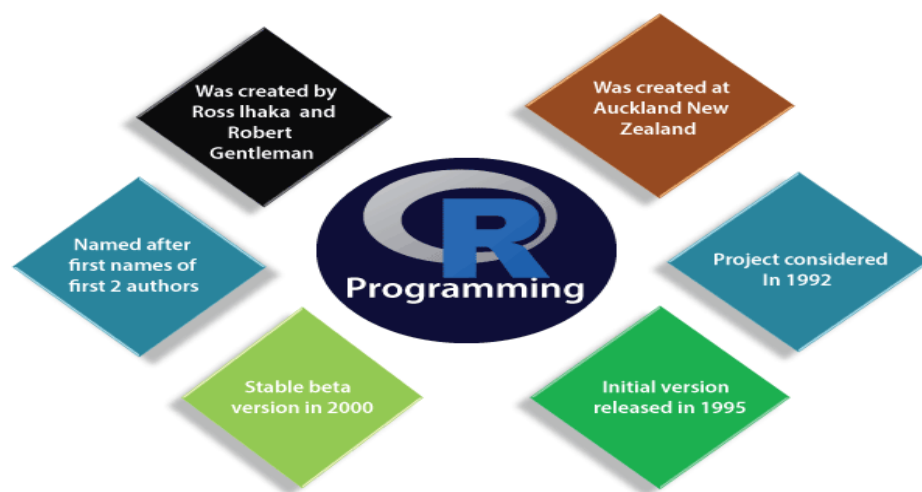

Manual

R PROGRAMMING LAB





Prepared BY

KVN VALLI

Assistant Professor

MVVS SUBRAHMANYAM

Assistant Professor

This lab manual is intended to aid second-year undergraduate Artificial Intelligence and Machine learning students in their course *R Programming lab* [B20CS2205].

About the author

K V N Valli : She is currently pursuing her (PhD) in Sathyabama University. She got her M.Tech. degree from Acharya Nagarjuna University Presently she is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

MVVS Subramanyam: He is currently pursuing his (PhD) in Andhra University. He got his M.Tech. degree from Sri Vasavi Engineering College, Tadepalligudem Presently he is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

For private circulation among 2/4 B.Tech.(AIML) students.

Preface

R PROGRAMMING LAB

This lab introduces R as a programming and mining tool. R is a freely downloadable language and environment for statistical computing and graphics. Its capabilities and the large set of available add-on packages make this tool an excellent alternative to many existing data mining tools.

R is a programming language created by statisticians for statistics, specifically for working with data. It is a language for statistical computing and data visualizations used widely by business analysts, data analysts, data scientists, and scientists.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering,) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

This manual is helpful to perform statistical operations by using R Language.. It provides an opportunity for the students to learn and apply R packages and functions for programming and statistical analysis . It introduces the fundamentals of R programming, standard R libraries, solid understanding of R functions, write programs using the R and gain skills in R programming.

Evaluation Scheme	
Examination	Marks
Exercise Programs	5
Record	5
Internal Exam	5
External Exam	35

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE(A)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

[B20 CS 2205]

R PROGRAMMING LAB **(Common For AIML and CSE)**

Course Objectives:

1.To In this course student will learn about the fundamentals of R programming, standard R libraries, solid understanding of R functions, write programs using R and gain skills in R programming language , get acquaintances with Arrays, Files, Strings, Packages and distributing using R

Course Outcomes:

At the end of the course Students will be able

1. Implement basic concept of R programming and its different modules that includes conditional, Looping ,lists ,strings ,functions ,frames ,arrays and file programming K3
2. Implement the concepts of R script to extract data from data frames and file operations. K3
3. Implement the various statistical techniques using R K3
4. Extend the functionality of R by using add-on packages. K3
5. Use R graphics and tables to visualize results of various statistical operations on data. K4

LIST OF PROGRAMS

Exercise - 1

- a. Installing R and RStudio
- b. Basic functionality of R, variable, data types in R

Exercise - 2

- a. Implement R script to show the usage of various operators available in R language.
- b. Implement R script to read person's age from keyboard and display whether he is eligible for voting or not.
- c. Implement R script to find biggest number between two numbers.
- d. Implement R script to check the given year is leap year or not.

Exercise - 3

- a. Implement R Script to create a list.
- b. Implement R Script to access elements in the list.
- c. Implement R Script to merge two or more lists. Implement R Script to perform matrix operation

Exercise - 4

Implement R script to perform following operations:

- a. various operations on vectors
- b. Finding the sum and average of given numbers using arrays.
- c. To display elements of list in reverse order.
- d. Finding the minimum and maximum elements in the array.

Exercise - 5

- a. various operations on vectors
- b. Finding the sum and average of given numbers using arrays.
- c. To display elements of list in reverse order.
- d. Finding the minimum and maximum elements in the array.

Exercise - 6

- a. Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets.

- b. Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

Exercise-7

- a. Reading different types of data sets (.txt, .csv) from Web or disk and writing in file in specific disk location.
- b. Reading Excel data sheet in R.
- c. Reading XML dataset in R.

Exercise-8

- a. Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram (Introduction to ggplot0 graphics)
- b. Implement R Script to perform mean, median, mode, range, summary, variance, standard deviation operations.

Exercise-9

- a. Implement R Script to perform Normal, Binomial distributions.
- b. Implement R Script to perform correlation, Linear and multiple regression.

Exercise-10

Introduction to Non-Tabular Data Types: Time series, spatial data, Network data. Data Transformations: Converting Numeric Variables into Factors, Date Operations, String Parsing, Geocoding.

Exercise-11

Introduction Dirty data problems: Missing values, data manipulation, duplicates, forms of data dates, outliers, spelling.

Exercise-12

Data sources: SQLite examples for relational databases, Loading SPSS and SAS files, Reading from Google Spreadsheets, API and web scraping examples.

Reference Books:

1. R for Data Science is a book written by Hadley Wickham (Author), Garrett Golemund.

INTRODUCTION

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac. This programming language was named R, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of the Bell Labs Language S.

Evolution of R

R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

A large group of individuals has contributed to R by sending code and bug reports.

Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

Features of R

The following are the important features of R –

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

How to install R Studio on Ubuntu 20.04 step by step instructions

1. First, we need to install all prerequisites. Open up a terminal and enter:

```
$ sudo apt update
```

```
$ sudo apt -y install r-base gdebi-core
```

2. Next, download the R studio *.deb package from the official R studio website(<https://www.rstudio.com/products/rstudio/download/#download>). Head for the most recent Ubuntu release available, meaning if the Ubuntu 20 - focal package is not available, download the Ubuntu 18 - bionic version. Example of downloaded package:

```
$ ls
```

```
rstudio-2022.02.0-443-amd64.deb
```

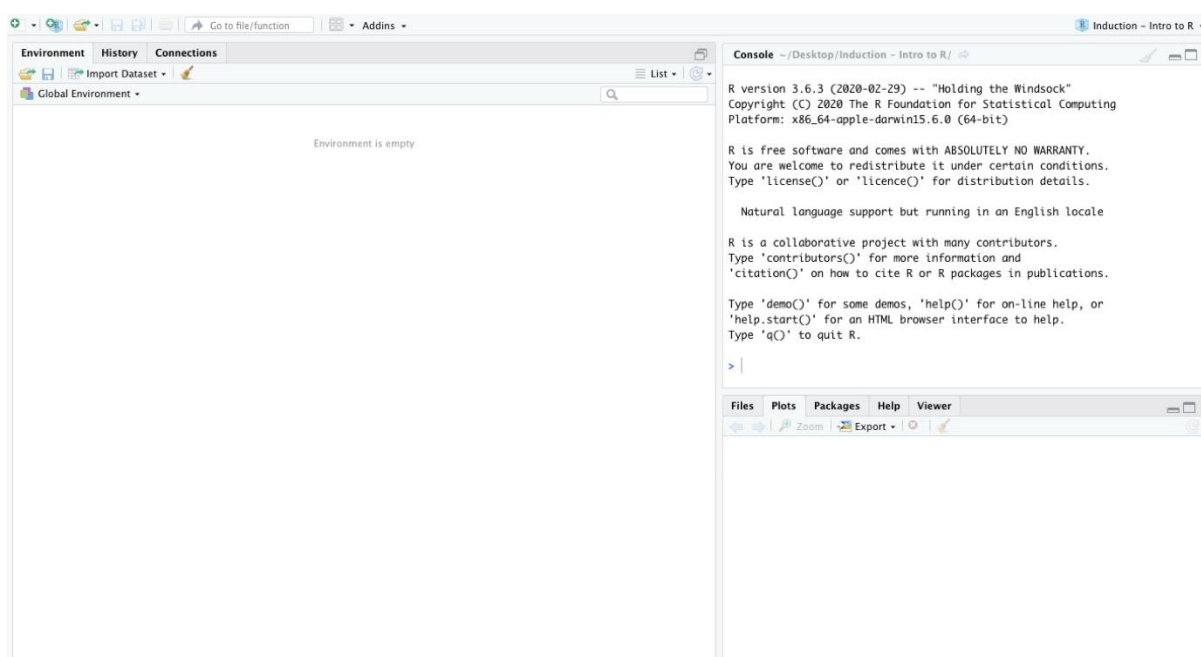
3. Use the gdebi command to install the previously downloaded package. The gdebi command will ensure that all additional prerequisites are also downloaded to fulfil the RStudio requirements:

```
$ sudo gdebi rstudio-2022.02.0-443-amd64.deb
```

4. Use your desktop menu to start the R Studio application. Launch R Studio on Ubuntu 20.04 Focal Fossa Linux. Alternatively, you can start the application by executing the below command:

```
$ rstudio
```

R-Studio interface



The following windows will appear as part of the RStudio user-interface:

1. **Console window** - The window on the right in Figure 1 is the standard R console window (the only one you would see if you used only R and not R via RStudio. This is the window where you type in commands and the results are returned.
2. **Workspace / History** - This window (left side in Figure 1) shows all the objects that you have created in the current R session (Workspace tab) and the commands you have used in the current R session (History tab).
3. **Files / Plots / Packages / Help** - This window (bottom right in Figure 1) is primarily used for displaying plots (graphs) and for using the help system.

R Data Types

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are –

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Type	Example	Verify
Logical	TRUE, FALSE	<pre>v <- TRUE print(class(v))</pre> it produces the following result – [1] "logical"
Numeric	12.3, 5, 999	<pre>v <- 23.5 print(class(v))</pre> it produces the following result – [1] "numeric"

Integer	2L, 34L, 0L	<pre>v <- 2L print(class(v))</pre> <p>it produces the following result – [1] "integer"</p>
Complex	3 + 2i	<pre>v <- 2+5i print(class(v))</pre> <p>it produces the following result – [1] "complex"</p>
Character	'a' , "good", "TRUE", '23.4'	<pre>v <- "TRUE" print(class(v))</pre> <p>it produces the following result – [1] "character"</p>
Raw	"Hello" is stored as 48 65 6c 6c 6f	<pre>v <- charToRaw("Hello") print(class(v))</pre> <p>it produces the following result – [1] "raw"</p>

In R programming, the very basic data types are the R-objects called **vectors** which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types. For example, we can use many atomic vectors and create an array whose class will become array.

