Roll No. 45 Exam Seat No.

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R. C. Marg,
Chembur, Mumbai – 400074. Contact No. 02261532532

Since 1962

CERTIFICATE

Certified that Mr. Pushkar Prasad Sane

of FYMCA/A has

satisfactorily completed a course of the necessary experiments in

Data Analytics with R LAB under my supervision

in the Institute of Technology in the academic year 20 23 – 2024.

Principal Head of Department

Lab In-charge Subject Teacher

V.E.S. Institute of Technology, Collector Colony,
Chembur, Mumbai, Maharashtra 400047
Department of M.C.A

INDEX

Sr. No.	Contents	Date of Preparation	Date of Submission	Marks	Faculty Sign
1	Packages, Write commands for following Install packages, loading packages Data types, checking type of variable, printing variable and objects (Vector, Matrix, List, Factor, Data frame, Table)	15-09-2023	16-09-2023		
2	Write commands for following cbind-ing and rbind-ing, Reading and Writing data. setwd(), getwd(), data(), rm(), Attaching and Detaching data. Reading data from the consol. Loading data from different data sources.(CSV, Excel).	19-09-2023	20-09-2023		
3	Write commands for Implementation of Data preprocessing techniques like, Naming and Renaming variables, adding a new variable. Dealing with missing data. Dealing with categorical data. Data reduction using subsetting.	17-09-2023	28-09-2023		

4	Write commands for Implementation of Data reduction using subsetting, implementation and usage Dplyr & Tidyverse, select, transmute, arrange, filter, groupby on any dataset.	13-10-2023	13-10-2023		
5	Write commands for Working with different types of R Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts.	13-10-2023	19-10-2023		
6	Implement data Visualization with Ggplot2.	22-10-2023	27-10-2023		
7	Implement commands for drawing various Correlation Plots and learn the process of EDA.	14-10-2023	20-10-2023		
8	Implementation of Normal and Binomial distributions, Univariate and Bivariate Analysis.	26-10-2023	27-10-2023		
9	Implementation and analysis of Apriori Algorithm using Market Basket Analysis.	21-10-2023	27-10-2023		
10	Implementation and analysis of Linear regression through graphical methods.	24-10-2023	23-10-2023		

<table>
<tr><td colspan="3">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td>Lab Assignment Number: 1</td></tr>
<tr><td colspan="3">Title of Lab Assignment: Study of basic command in R Studio.</td></tr>
<tr><td colspan="2">DOP: 15-09-2023</td><td>DOS: 16-09-2023</td></tr>
<tr><td>CO Mapped:
CO1</td><td>PO Mapped:
PO1, PO2, PO3, PO4, PO6, PO7, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 1

Title: Study of basic command in R Studio.

Aim: Write the commands for the following Data acquisition, install packages, loading packages, Data types, checking type of variables, printing variables and objects (Vector, matrix, list, factor, data frame, tables).

Theory:

1. **Data acquisition:** In R programming, there are various ways to acquire data, whether from local files, databases, APIs, or other sources. Here are some commonly used data acquisition commands in R:

   Reading Data from Files:

   Reading CSV files: read.csv(), read.csv2(), read.delim(), read.table()

   Reading Excel files: read.xlsx() (using the openxlsx package) or readxl package

   Reading text files: readLines(), scan()

   Example: # Reading a CSV file data

   <- read.csv("data.csv") # Reading an Excel file library(openxlsx) data

   <- read.xlsx("data.xlsx")

2. **Install Packages:** In R programming, the install.packages() command is used to install packages from CRAN (Comprehensive R Archive Network) or other repositories. Packages are collections of functions, data sets, and documentation that provide additional functionality to R.
   Example: install.packages("package_name")

3. **Loading Packages:** In R programming, you can load packages using the library() function. Loading a package makes its functions, datasets, and other features available for use in your R script or session.
   Example: library(package_name)

4. **Data Types:** In R programming, there are several data types you can work with. Some common data types in R include:

   a. **Numeric**: Used for representing numerical values, such as integers or decimals. For example, 5, 3.14.

   b. **Character:** Used for representing text or strings. Strings are enclosed in double or single quotes. For example, "Hello", 'World'.

   c. **Logical:** Used for representing boolean values, which can be either TRUE or FALSE.

   d. **Integer:** A special type of numeric data that represents whole numbers without decimals. For example, 10, -3.

   e. **Matrix:** A matrix is a 2-dimensional data structure with rows and columns.
      Example: mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

   f. **Data Frame:** A data frame is a 2-dimensional table-like structure where each column can have a different data type. Eg: df <- data.frame(name = c("Alice", "Bob"), age = c(25, 30)).

   g. **List:** A list is a collection of elements that can be of different data types.
      Eg: my_list <- list(name = "Alice", age = 25, is_student = TRUE)

5. **Checking types of variables:** In R programming, you can use several functions to determine the data type of an object or variable. Here are some common functions you can use to find data types:

   a. **class():** The class() function returns the class or data type of an R object.
      Function:

```
x <- 5
y <- “Hello”
print(class(x))          # Output: “numeric”
print(class(y))          # Output: “character”
```

b. typeof(): The typeof() function returns the fundamental object of an R object.

