] [,3]
## [1,]    3   13   23
## [2,]    5   15   25
## [3,]    7   17   27
## [4,]    9   19   29
## [5,]   11   21   31
```

What if we want to add matrices of different dimensions? We can check the dimensions of matrices using the `dim` function:

```
dim(M1)
## [1] 5 3
dim(M3)
## [1] 5 2
```

Multiplying matrices of the same dimension: The following operation using `"*"` is element-wise:

```
M1 * M2
##      [,1] [,2] [,3]
## [1,]    2   42  132
## [2,]    6   56  156
## [3,]   12   72  182
## [4,]   20   90  210
## [5,]   30  110  240
```

What if we want Matrix Multiplication? As an exercise, try `M1%*%M2` in the R console for matrix multiplication and check the result. Once you run the operation, you must have got an error message "Error in M1 %*% M2: non-conformable arguments". Why is it so? As a basic rule for matrix multiplication, the number of rows of Matrix1 should be equal to the number of columns in Matrix2 when multiplying Matrix1 with Matrix2. You can check the following link *mathisfun*.

```
M4 <- matrix(1:9, nrow=3)
M5 <- matrix(10:18, nrow=3)
M4
```

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

```
M5
```

```
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

Let's see the result of Matrix Multiplication:

```
M4 <- matrix(1:9, nrow=3)
M5 <- matrix(10:18, nrow=3)
M4 \%*\% M5
```

```
##      [,1] [,2] [,3]
## [1,]  138  174  210
## [2,]  171  216  261
## [3,]  204  258  312
```

You can see the elementwise operation result in R:

```
M4 * M5
```

```
##      [,1] [,2] [,3]
## [1,]   10   52  112
## [2,]   22   70  136
## [3,]   36   90  162
```

```
M1 / M2
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.5000000 0.8571429 0.9166667
## [2,] 0.6666667 0.8750000 0.9230769
## [3,] 0.7500000 0.8888889 0.9285714
## [4,] 0.8000000 0.9000000 0.9333333
## [5,] 0.8333333 0.9090909 0.9375000
```

What if one of the matrices' row or column is shorter than the other? Will recycling occur?

As an exercise, create a 3X3 matrix and add it to M1.

2.21.3 Transposing a Matrix

You may also want to transpose your data in R. This is used to interchange rows and columns, i.e., rows become columns and columns become rows in the new transposed matrix. For example:

```
M3 <- t(M1)
M3
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
```

As you can see from the output, the rows have become columns and the columns have become rows. This can also be used to reshape a DataFrame. We will cover DataFrame in the next chapter.

2.21.4 Common Matrix Operations in R

Sum of rows in Matrix:

```
rowSums(M1)
```

```
## [1] 18 21 24 27 30
```

Sum of columns in Matrix:

```
colSums(M1)
```

```
## [1] 15 40 65
```

Mean of rows in Matrix:

```
rowMeans(M1)
```

```
## [1]  6  7  8  9 10
```

2.21.5 Naming Matrix Rows and Columns

```
rownames(M4) <- c("A","B","C")
colnames(M4) <- c("D","E","F")
M4
```

```
##   D E F
## A 1 4 7
## B 2 5 8
## C 3 6 9
```

We can also extract specific elements of a Matrix by specifying rows and columns:

```
M4[3,] % # Extracting third row from Matrix M4
```

```
## D E F
## 3 6 9
```

```
M4[,3] % # Extracting third column from Matrix M4
```

```
## A B C
## 7 8 9
```

```
M4[2,3] % # Extracting second row and third column element from Matrix M4
```

```
## [1] 8
```

2.22 Functions in R

Introduction to functions: syntax and structure.

- Writing custom functions in R.
- Understanding scope and return values.
- Using built-in functions and applying functions like `apply()`, `lapply()`, etc.

2.23 Data Import and Export

R provides various functions for importing and exporting data.

- Reading data into R from various sources (CSV, Excel, databases).
- Writing data to files (CSV, Excel).
- Using packages like `readr`, `openxlsx`, and `DBI`.

2.24 Data Manipulation

Using the `dplyr` package allows for streamlined data manipulation.

- Key functions: `filter()`, `select()`, `mutate()`, `arrange()`, `summarize()`.
- Piping (`%>%`) and its advantages in data processing.

2.25 Data Visualization

Data visualization is vital for interpreting statistical results.

- Introduction to the `ggplot2` package.
- Creating various plots: scatter plots, histograms, boxplots, etc.
- Customizing plots (labels, colors, themes).

2.26 Programming with R

In addition to statistical functions, R allows for traditional programming constructs.

- Control structures: loops (`for`, `while`), conditionals (`if`, `else`).
- Writing functions and understanding scope.
- Error handling and debugging techniques.

2.27 Working with Packages

Packages enhance R's capabilities, providing additional functions.

- How to find and use R packages.
- Overview of essential packages for statistics (e.g., `MASS`, `caret`, `tidyverse`).
- How to check package documentation and help files.

2.28 R Markdown

R Markdown is a powerful tool for creating dynamic reports.

- Creating dynamic reports that combine code and output.
- Exporting reports to different formats (HTML, PDF, Word).

2.29 Basic Project Structure

Organizing R projects for better workflow is essential.

- Using RStudio projects for file management and version control.
- Best practices for coding in R.

2.30 Resources for Learning R

Numerous resources are available for those who wish to deepen their knowledge.

- Recommended books, online courses, and tutorials for further learning.
- Community resources and forums (e.g., R-bloggers, Stack Overflow).

2.31 Advanced Topics (Optional)

For advanced users, R offers powerful features for machine learning and large datasets.

- Introduction to advanced statistical methods in R (e.g., machine learning with `caret` or `mlr`).
- Working with large datasets using `data.table`.
- Building shiny applications for interactive data visualization.

A Appendix A: LaTeX Resources

Here are some useful resources to help you further explore LaTeX:

- LaTeX Project Official Website
- Overleaf LaTeX Documentation

B Appendix B: Common LaTeX Errors

When working with LaTeX, you may encounter common errors like missing braces or improper package usage. Refer to the following resources for troubleshooting:

- StackExchange: https://tex.stackexchange.com/
- LaTeX Wikibook: https://en.wikibooks.org/wiki/LaTeX