Vectors

When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green','yellow')
print(apple)

# Get the class of the vector.
print(class(apple))
```

When we execute the above code, it produces the following result –

```
[1] "red""green""yellow"
[1] "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.  
list1 <- list(c(2,5,3),21.3, sin)  
  
# Print the list.  
print(list1)
```

When we execute the above code, it produces the following result –

```
[[1]]  
[1] 2 5 3  
  
[[2]]  
[1] 21.3  
  
[[3]]  
function (x) .Primitive("sin")
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.  
M = matrix( c('a','a','b','c','b','a'), nrow =2, ncol =3, byrow = TRUE)  
print(M)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]  
[1,] "a""a""b"  
[2,] "c""b""a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.  
a <- array(c('green','yellow'),dim = c(3,3,2))  
print(a)
```

When we execute the above code, it produces the following result –

```
,, 1
```

```
[1] [2] [3]
[1,] "green""yellow""green"
[2,] "yellow""green""yellow"
[3,] "green""yellow""green"

, , 2
```

```
[1] [2] [3]
[1,] "yellow""green""yellow"
[2,] "green""yellow""green"
[3,] "yellow""green""yellow"
```

Factors

Factors are the R-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor()** function. The **nlevels** function gives the count of levels.

```
# Create a vector.
apple_colors <- c('green','green','yellow','red','red','red','green')

# Create a factor object.
factor_apple <- factor(apple_colors)

# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

When we execute the above code, it produces the following result –

```
[1] green green yellow red red red green
Levels: green red yellow
[1] 3
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <- data.frame(
  gender = c("Male", "Male", "Female"),
  height = c(152, 171.5, 165),
  weight = c(81, 93, 78),
  Age = c(42, 38, 26)
)
print(BMI)
```

When we execute the above code, it produces the following result –

```
gender height weight Age
1 Male 152.0 81 42
2 Male 171.5 93 38
3 Female 165.0 78 26
```

Certainly! R is a programming language and environment designed for statistical computing and graphics. Here's some information about R programming:

1. Open Source:

- R is an open-source language, which means that its source code is freely available for users to view, modify, and distribute.

2. Designed for Data Analysis:

- R was specifically developed for statistical computing and data analysis. It provides a wide range of statistical and graphical techniques and is extensible through packages.

3. Comprehensive Package System:

- R has a vast ecosystem of packages contributed by the community. These packages cover a wide array of topics, from data manipulation and visualization to machine learning and advanced statistical modeling.

4. Data Structures:

- R supports various data structures, including vectors, matrices, data frames, and lists. These structures are crucial for handling and analyzing different types of data.

5. Data Visualization:

- R is known for its powerful data visualization capabilities. The ggplot2 package, for example, is widely used for creating high-quality graphics and visualizations.

6. Data Manipulation:

- R provides extensive tools for data manipulation. The dplyr and tidyr packages, for instance, offer functions for filtering, transforming, and cleaning data.

7. Statistical Modeling:

- R is equipped with a rich set of statistical functions and packages for various types of analyses. It supports linear and nonlinear modeling, time-series analysis, clustering, and more.

8. Reproducible Research:

- R is popular in the field of reproducible research. Projects and analyses conducted in R can be easily documented and shared, ensuring transparency and reproducibility.

9. RStudio:

- RStudio is a popular integrated development environment (IDE) for R. It provides a user-friendly interface for writing and executing R code, managing projects, and visualizing results.

10. Community and Support: -

R has a vibrant and active community. Users can seek help, share knowledge, and contribute to the development of R through forums, mailing lists, and social media.

11. Interoperability: -

R can be integrated with other languages like C, C++, and Python. This makes it flexible for tasks that may require the use of different programming languages.

12. Learning Resources: -

Numerous resources, including online courses, books, and tutorials, are available for learning R. The extensive documentation and community support make it accessible to users with various levels of expertise.

Certainly! In R, operators are symbols or functions that perform operations on variables and values. Here's a brief overview of the types of operators in R:

1. Arithmetic Operators:

- Examples: + (addition), - (subtraction), * (multiplication), / (division), %% (modulo), ^ (exponentiation).

2. Relational Operators:

- Examples: < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), == (equal to), != (not equal to).

3. Logical Operators:

- Examples: & (element-wise AND), | (element-wise OR), ! (element-wise NOT).

4. Assignment Operators:

- Examples: <- (assignment), -> (right assignment), = (assignment), += (addition assignment), -= (subtraction assignment), *=(multiplication assignment), /=` (division assignment).

5. Increment and Decrement Operators:

- Examples: ++ (increment by 1), -- (decrement by 1).

6. Bitwise Operators:

- Examples: & (bitwise AND), | (bitwise OR), xor() (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift).

7. Membership Operators:

- Examples: %in% (element of), !%in% (not an element of).

8. Identity Operators:

- Examples: identical() (TRUE if the objects are identical), isTRUE() (TRUE if the argument is TRUE).

9. Miscellaneous Operators:

- Examples: : (sequence operator), \$ (list element selection), ? (help operator).

Understanding and using these operators is crucial for performing various operations, comparisons, and assignments in R. They play a fundamental role in data manipulation, analysis, and programming tasks.

Certainly! I'll provide an outline of R programming concepts in the form of a tree. Each node in the tree represents a broader category or concept, and subnodes break down into more specific terms or functions.

- **R Programming**
 - | - **Basics**
 - | | - **Variables**
 - | | - **Data Types**
 - | | - **Operators**
 - | | - **Control Structures**
 - | | | - **Conditional Statements**
 - | | | - **Loops**
 - | - **Data Structures**
 - | | - **Vectors**
 - | | - **Matrices**
 - | | - **Data Frames**
 - | | - **Lists**
 - | - **Functions**
 - | | - **Built-in Functions**
 - | | - **User-Defined Functions**
 - | - **Data Import and Export**
 - | | - **read.csv()**
 - | | - **read.table()**
 - | | - **write.csv()**
 - | - **Data Manipulation**
 - | | - **dplyr Package**
 - | | | - **filter()**
 - | | | - **select()**
 - | | | - **mutate()**
 - | | | - **summarise()**
 - | | | - **arrange()**
 - | | - **tidyr Package**
 - | | | - **gather()**
 - | | | - **spread()**
 - | - **Data Visualization**
 - | | - **ggplot2 Package**
 - | | | - **ggplot()**
 - | | | - **aes()**
 - | | | - **geom_***
 - | | | - **facet_wrap()**
 - | - **Statistical Modeling**
 - | | - **lm() Function**
 - | | - **glm() Function**
 - | - **Reproducible Research**
 - | | - **R Markdown**
 - | | - **knitr Package**

- | - **External Integration**
 - | - **Interfaces with C, C++, Python**
 - | - **Shiny Package (Interactive Web Apps)**

Explanation of Terms:

- **Variables:** Storage locations with a symbolic name (identifier) that contains some known or unknown quantity or information.
- **Data Types:** The classification or categorization of data into different types, such as numeric, character, logical, etc.
- **Operators:** Symbols or functions that perform operations on variables and values.
- **Control Structures:** Statements that control the flow of program execution, including conditional statements (if, else) and loops (for, while).
- **Data Structures:** Different ways of organizing and storing data, including vectors, matrices, data frames, and lists.
- **Functions:** Blocks of code designed to perform a specific task. Can be built-in or user-defined.
- **Data Import and Export:** Functions and methods for reading and writing data to and from files.
- **Data Manipulation:** Techniques for transforming and reshaping data, often performed using the dplyr and tidyr packages.
- **Data Visualization:** Creating graphical representations of data, often done using the ggplot2 package.
- **Statistical Modeling:** Techniques for building statistical models, including linear and generalized linear models.
- **Reproducible Research:** Approaches to creating documents that combine code, results, and narrative for transparent and reproducible analyses.
- **External Integration:** Interaction with other programming languages (C, C++, Python) and tools like Shiny for building interactive web applications.

PROGRAMS

Exercise 1

- a) Installing R and RStudio
- b) Basic functionality of R, variable, data types in R

Learning Context

First, we need to install all prerequisites. Open up a terminal and enter:

```
$ sudo apt update
```

```
$ sudo apt -y install r-base gdebi-core
```

Next, download the R studio *.deb package from the official R studio website(<https://www.rstudio.com/products/rstudio/download/#download>). Head for the most recent Ubuntu release available, meaning if the Ubuntu 20 - focal package is not available, download the Ubuntu 18 - bionic version. Example of downloaded package:

```
$ ls
```

```
rstudio-2022.02.0-443-amd64.deb
```

Use the gdebi command to install the previously downloaded package. The gdebi command will ensure that all additional prerequisites are also downloaded to fulfil the RStudio requirements:

```
$ sudo gdebi
```

```
rstudio-2022.02.0-443-amd64.deb
```

Use your desktop menu to start the R Studio application. Launch R Studio on Ubuntu 20.04 Focal Fossa Linux. Alternatively, you can start the application by executing the below command:

```
$ rstudio
```

R Data Types

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of

the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are –

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Vectors

When you want to create vector with more than one element, you should use `c()` function which means to combine the elements into a vector.

Create a vector.

```
apple <- c('red','green',"yellow") print(apple)
```

Get the class of the vector.

```
print(class(apple))
```

When we execute the above code, it produces the following result –

```
[1] "red" "green" "yellow"
```

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

Lists



A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

Create a list.

```
list1 <- list(c(2,5,3),21.3, sin)
```

Print the list. `print(list1)`

When we execute the above code, it produces the following result –



[[1]]
[1] 2 5 3

Exercise 2

Learning Objective

- A) To implement an R script to show the usage of various operators available in R language.**
- B) Implement R script to read person's age from keyboard and display whether he is eligible for voting or not.**
- C) Implement R script to find biggest number between two numbers.**
- D) Implement R script to check the given year is leap year or not.**

Learning Context

Types of the operator in R language

- Arithmetic Operators
- Logical Operators
- Relational Operators
- Assignment Operators
- Miscellaneous Operators

Arithmetic Operators

These operators perform basic arithmetic operations like addition, subtraction, multiplication, division, exponent, modulus, etc.

For example:

```
x <- c(9,9,9)
```

```
y <- c(1,1,1)
```

```
print(x+y)
```

Output: [1] 10 10 10

Assignment Operators

The use of these operators is to assign values to the variables. There are two kinds of assignments, leftwards assignment, and rightwards assignment. Operators '<-' and '=' are used to assign values to any variable.

`x <- 3` or `x = 3` (Leftwards Assignment)

`3 -> x` or `x = 3` (Rightwards Assignment)

Logical Operators

These operators are used to perform Boolean operations like AND, OR, NOT, etc. on variables.

For example:

`x <- c(FALSE, TRUE, 3, 0)`

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

`!x` (NOT operation)

`x & y` (AND operation)

Output:

`[1] TRUE FALSE FALSE TRUE`

`[1] FALSE TRUE FALSE FALSE`

Relational Operators

These operators are used to compare two values or variables. To find if one is smaller, greater, equal, not equal, and other similar operations these operators are used. The output of a relational operator is always a Logical value, that is either TRUE or FALSE.

For example:-

```
x <- 10
```

```
y <- 5
```

Miscellaneous Operators

These R programming operators are used for special cases and are not for general mathematical or logical computation.

colon operator - It is used to generate a series of numbers in sequence for a vector.

%in% - This operator is used to check if an element belongs to a vector or not.

%*% - This operator multiplies a matrix with its transpose.

A) To implement an R script to show the usage of various operators available in R language

```
# Arithmetic Operators
```

```
a <- 10
```

```
b <- 5
```

```
addition <- a + b
```

```
subtraction <- a - b
```

```
multiplication <- a * b
```

```
division <- a / b
```

```
remainder <- a %% b
```

```
exponentiation <- a ^ b
```

```
print("Arithmetic Operators:")
```

```
print(paste("Addition:", addition))
```

```
print(paste("Subtraction:", subtraction))
```

```
print(paste("Multiplication:", multiplication))

print(paste("Division:", division))

print(paste("Remainder:", remainder))

print(paste("Exponentiation:", exponentiation))
```

```
# Relational Operators
```

```
x <- 7
```

```
y <- 12
```

```
equality <- x == y
```

```
inequality <- x != y
```

```
greater_than <- x > y
```

```
less_than <- x < y
```

```
greater_than_equal <- x >= y
```

```
less_than_equal <- x <= y
```

```
print("\nRelational Operators:")
```

```
print(paste("Equality:", equality))
```

```
print(paste("Inequality:", inequality))
```

```
print(paste("Greater Than:", greater_than))
```

```
print(paste("Less Than:", less_than))
```

```
print(paste("Greater Than or Equal:", greater_than_equal))
```

```
print(paste("Less Than or Equal:", less_than_equal))
```

```
# Logical Operators
```

```
p <- TRUE
```

```
q <- FALSE
```

```
logical_and <- p & q
```

```
logical_or <- p | q
```

```
logical_not <- !p
```



```
print("\nLogical Operators:")

print(paste("Logical AND:", logical_and))

print(paste("Logical OR:", logical_or))

print(paste("Logical NOT:", logical_not))
```

```
# Assignment Operators
```

```
m <- 3
```

```
n <- 8
```

```
m <- m + 2
```

```
n <- n * 2
```

```
print("\nAssignment Operators:")
```

```
print(paste("Updated m:", m))
```

```
print(paste("Updated n:", n))
```

output

```
[1] "Less Than: TRUE"
```

```
[1] "Greater Than or Equal: FALSE"
```

```
[1] "Less Than or Equal: TRUE"
```

```
[1] "\nLogical Operators:"
```

```
[1] "Logical AND: FALSE"
```

```
[1] "Logical OR: TRUE"
```

```
[1] "Logical NOT: FALSE"
```

```
[1] "\nAssignment Operators:"
```

```
[1] "Updated m: 5"
```

```
[1] "Updated n: 16"
```

```
[Execution complete with exit code 0]
```

B) Implement R script to read person's age from keyboard and display whether he is eligible for voting or not.

```
{  
  age <- as.integer(readline(prompt = "Enter your age :"))  
  
  if (age >= 18) {  
    print(paste("You are valid for voting :", age))  
  } else{  
    print(paste("You are not valid for voting :", age))  
  }  
  
}
```

Output:

```
Enter your age :48
```

```
[1] "You are valid for voting : 48"
```

C) Implement R script to find biggest number between two numbers.

```
{  
  x <- as.integer(readline(prompt = "Enter first number :"))  
  y <- as.integer(readline(prompt = "Enter second number :"))
```

```
z <- as.integer(readline(prompt = "Enter third number :"))
```

```
if (x > y && x > z) {  
  print(paste("Greatest is :", x))  
} else if (y > z) {  
  print(paste("Greatest is :", y))  
} else {  
  print(paste("Greatest is :", z))  
}
```

```
}
```

Output:

```
Enter first number :2
```

```
Enter second number :22
```

```
Enter third number :4
```

```
[1] "Greatest is : 22"
```

D) Implement R script to check the given year is leap year or not

```
# Program to check if the input year is a leap year or not
```

```
year = as.integer(readline(prompt="Enter a year: "))
```

```
if((year %% 4) == 0) {
```

```
  if((year %% 100) == 0) {
```

```
    if((year %% 400) == 0) {
```

```
      print(paste(year,"is a leap year"))
```

```
    } else {
```

```
      print(paste(year,"is not a leap year"))
```

```
    }
```

```
  } else {
```

```
    print(paste(year,"is a leap year"))
```

```
  }
```

```
} else {  
print(paste(year,"is not a leap year"))  
}
```

Output:

```
Enter a year: 1900      [1] "1900 is not a leap year"
```

Exercise - 3

- A) Implement R Script to create a list.
- B) Implement R Script to access elements in the list.
- C) Implement R Script to merge two or more lists. Implement R Script to perform matrix operation

Learning Objective

To implement R Script to create a list, accessing elements from the list and merging lists.

Learning Context

A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable.

To create a list, use the `list()` function:

```
# List of strings
thislist <- list("apple", "banana", "cherry")
# Print the list
thislist
output :
[[1]]
[1] "apple"
[[2]]
[1] "banana"
[[3]]
[1] "cherry"
```

Accessing components of a list

We can access components of an R list in two ways.

Access components by names: All the components of a list can be named and we can use those names to access the components of the R list using the dollar command.

Access components by indices: We can also access the components of the R list using indices. To access the top-level components of a R list we have to use a double slicing operator “[[]]” which is two

square brackets and if we want to access the lower or inner-level components of a R list we have to use another square bracket “[]” along with the double slicing operator “[[]]”.

a. Implement R Script to create a list.

solution :

Creating a list in R

Method 1: Using list() function

```
my_list1 <- list(1, "hello", TRUE, 3.14)
print("List created using list() function:")
print(my_list1)
```

Method 2: Using c() function

```
my_list2 <- c(1, "world", FALSE, 2.71)
print("List created using c() function:")
print(my_list2)
```

Method 3: Creating an empty list and adding elements later

```
empty_list <- list()
empty_list[[1]] <- 42
empty_list[[2]] <- "openai"
empty_list[[3]] <- FALSE
print("Empty list with elements added later:")
```



```
print(empty_list)
```

Output

```
[1] "List created using list() function:"  
[[1]]  
[1] 1  
  
[[2]]  
[1] "hello"  
  
[[3]]  
[1] TRUE  
  
[[4]]  
[1] 3.14  
  
[1] "List created using c() function:"  
[1] "1"    "world" "FALSE" "2.71"  
[1] "Empty list with elements added later:"  
[[1]]  
[1] 42  
  
[[2]]  
[1] "openai"  
  
[[3]]  
[1] FALSE
```

[Execution complete with exit code 0]

b. Implement R Script to access elements in the list.

Solution :

Accessing elements in a list

Creating a list

```
my_list <- list(1, "hello", TRUE, 3.14)
```

Accessing elements by index

```
element1 <- my_list[[1]]
```

```
element2 <- my_list[[2]]
```

```
element3 <- my_list[[3]]
```

```
element4 <- my_list[[4]]
```

Printing the elements

```
print("Accessing elements in the list:")
```

```
print(paste("Element 1:", element1))
```

```
print(paste("Element 2:", element2))
```

```
print(paste("Element 3:", element3))
```

```
print(paste("Element 4:", element4))
```

Accessing elements using names (if the list has names)


```
named_list <- list(a = 10, b = "world", c = FALSE)
```

```
element_a <- named_list[["a"]]
```

```
element_b <- named_list[["b"]]
```

```
element_c <- named_list[["c"]]
```

```
# Printing elements from the named list
```

```
print("\nAccessing elements in the named list:")
```

```
print(paste("Element 'a':", element_a))
```

```
print(paste("Element 'b':", element_b))
```

```
print(paste("Element 'c':", element_c))
```

```
print(paste("Element 'c':", element_c))
```

Output

```
[1] "Accessing elements in the list:"
```

```
[1] "Element 1: 1"
```

```
[1] "Element 2: hello"
```

```
[1] "Element 3: TRUE"
```

```
[1] "Element 4: 3.14"
```

```
[1] "\nAccessing elements in the named list:"
```

```
[1] "Element 'a': 10"
```

```
[1] "Element 'b': world"
```

```
[1] "Element 'c': FALSE"
```

```
[Execution complete with exit code 0]
```

c. Implement R Script to merge two or more lists.

```
# Merge two or more lists in R
```

```
# Creating two lists
```

```
list1 <- list(1, "hello", TRUE)
```

```
list2 <- list(3.14, "world", FALSE)
```

```
# Merging lists using the c() function
```

```
merged_list <- c(list1, list2)
```

```
# Printing the original lists and the merged list
```

```
print("Original List 1:")
```

```
print(list1)
```

```
print("\nOriginal List 2:")
```

```
print(list2)
```

```
print("\nMerged List:")
```

```
print(merged_list)
```

Output

```
[1] "Original List 1:"
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] "hello"
```

```
[[3]]
```

```
[1] TRUE
```

```
[1] "\nOriginal List 2:"
```

```
[[1]]
```

```
[1] 3.14
```

```
[[2]]
```

```
[1] "world"
```

```
[[3]]
```

```
[1] FALSE
```

```
[1] "\nMerged List:"
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] "hello"
```

```
[[3]]
```

```
[1] TRUE
```

```
[[4]]  
[1] 3.14  
  
[[5]]  
[1] "world"  
  
[[6]]  
[1] FALSE
```

[Execution complete with exit code 0]

d. Implement R Script to perform matrix operation.

Solution:

```
# Matrix operations in R  
# Creating two matrices  
matrix1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)  
matrix2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2, byrow = TRUE)  
# Displaying the original matrices  
print("Matrix 1:")  
print(matrix1)  
print("\nMatrix 2:")  
print(matrix2)  
# Matrix addition  
matrix_sum <- matrix1 + matrix2  
print("\nMatrix Addition:")  
print(matrix_sum)
```


```
# Matrix subtraction
matrix_diff <- matrix1 - matrix2
print("\nMatrix Subtraction:")
print(matrix_diff)

# Matrix multiplication
matrix_prod <- matrix1 %*% matrix2
print("\nMatrix Multiplication:")
print(matrix_prod)

# Transposing a matrix
matrix1_transposed <- t(matrix1)
print("\nTransposed Matrix 1:")
print(matrix1_transposed)
```

Output

```
[1] "Matrix 1:"
      [,1] [,2]
[1,]   1   2
[2,]   3   4
[1] "\nMatrix 2:"
      [,1] [,2]
[1,]   5   6
[2,]   7   8
[1] "\nMatrix Addition:"
      [,1] [,2]
[1,]   6   8
[2,]  10  12
[1] "\nMatrix Subtraction:"
      [,1] [,2]
```



```
[1,] -4 -4
```

```
[2,] -4 -4
```

```
[1] "\nMatrix Multiplication:"
```

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

```
[1,] 19 22
```

```
[2,] 43 50
```

```
[1] "\nTransposed Matrix 1:"
```

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

```
[1,]  1  3
```

```
[2,]  2  4
```

```
[Execution complete with exit code 0]
```

Exercise - 4

Implement R script to perform following operations:

- A) Various operations on vectors**
- B) Finding the sum and average of given numbers using arrays.**
- C) To display elements of list in reverse order.**
- D) Finding the minimum and maximum elements in the array.**

Learning Objective

To Implement R script to perform operations on Vectors and arrays.

Learning Context

In R, arrays are the data objects which allow us to store data in more than two dimensions. In R, an array is created with the help of the **array()** function. This array() function takes a vector as an input and to create an array it uses vectors values in the **dim** parameter.

For example- If we create an array of dimension (2, 3, 4) then it will create 4 rectangular matrices of 2 row and 3 columns.

R Array Syntax

There is the following syntax of R arrays:

1. `array_name <- array(data, dim= (row_size, column_size, matrices, dim_names))`

The data is the first argument in the array() function. It is an input vector which is given to the array.

matrices

In R, the array consists of multi-dimensional matrices.

row_size

This parameter defines the number of row elements which an array can store.

`column_size`

This parameter defines the number of columns elements which an array can store.

`dim_names`

This parameter is used to change the default names of rows and columns.

In R, array creation is quite simple. We can easily create an array using vector and `array()` function. In array, data is stored in the form of the matrix. There are only two steps to create a matrix which are as follows

1. In the first step, we will create two vectors of different lengths.
2. Once our vectors are created, we take these vectors as inputs to the array.

Example to understand how we can implement an array with the help of the vectors and `array()` function.

Example

1. `#Creating two vectors of different lengths`
2. `vec1 <-c(1,3,5)`
3. `vec2 <-c(10,11,12,13,14,15)`
- 4.
5. `#Taking these vectors as input to the array`
6. `res <- array(c(vec1,vec2),dim=c(3,3,2))`
7. `print(res)`

Output

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



```
[1,] 1 10 13
```

```
[2,] 3 11 14
```

```
[3,] 5 12 15
```

```
., 2
```

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

```
[1,] 1 10 13
```

```
[2,] 3 11 14
```

```
[3,] 5 12 15
```

Naming rows and columns

In R, we can give the names to the rows, columns, and matrices of the array. This is done with the help of the `dim name` parameter of the `array()` function.

It is not necessary to give the name to the rows and columns. It is only used to differentiate the row and column for better understanding.

Below is an example, in which we create two arrays and giving names to the rows, columns, and matrices.

Example

1. `#Creating two vectors of different lengths`
2. `vec1 <-c(1,3,5)`
3. `vec2 <-c(10,11,12,13,14,15)`
- 4.
5. `#Initializing names for rows, columns and matrices`

```

6.    col_names <- c("Col1","Col2","Col3")
7.    row_names <- c("Row1","Row2","Row3")
8.    matrix_names <- c("Matrix1","Matrix2")
9.
10.   #Taking the vectors as input to the array
11.   res                                     <-
array(c(vec1,vec2),dim=c(3,3,2),dimnames=list(row_names,col_names,matrix_names))
12.   print(res)

```

Output

, , Matrix1

	Col1	Col2	Col3
Row1	1	10	13
Row2	3	11	14
Row3	5	12	15

, , Matrix2

	Col1	Col2	Col3
Row1	1	10	13
Row2	3	11	14
Row3	5	12	15

Accessing array elements

Like C or C++, we can access the elements of the array. The elements are accessed with the help of the index. Simply, we can access the elements of the array with the help of the indexing method. Let see an example to understand how we can access the elements of the array using the indexing method.

Example

1. , , Matrix1
2. Col1 Col2 Col3
3. Row1 1 10 13
4. Row2 3 11 14
5. Row3 5 12 15
- 6.
7. , , Matrix2
8. Col1 Col2 Col3
9. Row1 1 10 13
10. Row2 3 11 14
11. Row3 5 12 15
- 12.
13. Col1 Col2 Col3
14. 5 12 15
- 15.
16. [1] 13

17.

18. Col1 Col2 Col3

19. Row1 1 10 13

20. Row2 3 11 14

21. Row3 5 12 15

Manipulation of elements

The array is made up matrices in multiple dimensions so that the operations on elements of an array are carried out by accessing elements of the matrices.

Example

1. #Creating two vectors of different lengths

2. vec1 <-c(1,3,5)

3. vec2 <-c(10,11,12,13,14,15)

4.

5. #Taking the vectors as input to the array1

6. res1 <- array(c(vec1,vec2),dim=c(3,3,2))

7. print(res1)

8.

9. #Creating two vectors of different lengths

10. vec1 <-c(8,4,7)

11. vec2 <-c(16,73,48,46,36,73)

12.

13. #Taking the vectors as input to the array2

```
14. res2 <- array(c(vec1,vec2),dim=c(3,3,2))
15. print(res2)
16.
17. #Creating matrices from these arrays
18. mat1 <- res1[,2]
19. mat2 <- res2[,2]
20. res3 <- mat1+mat2
21. print(res3)
```

Output

```
,, 1
```

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

```
[1,]  1 10 13
```

```
[2,]  3 11 14
```

```
[3,]  5 12 15
```

```
,, 2
```

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

```
[1,]  1 10 13
```

```
[2,]  3 11 14
```

```
[3,]  5 12 15
```

```
,, 1
```

```

[,1] [,2] [,3]
[1,]  8 16 46
[2,]  4 73 36
[3,]  7 48 73

```

```

,,2

```

```

[,1] [,2] [,3]
[1,]  8 16 46
[2,]  4 73 36
[3,]  7 48 73

```

```

[,1] [,2] [,3]
[1,]  9 26 59
[2,]  7 84 50
[3,] 12 60 88

```

Calculations across array elements

For calculation purpose, R provides `apply()` function. This `apply` function contains three parameters i.e., `x`, `margin`, and `function`.

This function takes the array on which we have to perform the calculations. The basic syntax of the `apply()` function is as follows:

1. `apply(x, margin, fun)`

Here, x is an array, and a margin is the name of the dataset which is used and fun is the function which is to be applied to the elements of the array.

Example

1. `#Creating two vectors of different lengths`
2. `vec1 <-c(1,3,5)`
3. `vec2 <-c(10,11,12,13,14,15)`
- 4.
5. `#Taking the vectors as input to the array1`
6. `res1 <- array(c(vec1,vec2),dim=c(3,3,2))`
7. `print(res1)`
- 8.
9. `#using apply function`
10. `result <- apply(res1,c(1),sum)`
11. `print(result)`

Output

`., 1`

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

`[1,] 1 10 13`

`[2,] 3 11 14`

`[3,] 5 12 15`

`., 2`

```
[,1] [,2] [,3]
[1,]  1  10  13
[2,]  3  11  14
[3,]  5  12  15
```

```
[1] 48 56 64
```

A) various operations on vectors

Various operations on vectors in R

Creating vectors

```
vec1 <- c(1, 2, 3, 4, 5)
```

```
vec2 <- c(6, 7, 8, 9, 10)
```

Displaying the original vectors

```
print("Vector 1:")
```

```
print(vec1)
```

```
print("\nVector 2:")
```

```
print(vec2)
```

Vector addition

```
vec_sum <- vec1 + vec2
```

```
print("\nVector Addition:")
```

```
print(vec_sum)
```

Vector subtraction

```
vec_diff <- vec1 - vec2
```

```
print("\nVector Subtraction:")
```




```
print(vec_diff)
```

```
# Element-wise multiplication
```

```
vec_prod <- vec1 * vec2
```

```
print("\nElement-wise Multiplication:")
```

```
print(vec_prod)
```

```
# Element-wise division
```

```
vec_div <- vec1 / vec2
```

```
print("\nElement-wise Division:")
```

```
print(vec_div)
```

```
# Vector concatenation
```

```
vec_concat <- c(vec1, vec2)
```

```
print("\nVector Concatenation:")
```

```
print(vec_concat)
```

```
# Vector length
```

```
length_vec1 <- length(vec1)
```

```
length_vec2 <- length(vec2)
```

```
print("\nVector Length:")
```

```
print(paste("Length of Vector 1:", length_vec1))
```

```
print(paste("Length of Vector 2:", length_vec2))
```

```
# Sum and mean of vectors
```

```
sum_vec1 <- sum(vec1)
```

```
mean_vec2 <- mean(vec2)
```

```
print("\nSum and Mean of Vectors:")
```

```
print(paste("Sum of Vector 1:", sum_vec1))
```

```
print(paste("Mean of Vector 2:", mean_vec2))
```

Output

```
[1] "Vector 1:"  
[1] 1 2 3 4 5  
[1] "\nVector 2:"  
[1] 6 7 8 9 10  
[1] "\nVector Addition:"  
[1] 7 9 11 13 15  
[1] "\nVector Subtraction:"  
[1] -5 -5 -5 -5 -5  
[1] "\nElement-wise Multiplication:"  
[1] 6 14 24 36 50  
[1] "\nElement-wise Division:"  
[1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000  
[1] "\nVector Concatenation:"  
[1] 1 2 3 4 5 6 7 8 9 10  
[1] "\nVector Length:"  
[1] "Length of Vector 1: 5"  
[1] "Length of Vector 2: 5"  
[1] "\nSum and Mean of Vectors:"  
[1] "Sum of Vector 1: 15"  
[1] "Mean of Vector 2: 8"  
  
[Execution complete with exit code 0]
```

B) Finding the sum and average of given numbers using arrays.

```
# Finding the sum and average of given numbers using arrays in R

# Creating an array (one-dimensional)
numbers <- c(10, 15, 20, 25, 30)

# Displaying the original array
print("Original Array:")
print(numbers)

# Calculating the sum
sum_numbers <- sum(numbers)
print(paste("\nSum of Numbers:", sum_numbers))

# Calculating the average
average_numbers <- mean(numbers)
print(paste("Average of Numbers:", average_numbers))
```

Output

```
[1] "Original Array:"
[1] 10 15 20 25 30
[1] "\nSum of Numbers: 100"
[1] "Average of Numbers: 20"
```

C) To display elements of list in reverse order.

```
# Displaying elements of a list in reverse order

# Creating a sample list
my_list <- list("apple", "banana", "cherry", "date", "elderberry")

# Displaying the original list
print("Original List:")
print(my_list)
```

```
# Reversing the list using rev() function
```

```
reversed_list <- rev(my_list)
```

```
# Displaying the list in reverse order
```

```
print("\nList in Reverse Order:")
```

```
print(reversed_list)
```

Output

```
[1] "Original List:"
```

```
[[1]]
```

```
[1] "apple"
```

```
[[2]]
```

```
[1] "banana"
```

```
[[3]]
```

```
[1] "cherry"
```

```
[[4]]
```

```
[1] "date"
```

```
[[5]]
```

```
[1] "elderberry"
```

```
[1] "\nList in Reverse Order:"
```

```
[[1]]
```

```
[1] "elderberry"
```

```
[[2]]
[1] "date"

[[3]]
[1] "cherry"

[[4]]
[1] "banana"

[[5]]
[1] "apple"
```

```
[Execution complete with exit code 0]
```

D) Finding the minimum and maximum elements in the array.

Finding the minimum and maximum elements in an array in R

Creating an array (or a vector)

```
numbers <- c(10, 15, 7, 25, 30, 5, 18)
```

Displaying the original array

```
print("Original Array:")
```

```
print(numbers)
```

Finding the minimum element

```
min_element <- min(numbers)
```

```
print(paste("\nMinimum Element:", min_element))
```

```
# Finding the maximum element
max_element <- max(numbers)
print(paste("Maximum Element:", max_element))
```

Output

```
[1] "Original Array:"
[1] 10 15 7 25 30 5 18
[1] "\nMinimum Element: 5"
[1] "Maximum Element: 30"

[Execution complete with exit code 0]
```

Exercise 5

- a. Implement R Script to perform various operations on matrices
- b. Implement R Script to extract the data from dataframes.
- c. Write R script to display file contents.
- d. Write R script to copy file contents from one file to another

Learning Objective

Implement R Script to extract the data from data frames and to display file contents.

Learning Context

A data frame is a two-dimensional array-like structure or a table in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a special case of the list in which each component has equal length.

A data frame is used to store a data table and the vectors which are present in the form of a list in a data frame, are of equal length. It is a list of equal length vectors. A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.

In R, one can perform various types of operations on a data frame like **accessing** rows and columns, selecting the subset of the data frame, editing data frames, delete rows and columns in a data frame, etc.

Creating a File

Using **file.create()** function, a new file can be created from console or truncates if already exists. The function returns a TRUE logical value if file is created otherwise, returns FALSE.

Syntax:

```
file.create(" ")
```

Parameters:

" ": name of the file is passed in " " that has to be created

Writing into a File

[write.table\(\)](#) function in R programming is used to write an object to a file. This function is present in **utils** package in R and writes data frame or matrix object to any type of file.

Syntax:

`write.table(x, file)`

Parameters:

x: indicates the object that has to be written into the file
file: indicates the name of the file that has to be written

Renaming a File

The **file.rename()** function renames the file and return a logical value. The function renames files but not directories.

Syntax:

`file.rename(from, to)`

Parameters:

from: indicates current file name or path
to: indicates new file name or path

Copy a File

The **file.copy()** function in R helps to create a copy of specified file from console itself.

Syntax:

`file.copy(from, to)`

Parameters:

from: indicates the file path that has to be copied
to: indicates the path where it has to be copied
To know about more optional parameters, use below command in console: `help("file.copy")`

a. Implement R Script to perform various operations on matrices

```
# Various operations on matrices in R
```

```
# Creating matrices
```

```
matrix1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
```

```
matrix2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2, byrow = TRUE)
```

```
# Displaying the original matrices
```

```
print("Matrix 1:")
```

```
print(matrix1)
```

```
print("\nMatrix 2:")
```

```
print(matrix2)
```

```
# Matrix addition
```

```
matrix_sum <- matrix1 + matrix2
```

```
print("\nMatrix Addition:")
```

```
print(matrix_sum)
```

```
# Matrix subtraction
```

```
matrix_diff <- matrix1 - matrix2
```

```
print("\nMatrix Subtraction:")
```

```
print(matrix_diff)
```

```
# Matrix multiplication (element-wise)
```

```
matrix_prod_elementwise <- matrix1 * matrix2
```

```
print("\nElement-wise Matrix Multiplication:")
```

```
print(matrix prod elementwise)
```

```
# Matrix multiplication (dot product)
```

```
matrix prod <- matrix1 %*% matrix2
```

```
print("\nMatrix Multiplication (Dot Product):")
```

```
print(matrix prod)
```

```
# Transposing a matrix
```

```
matrix1 transposed <- t(matrix1)
```

```
print("\nTransposed Matrix 1:")
```

```
print(matrix1 transposed)
```

```
# Determinant of a matrix
```

```
matrix det <- det(matrix1)
```

```
print("\nDeterminant of Matrix 1:")
```

```
print(matrix det)
```

Output

```
[1] "Matrix 1:"
```

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

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

```
[2,]  3  4
```

```

[1] "\nMatrix 2:"
    [,1] [,2]
[1,]  5  6
[2,]  7  8
[1] "\nMatrix Addition:"
    [,1] [,2]
[1,]  6  8
[2,] 10 12
[1] "\nMatrix Subtraction:"
    [,1] [,2]
[1,] -4 -4
[2,] -4 -4
[1] "\nElement-wise Matrix Multiplication:"
    [,1] [,2]
[1,]  5 12
[2,] 21 32
[1] "\nMatrix Multiplication (Dot Product):"
    [,1] [,2]
[1,] 19 22
[2,] 43 50
[1] "\nTransposed Matrix 1:"
    [,1] [,2]
[1,]  1  3
[2,]  2  4
[1] "\nDeterminant of Matrix 1:"
[1] -2

```

[Execution complete with exit code 0]

b. Implement R Script to extract the data from dataframes

Implementing R Script to extract data from dataframes

Creating a sample dataframe

```
data <- data.frame(  
  Name = c("Alice", "Bob", "Charlie", "David", "Eva"),  
  Age = c(25, 30, 22, 35, 28),  
  Score = c(90, 85, 95, 80, 88)  
)
```

Displaying the original dataframe

```
print("Original Dataframe:")  
print(data)
```

Extracting specific columns

```
name_column <- data$Name  
age_column <- data$Age  
print("\nExtracted Columns:")  
print("Name Column:")  
print(name_column)  
print("Age Column:")  
print(age_column)
```

```
# Extracting specific rows
```

```
selected_rows <- data[c(1, 3, 5), ]
```

```
print("\nSelected Rows:")
```

```
print(selected_rows)
```

```
# Extracting specific rows and columns
```

```
subset_data <- data[2:4, c("Name", "Score")]
```

```
print("\nSubset of Dataframe:")
```

```
print(subset_data)
```

Output

```
[1] "Original Dataframe:"
```

```
  Name Age Score
```

```
1 Alice 25  90
```

```
2  Bob 30  85
```

```
3 Charlie 22  95
```

```
4 David 35  80
```

```
5  Eva 28  88
```

```
[1] "\nExtracted Columns:"
```

```
[1] "Name Column:"
```

```
[1] "Alice" "Bob" "Charlie" "David" "Eva"
```

```
[1] "Age Column:"
```

```
[1] 25 30 22 35 28
```

```
[1] "\nSelected Rows:"
```

```
  Name Age Score
```

```
1 Alice 25 90
3 Charlie 22 95
5 Eva 28 88
[1] "\nSubset of Dataframe:"
      Name Score
2 Bob 85
3 Charlie 95
4 David 80
```

[Execution complete with exit code 0]

c. Write R script to display file contents.

```
# Displaying file contents in R
```

```
# File path and name
```

```
file_path <- "path/to/your/file.txt"
```

```
# Reading and displaying file contents
```

```
file_contents <- readLines(file_path)
```

```
cat("File Contents:\n")
```

```
cat(file_contents, sep = "\n")
```

d. Write R script to copy file contents from one file to another

```
# Copying file contents from one file to another in R
```




```
# Source file path and name
```

```
source_file_path <- "path/to/source/file.txt"
```

```
# Destination file path and name
```

```
destination_file_path <- "path/to/destination/destination_file.txt"
```

```
# Read contents from the source file
```

```
file_contents <- readLines(source_file_path)
```

```
# Write contents to the destination file
```

```
writeLines(file_contents, destination_file_path)
```

```
cat("File contents copied successfully from", source_file_path, "to", destination_file_path, "\n")
```

Exercise - 6

- a. Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets.
- b. Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

Learning Objective

To implement R Script to find basic descriptive statistics using summary, str, quartile function on mtcars& cars datasets.

To implement R Script to subset (), aggregate () functions on iris dataset.

Learning Context

In Descriptive analysis, we are describing our data with the help of various representative methods like using charts, graphs, tables, excel files, etc. In the descriptive analysis, we describe our data in some manner and present it in a meaningful way so that it can be easily understood. Most of the time it is performed on small data sets and this analysis helps us a lot to predict some future trends based on the current findings. Some measures that are used to describe a data set are measures of central tendency and measures of variability or dispersion.

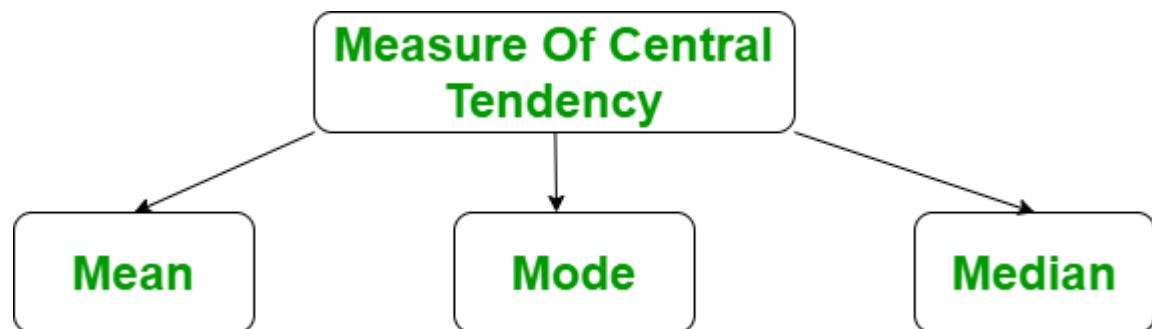
Process of Descriptive Analysis

- The measure of central tendency
- Measure of variability

Measure of central tendency

It represents the whole set of data by a single value. It gives us the location of central points. There are three main measures of central tendency:

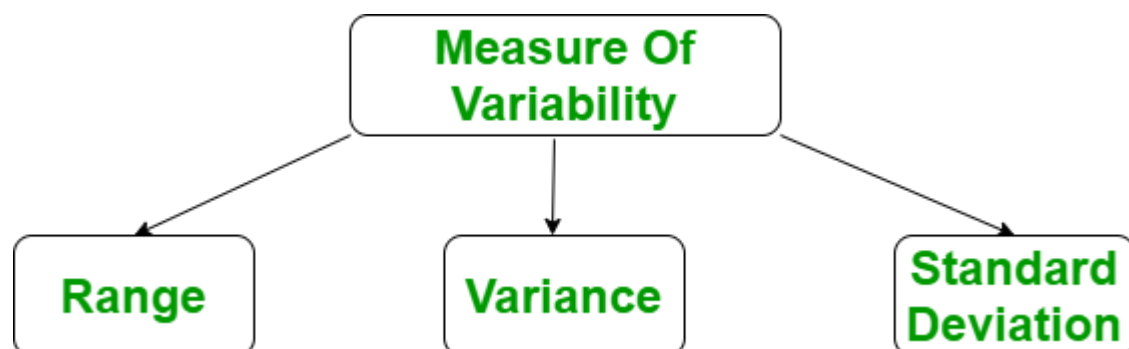
- Mean
- Mode
- Median



Measure of variability

Measure of variability is known as the spread of data or how well is our data is distributed. The most common variability measures are:

- Range
- Variance
- Standard deviation



a. Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets.

Load the required datasets

```
data(mtcars)
```

```
data(cars)
```

Displaying basic descriptive statistics for mtcars dataset

```
cat("Summary for mtcars dataset:\n")
```

```
print(summary(mtcars))
```

```
cat("\nStructure of mtcars dataset:\n")
str(mtcars)
```

```
cat("\nQuantiles for mtcars dataset:\n")
quantiles_mtcars <- quantile(mtcars$mpg, probs = c(0, 0.25, 0.5, 0.75, 1))
print(quantiles_mtcars)
```

```
# Displaying basic descriptive statistics for cars dataset
cat("\nSummary for cars dataset:\n")
print(summary(cars))
```

```
cat("\nStructure of cars dataset:\n")
str(cars)
```

```
cat("\nQuantiles for cars dataset:\n")
quantiles_cars <- quantile(cars$speed, probs = c(0, 0.25, 0.5, 0.75, 1))
print(quantiles_cars)
```

Summary for mtcars dataset:

mpg	cyl	disp	hp
Min. :10.40	Min. :4.000	Min. : 71.1	Min. : 52.0
1st Qu.:15.43	1st Qu.:4.000	1st Qu.:120.8	1st Qu.: 96.5
Median :19.20	Median :6.000	Median :196.3	Median :123.0
Mean :20.09	Mean :6.188	Mean :230.7	Mean :146.7
3rd Qu.:22.80	3rd Qu.:8.000	3rd Qu.:326.0	3rd Qu.:180.0
Max. :33.90	Max. :8.000	Max. :472.0	Max. :335.0

drat	wt	qsec	vs
Min. :2.760	Min. :1.513	Min. :14.50	Min. :0.0000
1st Qu.:3.080	1st Qu.:2.581	1st Qu.:16.89	1st Qu.:0.0000

```

Median :3.695  Median :3.325  Median :17.71  Median :0.0000
Mean   :3.597  Mean   :3.217  Mean   :17.85  Mean   :0.4375
3rd Qu.:3.920  3rd Qu.:3.610  3rd Qu.:18.90  3rd Qu.:1.0000
Max.   :4.930  Max.   :5.424  Max.   :22.90  Max.   :1.0000

      am      gear      carb
Min.   :0.0000  Min.   :3.000  Min.   :1.000
1st Qu.:0.0000  1st Qu.:3.000  1st Qu.:2.000
Median :0.0000  Median :4.000  Median :2.000
Mean   :0.4062  Mean   :3.688  Mean   :2.812
3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:4.000
Max.   :1.0000  Max.   :5.000  Max.   :8.000

```

Structure of mtcars dataset:

```

'data.frame':      32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

```

Quantiles for mtcars dataset:

```
0% 25% 50% 75% 100%
10.400 15.425 19.200 22.800 33.900
```

Summary for cars dataset:

```
speed      dist
Min.   :4.0  Min.   : 2.00
1st Qu.:12.0 1st Qu.:26.00
Median :15.0 Median :36.00
Mean   :15.4 Mean   :42.98
3rd Qu.:19.0 3rd Qu.:56.00
Max.   :25.0 Max.   :120.00
```

Structure of cars dataset:

```
'data.frame':      50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

Quantiles for cars dataset:

```
0% 25% 50% 75% 100%
4 12 15 19 25
```

[Execution complete with exit code 0]

b. Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

```
# Load the iris dataset
data(iris)
```

```
# Displaying the structure of the iris dataset
```

```

cat("Structure of iris dataset:\n")
str(iris)

# Subset of the iris dataset where Sepal.Length is greater than 5
subset_iris <- subset(iris, Sepal.Length > 5)

cat("\nSubset of iris dataset where Sepal.Length > 5:\n")
print(subset_iris)

# Aggregate function to calculate mean Petal.Length for each Species
aggregate_result <- aggregate(Petal.Length ~ Species, data = iris, FUN = mean)

cat("\nAggregate result - Mean Petal.Length for each Species:\n")
print(aggregate_result)

```

Output

Structure of iris dataset:

'data.frame': 150 obs. of 5 variables:

\$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

\$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

\$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

\$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

\$ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...

Subset of iris dataset where Sepal.Length > 5:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
11	5.4	3.7	1.5	0.2	setosa

15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
24	5.1	3.3	1.7	0.5	setosa
28	5.2	3.5	1.5	0.2	setosa
29	5.2	3.4	1.4	0.2	setosa
32	5.4	3.4	1.5	0.4	setosa
33	5.2	4.1	1.5	0.1	setosa
34	5.5	4.2	1.4	0.2	setosa
37	5.5	3.5	1.3	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
45	5.1	3.8	1.9	0.4	setosa
47	5.1	3.8	1.6	0.2	setosa
49	5.3	3.7	1.5	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
53	6.9	3.1	4.9	1.5	versicolor
54	5.5	2.3	4.0	1.3	versicolor
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
59	6.6	2.9	4.6	1.3	versicolor

60	5.2	2.7	3.9	1.4 versicolor
62	5.9	3.0	4.2	1.5 versicolor
63	6.0	2.2	4.0	1.0 versicolor
64	6.1	2.9	4.7	1.4 versicolor
65	5.6	2.9	3.6	1.3 versicolor
66	6.7	3.1	4.4	1.4 versicolor
67	5.6	3.0	4.5	1.5 versicolor
68	5.8	2.7	4.1	1.0 versicolor
69	6.2	2.2	4.5	1.5 versicolor
70	5.6	2.5	3.9	1.1 versicolor
71	5.9	3.2	4.8	1.8 versicolor
72	6.1	2.8	4.0	1.3 versicolor
73	6.3	2.5	4.9	1.5 versicolor
74	6.1	2.8	4.7	1.2 versicolor
75	6.4	2.9	4.3	1.3 versicolor
76	6.6	3.0	4.4	1.4 versicolor
77	6.8	2.8	4.8	1.4 versicolor
78	6.7	3.0	5.0	1.7 versicolor
79	6.0	2.9	4.5	1.5 versicolor
80	5.7	2.6	3.5	1.0 versicolor
81	5.5	2.4	3.8	1.1 versicolor
82	5.5	2.4	3.7	1.0 versicolor
83	5.8	2.7	3.9	1.2 versicolor
84	6.0	2.7	5.1	1.6 versicolor
85	5.4	3.0	4.5	1.5 versicolor
86	6.0	3.4	4.5	1.6 versicolor
87	6.7	3.1	4.7	1.5 versicolor

88	6.3	2.3	4.4	1.3 versicolor
89	5.6	3.0	4.1	1.3 versicolor
90	5.5	2.5	4.0	1.3 versicolor
91	5.5	2.6	4.4	1.2 versicolor
92	6.1	3.0	4.6	1.4 versicolor
93	5.8	2.6	4.0	1.2 versicolor
95	5.6	2.7	4.2	1.3 versicolor
96	5.7	3.0	4.2	1.2 versicolor
97	5.7	2.9	4.2	1.3 versicolor
98	6.2	2.9	4.3	1.3 versicolor
99	5.1	2.5	3.0	1.1 versicolor
100	5.7	2.8	4.1	1.3 versicolor
101	6.3	3.3	6.0	2.5 virginica
102	5.8	2.7	5.1	1.9 virginica
103	7.1	3.0	5.9	2.1 virginica
104	6.3	2.9	5.6	1.8 virginica
105	6.5	3.0	5.8	2.2 virginica
106	7.6	3.0	6.6	2.1 virginica
108	7.3	2.9	6.3	1.8 virginica
109	6.7	2.5	5.8	1.8 virginica
110	7.2	3.6	6.1	2.5 virginica
111	6.5	3.2	5.1	2.0 virginica
112	6.4	2.7	5.3	1.9 virginica
113	6.8	3.0	5.5	2.1 virginica
114	5.7	2.5	5.0	2.0 virginica
115	5.8	2.8	5.1	2.4 virginica
116	6.4	3.2	5.3	2.3 virginica

117	6.5	3.0	5.5	1.8	virginica
118	7.7	3.8	6.7	2.2	virginica
119	7.7	2.6	6.9	2.3	virginica
120	6.0	2.2	5.0	1.5	virginica
121	6.9	3.2	5.7	2.3	virginica
122	5.6	2.8	4.9	2.0	virginica
123	7.7	2.8	6.7	2.0	virginica
124	6.3	2.7	4.9	1.8	virginica
125	6.7	3.3	5.7	2.1	virginica
126	7.2	3.2	6.0	1.8	virginica
127	6.2	2.8	4.8	1.8	virginica
128	6.1	3.0	4.9	1.8	virginica
129	6.4	2.8	5.6	2.1	virginica
130	7.2	3.0	5.8	1.6	virginica
131	7.4	2.8	6.1	1.9	virginica
132	7.9	3.8	6.4	2.0	virginica
133	6.4	2.8	5.6	2.2	virginica
134	6.3	2.8	5.1	1.5	virginica
135	6.1	2.6	5.6	1.4	virginica
136	7.7	3.0	6.1	2.3	virginica
137	6.3	3.4	5.6	2.4	virginica
138	6.4	3.1	5.5	1.8	virginica
139	6.0	3.0	4.8	1.8	virginica
140	6.9	3.1	5.4	2.1	virginica
141	6.7	3.1	5.6	2.4	virginica
142	6.9	3.1	5.1	2.3	virginica
143	5.8	2.7	5.1	1.9	virginica

```
144      6.8      3.2      5.9      2.3 virginica
145      6.7      3.3      5.7      2.5 virginica
146      6.7      3.0      5.2      2.3 virginica
147      6.3      2.5      5.0      1.9 virginica
148      6.5      3.0      5.2      2.0 virginica
149      6.2      3.4      5.4      2.3 virginica
150      5.9      3.0      5.1      1.8 virginica
```

Aggregate result - Mean Petal.Length for each Species:

```
Species Petal.Length
1  setosa      1.462
2 versicolor  4.260
3  virginica   5.552
```

[Execution complete with exit code 0]

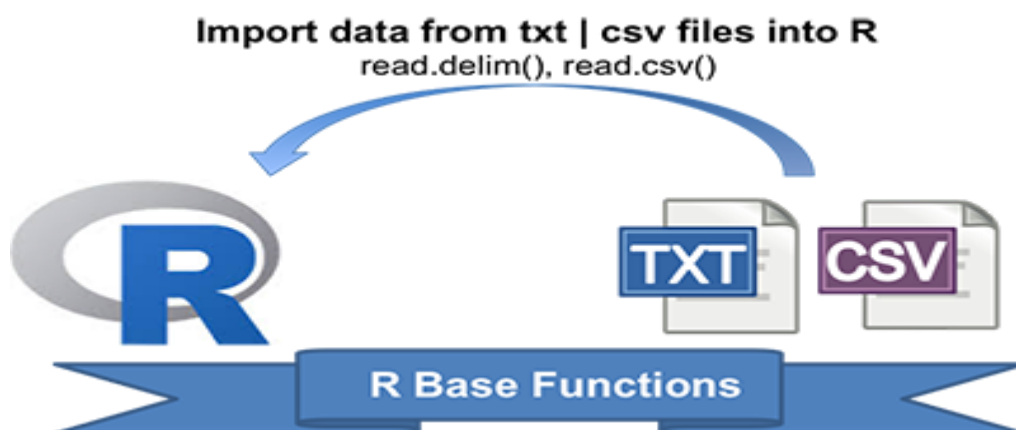
Exercise–7

- Reading different types of data sets (.txt, .csv) from Web or disk and writing in file in specific disk location.
- Reading Excel data sheet in R.
- Reading XML dataset in R.

Learning Objective

To implement an R script to read different types of data sets (.txt, .csv) from Web or disk and writing in file in specific disk location, reading Excel datasheet and XML datasets in R.

Learning Context



R base functions for importing data

The R base function **`read.table()`** is a general function that can be used to read a file in table format. The data will be imported as a [data frame](#).

- **`read.csv()`**: for reading “**comma separated value**” files (“.csv”).
- **`read.csv2()`**: variant used in countries that use a comma “,” as decimal point and a semicolon “;” as field separators.
- **`read.delim()`**: for reading “*tab-separated value*” files (“.txt”). By default, point (“.”) is used as decimal points.
- **`read.delim2()`**: for reading “*tab-separated value*” files (“.txt”). By default, comma (“,”) is used as decimal points.

Reading Excel data sheet in R.

Method 1: Using read_excel() from readxl

Syntax:

```
read_excel(path)
```

The **read_excel()** method extracts the data from the Excel file and returns it as an R data frame. It instantly recognizes the Excel file's sheets and imports the data from the default sheet, which is often the first page. The sheet option allows us to give a specific sheet's name or index in order to read that particular sheet.

Example:

```
library(readxl)
```

```
Data_gfg <- read_excel("Data_gfg.xlsx")
```

```
Data_gfg
```

Output:

```
  group value1 value2
1 Group A  9.539645 11.584165
2 Group A 13.117417  9.469207
3 Group A 10.141017 16.337912
4 Group A 10.258575 14.415924
5 Group A 13.430130  9.753783
6 Group A 10.921832 11.194230
7 Group A  7.469878 11.066689
8 Group A  8.626294 13.559930
9 Group A  9.108676 11.833262
10 Group A 12.448164 12.506637
```

Method 2: Using read.xlsx() from xlsx

read.xlsx() function is imported from the xlsx library of R language and used to read/import an excel file in R language.

Syntax:

```
read.xlsx(path)
```

We can deal with structured data from spreadsheets and incorporate Excel data with other R packages and workflows by reading Excel files in R using the `xlsx` package. For additional processing and analysis, Excel files can be read into R as data frames using the `read.xlsx()` function.

Example:

```
install.packages('xlsx')
```

```
Data_gfg <-read.xlsx('Data_gfg.xlsx')
```

Data_gfg

Output:

```
  group  value1  value2
1 Group A 9.539645 11.584165
2 Group A 13.117417 9.469207
3 Group A 10.141017 16.337912
4 Group A 10.258575 14.415924
5 Group A 13.430130 9.753783
6 Group A 10.921832 11.194230
7 Group A 7.469878 11.066689
8 Group A 8.626294 13.559930
9 Group A 9.108676 11.833262
10 Group A 12.448164 12.506637
```

Reading XML dataset in R

XML which stands for Extensible Markup Language is made up of markup tags, wherein each tag illustrates the information carried by the particular attribute in the XML file. We can work with the

XML files using the XML package provided by [R](#). The package has to be explicitly installed using the following command:

```
install.packages("XML")
```

Creating XML file

XML files can be created by saving the data with the respective tags containing information about the content and saving it with '.xml'. We will use the following XML file 'sample.xml' to see the various operations that can be performed on the file:

```
<RECORDS>
<STUDENT>
  <ID>1</ID>
  <NAME>Alia</NAME>
  <MARKS>620</MARKS>
  <BRANCH>IT</BRANCH>
</STUDENT>
<STUDENT>
  <ID>2</ID>
  <NAME>Brijesh</NAME>
  <MARKS>440</MARKS>
  <BRANCH>Commerce</BRANCH>
</STUDENT>
<STUDENT>
  <ID>3</ID>
  <NAME>Yash</NAME>
  <MARKS>600</MARKS>
  <BRANCH>Humanities</BRANCH>
</STUDENT>
<STUDENT>
  <ID>4</ID>
  <NAME>Mallika</NAME>
  <MARKS>660</MARKS>
  <BRANCH>IT</BRANCH>
</STUDENT>
<STUDENT>
  <ID>5</ID>
  <NAME>Zayn</NAME>
  <MARKS>560</MARKS>
```

```
<BRANCH>IT</BRANCH>
</STUDENT>
</RECORDS>
```

Reading XML File

The XML file can be read after installing the package and then parsing it with **xmlparse()** function, which takes as input the XML file name and prints the content of the file in the form of a list. The file should be located in the current working directory. An additional package named ‘methods’ should also be installed. The following code can be used to read the contents of the file “sample.xml”.

loading the library and other important packages

```
library("XML")
library("methods")
```

the contents of sample.xml are parsed

```
data <- xmlParse(file = "sample.xml")
```

```
print(data)
```

a. Reading different types of data sets (.txt, .csv) from Web or disk and writing in file in specific disk location.

```
# Reading and writing datasets in R
```

```
# Load necessary libraries
```

```
library(utils)
```

```
# Set the working directory (replace with your desired directory)
```

```
setwd("/path/to/your/directory")
```

```
# Reading a text file from the web
```

```
url_text <- "https://example.com/data.txt"
```

```
text_data <- readLines(url_text)
```

```
# Writing the text data to a file
writeLines(text_data, "text_data.txt")

cat("Text data successfully written to text_data.txt\n")

# Reading a CSV file from the web
url_csv <- "https://example.com/data.csv"
csv_data <- read.csv(url_csv)

# Writing the CSV data to a file
write.csv(csv_data, "csv_data.csv", row.names = FALSE)

cat("CSV data successfully written to csv_data.csv\n")
```



b. Reading Excel data sheet in R.

```
# Install and load the readxl package
if (!requireNamespace("readxl", quietly = TRUE)) {
  install.packages("readxl")
}

library(readxl)

# Set the working directory (replace with your desired directory)
setwd("/path/to/your/directory")

# Specify the Excel file and sheet name
excel_file <- "example.xlsx"
sheet_name <- "Sheet1"
```

```
# Read the Excel sheet into a data frame
excel_data <- read_excel(excel_file, sheet = sheet_name)

# Display the first few rows of the data frame
print("First few rows of the Excel data:")
print(head(excel_data))
```

c. Reading XML dataset in R.

```
# Install and load the XML package
if (!requireNamespace("XML", quietly = TRUE)) {
  install.packages("XML")
}



library(XML)

# Set the working directory (replace with your desired directory)
setwd("/path/to/your/directory")

# Specify the XML file
xml_file <- "example.xml"

# Read the XML file into an R object
xml_data <- xmlTreeParse(xml_file, useInternalNodes = TRUE)

# Display the structure of the XML data
print("Structure of the XML data:")
print(xml_data)
```



```
# Extract information from the XML data (modify as needed based on your XML structure)
root_node <- xmlRoot(xml_data)
child_nodes <- xmlChildren(root_node)

cat("Contents of the XML data:\n")
for (child in child_nodes) {
  cat(xmlName(child), ": ", xmlValue(child), "\n")
}
```

Exercise-8

- Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram (Introduction to ggplot0 graphics)
- Implement R Script to perform mean, median, mode, range, summary, variance, standard deviation operations.

Learning Objective

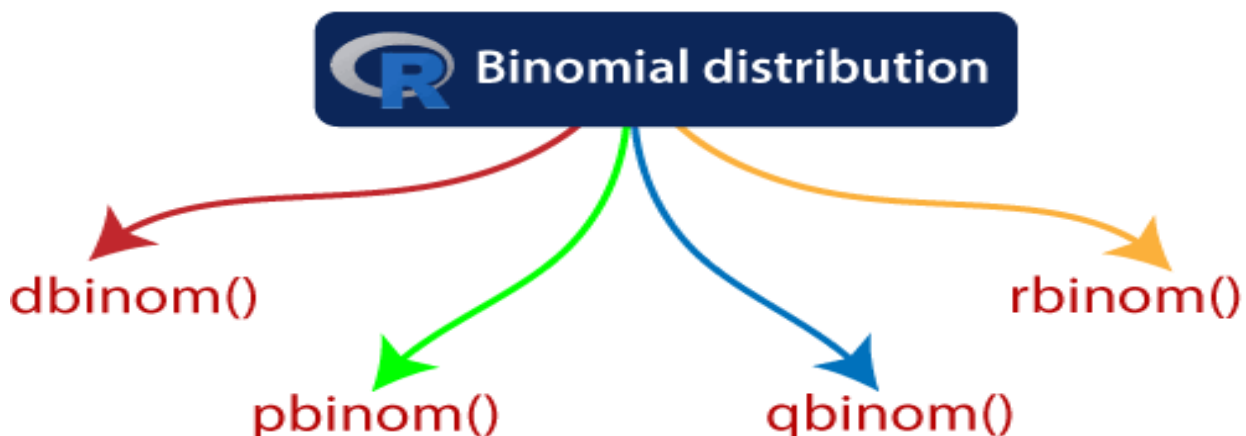
- To implement R Script to perform Normal, Binomial distributions.
- To implement R Script to perform correlation, Linear and multiple regression

Learning Context

Binomial Distribution

The binomial distribution is also known as **discrete probability distribution**, which is used to find the probability of success of an event. The event has only two possible outcomes in a series of experiments. The tossing of the coin is the best example of the binomial distribution. When a coin is tossed, it gives either a head or a tail. The probability of finding exactly three heads in repeatedly tossing the coin ten times is approximate during the binomial distribution.

R allows us to create binomial distribution by providing the following function:



These function can have the following parameters:

S.No	Parameter	Description
1.	x	It is a vector of numbers.
2.	p	It is a vector of probabilities.
3.	n	It is a vector of observations.
4.	size	It is the number of trials.
5.	prob	It is the probability of the success of each trial.

a. Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram (Introduction to ggplot0 graphics)

Install and load the ggplot2 package

```
if (!requireNamespace("ggplot2", quietly = TRUE)) {
  install.packages("ggplot2")
}
```

library(ggplot2)

Create a sample dataset

```
data <- data.frame(
  Category = c("A", "B", "C", "D"),
  Value = c(30, 20, 15, 35)
)
```

Pie Chart

```
pie_chart <- ggplot(data, aes(x = "", y = Value, fill = Category)) +  
  geom_bar(stat = "identity", width = 1, color = "white") +  
  coord_polar("y") +  
  ggtitle("Pie Chart") +  
  theme_void()
```

Bar Chart

```
bar_chart <- ggplot(data, aes(x = Category, y = Value, fill = Category)) +  
  geom_bar(stat = "identity", color = "white") +  
  ggtitle("Bar Chart") +  
  theme_minimal()
```

Scatter Plot

```
scatter_plot_data <- data.frame(  
  X = c(1, 2, 3, 4, 5),  
  Y = c(10, 8, 12, 6, 14)  
)  
  
scatter_plot <- ggplot(scatter_plot_data, aes(x = X, y = Y)) +  
  geom_point(size = 3, color = "blue") +  
  ggtitle("Scatter Plot") +  
  theme_minimal()
```

Histogram

```
histogram_data <- data.frame(  
  Values = c(25, 30, 40, 35, 50, 45, 55, 60, 70, 65)  
)
```

```
histogram <- ggplot(histogram_data, aes(x = Values)) +  
  geom_histogram(binwidth = 5, fill = "orange", color = "black", alpha = 0.7) +  
  ggtitle("Histogram")
```

b. Implement R Script to perform mean, median, mode, range, summary, variance, standard deviation operations.

```
# Create a sample dataset
```

```
data <- c(12, 15, 20, 25, 30, 35, 40, 45, 50, 55)
```

```
# Ensure that data is a numeric vector
```

```
data <- as.numeric(data)
```

```
# Mean
```

```
mean_value <- mean(data, na.rm = TRUE)
```

```
cat("Mean:", mean_value, "\n")
```

```
# Median
```

```
median_value <- median(data, na.rm = TRUE)
```

```
cat("Median:", median_value, "\n")
```

```
# Mode (using a custom function)
```

```
mode <- function(x) {
```

```
  ux <- unique(x)
```

```
  ux[which.max(tabulate(match(x, ux)))]
```

```
}
```

```
mode_value <- mode(data)
```

```
cat("Mode:", mode_value, "\n")
```

```
# Range
```

```
range_value <- range(data)
cat("Range:", diff(range_value), "\n")

# Summary
summary_stats <- summary(data)
cat("Summary Stats:\n")
print(summary_stats)

# Variance
variance_value <- var(data, na.rm = TRUE)
cat("Variance:", variance_value, "\n")

# Standard Deviation
std_dev_value <- sd(data, na.rm = TRUE)
cat("Standard Deviation:", std_dev_value, "\n")
```

Output

Mean: 32.7

Median: 32.5

Mode: 12

Range: 43

Summary Stats:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
------	---------	--------	------	---------	------

12.00	21.25	32.50	32.70	43.75	55.00
-------	-------	-------	-------	-------	-------

Variance: 219.5667

Standard Deviation: 14.81778

[Execution complete with exit code 0]

Exercise-9

- a. Implement R Script to perform Normal, Binomial distributions.
- b. Implement R Script to perform correlation, Linear and multiple regression.

a. Implement R Script to perform Normal, Binomial distributions.

```
# Install and load the ggplot2 package
if (!requireNamespace("ggplot2", quietly = TRUE)) {
  install.packages("ggplot2")
}

library(ggplot2)

# Set a seed for reproducibility
set.seed(123)

# Generate data for Normal distribution
normal_data <- rnorm(1000, mean = 0, sd = 1)

# Create a histogram for the Normal distribution
normal_plot <- ggplot(data.frame(x = normal_data), aes(x)) +
  geom_histogram(binwidth = 0.2, fill = "blue", color = "black", alpha = 0.7) +
  ggtitle("Normal Distribution") +
  theme_minimal()

# Generate data for Binomial distribution
binomial_data <- rbinom(1000, size = 20, prob = 0.5)
```



```
# Create a histogram for the Binomial distribution
```

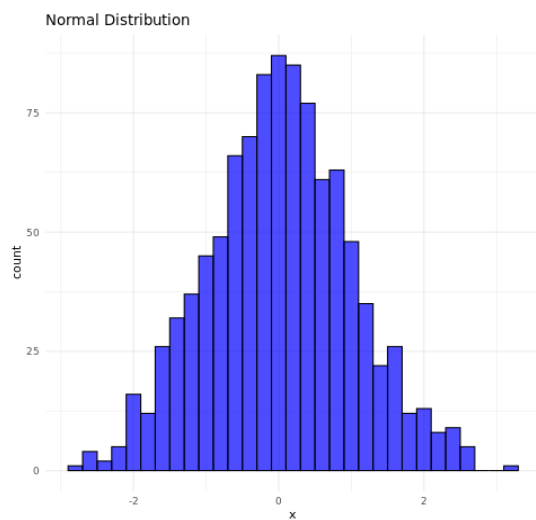
```
binomial_plot <- ggplot(data.frame(x = binomial_data), aes(x)) +  
  geom_histogram(binwidth = 1, fill = "green", color = "black", alpha = 0.7) +  
  ggtitle("Binomial Distribution") +  
  theme_minimal()
```

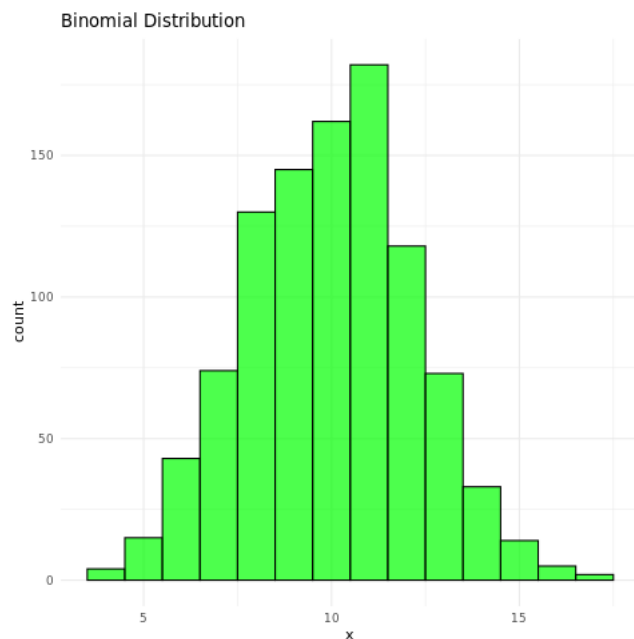
```
# Display the plots
```

```
print(normal_plot)
```

```
print(binomial_plot)
```

Output





b. Implement R Script to perform correlation, Linear and multiple regression.

```
# Load a sample dataset (you can replace this with your own dataset)
# For demonstration purposes, we will use the built-in 'mtcars' dataset
data(mtcars)
```

```
# Correlation
```

```
correlation_matrix <- cor(mtcars[, c("mpg", "disp", "hp", "wt")])
```

```
cat("Correlation Matrix:\n")
```

```
print(correlation_matrix)
```

```
# Linear Regression
```

```
linear_model <- lm(mpg ~ wt, data = mtcars)
```

```
cat("\nLinear Regression Summary:\n")
```

```
print(summary(linear_model))
```

```
# Multiple Regression
multiple_model <- lm(mpg ~ disp + hp + wt, data = mtcars)

cat("\nMultiple Regression Summary:\n")
print(summary(multiple_model))
```

Output

Correlation Matrix:

	mpg	disp	hp	wt
mpg	1.0000000	-0.8475514	-0.7761684	-0.8676594
disp	-0.8475514	1.0000000	0.7909486	0.8879799
hp	-0.7761684	0.7909486	1.0000000	0.6587479
wt	-0.8676594	0.8879799	0.6587479	1.0000000

Linear Regression Summary:

Call:

```
lm(formula = mpg ~ wt, data = mtcars)
```

Residuals:

Min	1Q	Median	3Q	Max
-4.5432	-2.3647	-0.1252	1.4096	6.8727

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	37.2851	1.8776	19.858	< 2e-16 ***
wt	-5.3445	0.5591	-9.559	1.29e-10 ***

```

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,    Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF, p-value: 1.294e-10

```

Multiple Regression Summary:

Call:

```
lm(formula = mpg ~ disp + hp + wt, data = mtcars)
```

Residuals:

Min	1Q	Median	3Q	Max
-3.891	-1.640	-0.172	1.061	5.861

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	37.105505	2.110815	17.579	< 2e-16 ***
disp	-0.000937	0.010350	-0.091	0.92851
hp	-0.031157	0.011436	-2.724	0.01097 *
wt	-3.800891	1.066191	-3.565	0.00133 **

```

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 2.639 on 28 degrees of freedom

Multiple R-squared: 0.8268, Adjusted R-squared: 0.8083

F-statistic: 44.57 on 3 and 28 DF, p-value: 8.65e-11

[Execution complete with exit code 0]

Exercise—10

Introduction to Non-Tabular Data Types: Time series, spatial data, Network data. Data Transformations: Converting Numeric Variables into Factors, Date Operations, String Parsing, Geocoding

Introduction to Non-Tabular Data Types in R:

1. Time Series Data:

- Description: Time series data in R represents observations collected over time, and it is commonly used in fields such as finance, economics, and environmental science.
- Handling in R: Time series data can be managed using specialized packages like xts or ts. Functions like ts() help convert data frames into time series objects, enabling time-based analyses.

2. Spatial Data:

- Description: Spatial data in R involves information tied to geographic locations. This type of data is prevalent in fields like geography, ecology, and urban planning.
- Handling in R: Spatial data is managed using packages such as sf and sp. These packages facilitate the creation, manipulation, and visualization of spatial objects. Operations like spatial joins and overlays are common.

3. Network Data:

- Description: Network data in R represents relationships between entities, modeled as nodes and edges in a graph. Examples include social networks, transportation systems, and communication networks.
- Handling in R: The igraph package is widely used for network analysis in R. It allows users to create, analyze, and visualize graphs. Functions like graph.data.frame() and centrality() aid in handling network data.

Data Transformations in R:

1. Converting Numeric Variables into Factors:

- Transformation: Factors are categorical variables in R. Conversion is done using the factor() function.
- Example Code:

```
# Convert numeric variable to factor
numeric_variable <- c(1, 2, 1, 3, 2)
```

```
factor_variable <- factor(numeric_variable)
```

```
)
```

2. Date Operations:

- Transformation: R has a rich set of functions for date manipulation, including `as.Date()`, `format()`, and `difftime()`.

- Example Code:

```
# Date operations
```

```
date_string <- "2023-01-01"
```

```
date_object <- as.Date(date_string)
```

3. String Parsing:

- Transformation: The `strsplit()` and `gsub()` functions are commonly used for string parsing and manipulation.

- Example Code:

```
# String parsing
```

```
text <- "John,Doe,30"
```

```
parsed_data <- strsplit(text, ",")
```

4. Geocoding:

- Transformation: Geocoding involves converting location names or addresses into geographic coordinates. The `ggmap` or `tmap` packages can be used.

- Example Code:

```
# Geocoding with ggmap
```

```
library(ggmap)
```

```
location <- "New York, NY"
```

```
geo_data <- geocode(location)
```

In R, these data types and transformations are supported by various packages and functions, making it a versatile language for working with diverse datasets and performing necessary transformations for analysis and visualization.

Exercise—11

Introduction Dirty data problems: Missing values, data manipulation, duplicates, forms of data dates, outliers, spelling.

Introduction to Dirty Data Problems in R:

1. Missing Values:

- Issue: Incomplete datasets with missing values for certain variables.
- Handling in R: Functions like `is.na()`, `complete.cases()`, and `na.omit()` help identify and handle missing values through imputation or removal.

2. Data Manipulation:

- Issue: Inconsistencies or errors resulting from manual entry, typos, or inconsistencies.
- Handling in R: Data cleaning techniques using functions such as `gsub()`, `str_replace()`, and `tolower()` can standardize formats and correct errors.

3. Duplicates:

- Issue: Presence of identical records in the dataset.
- Handling in R: Functions like `duplicated()` and `unique()` assist in identifying and removing duplicate entries based on certain criteria or variables.

4. Forms of Data Dates:

- Issue: Inconsistent date formats or data recorded as text instead of date/time.
- Handling in R: R provides functions like `as.Date()`, `format()`, and `lubridate` package functions to standardize date formats and convert text to date objects.

5. Outliers:

- Issue: Observations significantly deviating from the general pattern.
- Handling in R: Techniques include using statistical methods like the Z-score, the `outliers` package, or visualization tools like box plots and scatter plots to identify and handle outliers.

6. Spelling Errors:

- Issue: Incorrectly spelled or inconsistent entries in text data.
- Handling in R: Text processing functions like `stringdist`, `stringdistmatrix`, and string manipulation functions (`str_replace()`, `strsplit()`) can be employed to detect and correct spelling errors.

In R, addressing dirty data problems involves a combination of base R functions and additional packages like `dplyr`, `tidyr`, `stringr`, and `lubridate`. A systematic approach to cleaning and preprocessing data is crucial for accurate analyses and reliable results. Data scientists and analysts often use a combination of exploratory data analysis and specific cleaning techniques to ensure data quality in their R programming workflows.

Exercise–12

Data sources: SQLite examples for relational databases, Loading SPSS and SAS files, Reading from Google Spreadsheets, API and web scraping examples.

Data Sources in R:

1. SQLite Examples for Relational Databases:

- Description: SQLite is a lightweight, file-based relational database management system.
- Handling in R: The RSQLite package in R facilitates interaction with SQLite databases. Functions like `dbConnect()` and `dbGetQuery()` enable connecting to databases and executing SQL queries.
- Example Code:

```
library(RSQLite)

# Connect to SQLite database

con <- dbConnect(SQLite(), dbname = "example.db")

# Execute SQL query

result <- dbGetQuery(con, "SELECT * FROM table_name")
```

2. Loading SPSS and SAS Files:

- Description: SPSS (Statistical Package for the Social Sciences) and SAS (Statistical Analysis System) are statistical software packages, and their data files are common in research.
- Handling in R: The haven package in R provides functions like `read_sav()` and `read_sas()` to read data from SPSS and SAS files, respectively.
- Example Code:

```
library(haven)

# Read SPSS file

data_spss <- read_sav("example.sav")

# Read SAS file

data_sas <- read_sas7bdat("example.sas7bdat")
```

3. Reading from Google Spreadsheets:

- Description: Google Spreadsheets is a cloud-based spreadsheet tool.

- Handling in R: The googlesheets4 package facilitates reading from Google Spreadsheets. Functions like gs4_get() and gs4_read() are used.

- Example Code:

```
library(googlesheets4)

# Authenticate and access Google Spreadsheet

gs <- gs4_get("Spreadsheet Name")

# Read data

data_gs <- gs4_read(gs)
```

4. API and Web Scraping Examples:

- Description: APIs (Application Programming Interfaces) and web scraping are used to extract data from online sources.

- Handling in R:

- Using APIs: The httr package helps interact with APIs. Functions like GET() and content() are commonly used.

```
library(httr)

# API request example

response <- GET("https://api.example.com/data")

data_api <- content(response, "text")
```

- Web Scraping: Packages like rvest and httr assist in web scraping.

```
library(rvest)

# Web scraping example

url <- "https://example.com"

page <- read_html(url)

data_scraped <- html_text(html_nodes(page, "div.example-class"))
```

These examples illustrate how R can interact with various data sources, including relational databases, statistical software files, cloud-based spreadsheets, APIs, and websites, making it a versatile tool for data analysis and integration.