Function:

```
x <- 5
y <- “Hello”
print(typeof(x))        # Output: “double”
print(class(y))         # Output: “character”
```

Output:

Code:

```
# Commands to install packages from source editor
install.packages("openxlsx")
install.packages("csvread")
install.packages("XLS")

# Data Acquisition examples
# Loading packages "readxl" and "csvread'
library(readxl)

# Importing excel file to perform operations
SalesData <- read_excel("F:/Pushkar/MCA/Sem-1/DAR/SalesData.xlsx")
View(SalesData)

library(csvread)
# Importing .csv file to perform operations
SalesData1 <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/SalesData1.csv")
SalesData1

# Performing arithmetic operations using R
# Printing variables and objects
# Addition
a <- 5
b <- 10
sum_result <- a + b
```

```
print(sum_result)

# Subtraction
difference <- b - a
print(difference)

# Multiplication
product <- a * b
print(product)

# Division
quotient <- b / a
print(quotient)

# Exponentiation
power <- a^2
print(power)

# Modulus (remainder)
remainder <- b %% a
print(remainder)

# Data Types in R
# Identifying data types using class()
x <- 5.7     # Numeric(Double)
class(x)
y <- 10L     # The 'L' suffix indicates an integer
class(y)
name <- "Alice"     # Character
class(name)

is_valid <- TRUE    #Logical(Boolean)
class(is_valid)
z <- 3 + 2i    # complex
```

```
class(z)
gender <- factor(c("Male", "Female", "Male", "Female"))    #Factor class(gender)
today <- as.Date("2023-08-30")    #Date and Time
class(today)
nums <- c(1, 2, 3, 4, 5)    #Vector
class(vector())
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)    #Matrix
class(mat)
df <- data.frame(name = c("Alice", "Bob"), age = c(25, 30))    #Data Frame
class(df)
my_list <- list(name = "Alice", age = 25, is_student = TRUE)    #List
class(my_list)
```

Console Output:

```
> # Data Acquisition examples
> # Loading packages "readxl" and "csvread'
> library(readxl)
>
> # Importing excel file to perform operations
> SalesData <- read_excel("F:/Pushkar/MCA/Sem-1/DAR/SalesData.xlsx")
> View(SalesData)
>
> library(csvread)
> # Importing .csv file to perform operations
> SalesData1 <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/SalesData1.csv")
> SalesData1
   month_number facecream facewash toothpaste bathingsoap shampoo moisturizer
total_units total_profit
1    1   2500   1500   5200   9200   1200   1500   21100   211000
2    2   2630   1200   5100   6100   2100   1200   18330   183300
3    3   2140   1340   4550   9550   3550   1340   22470   224700
4    4   3400   1130   5870   8870   1870   1130   22270   222700
5    5   3600   1740   4560   7760   1560   1740   20960   209600
>
```

```
> # Performing arithmetic operations using R
> # Printing variables and objects
> # Addition
> a <- 5
> b <- 10
> sum_result <- a + b
> print(sum_result)
[1] 15
>
> # Subtraction
> difference <- b - a
> print(difference)
[1] 5
>
> # Multiplication
> product <- a * b
> print(product)
[1] 50
>
> # Division
> quotient <- b / a
> print(quotient)
[1] 2
>
> # Exponentiation
> power <- a^2
> print(power)
[1] 25
>
> # Modulus (remainder)
> remainder <- b %% a
> print(remainder)
[1] 0
>
```

```
> # Data Types in R
> # Identifying data types using class()
> x <- 5.7     # Numeric(Double)
> class(x)
[1] "numeric"
> y <- 10L     # The 'L' suffix indicates an integer
> class(y)
[1] "integer"
> name <- "Alice"     # Character
> class(name)
[1] "character"
>
> is_valid <- TRUE    #Logical(Boolean)
> class(is_valid)
[1] "logical"
> z <- 3 + 2i    # complex
> class(z)
[1] "complex"
> gender <- factor(c("Male", "Female", "Male", "Female"))    #Factor class(gender)
> today <- as.Date("2023-08-30")    #Date and Time
> class(today)
[1] "Date"
> nums <- c(1, 2, 3, 4, 5)    #Vector
> class(vector())
[1] "logical"
> mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)    #Matrix
> class(mat)
[1] "matrix" "array"
> df <- data.frame(name = c("Alice", "Bob"), age = c(25, 30))    #Data Frame
> class(df)
[1] "data.frame"
> my_list <- list(name = "Alice", age = 25, is_student = TRUE)    #List
> class(my_list)
[1] "list"
```

Conclusion: Successfully performed the commands required to perform the data acquisition, install packages, loading packages, data types, checking the type of variables, print variables and objects like vector, matrix, list, factor, data frame, tables.

Name of Student: Pushkar Sane		
Roll Number: 45	**Lab Assignment Number: 2**	
Title of Lab Assignment: Write commands for following cbind-ing and rbinding, Reading and Writing data. setwd(), getwd(), data(), rm(), Attaching and Detaching data. Reading data from the consol. Loading data from different data sources.(CSV, Excel).		
DOP: 19-09-2023	**DOS: 20-09-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO1, PO2, PO3, PO4, PO6, PO7, PSO1, PSO2**	**Signature:**

Aim: Write commands for following cbind-ing and rbind-ing, Reading and Writing data. setwd(), getwd(), data(), rm(), Attaching and Detaching data. Reading data from the console, Loading data from different data sources.(CSV, Excel).

Description:

1. **Cbind-ing and Rbind-ing:** In R, `cbind` and `rbind` are two functions used for combining data frames or matrices either by columns (`cbind`) or by rows (`rbind`). These functions are essential for data manipulation and aggregation tasks. Let's take a closer look at each of them:
   a. **`cbind` (Column Bind):**
      i. `cbind` stands for "column bind," and it is used to combine objects (usually data frames or matrices) by adding their columns side by side.
      ii. Syntax: `cbind(object1, object2, ...)`
      iii. Each object can be a data frame, matrix, or vector.
      iv. If you provide multiple objects to `cbind`, it will combine them by columns, aligning the columns from each object together
      v. If the objects have different numbers of rows, `cbind` will use recycling to match the shorter object's length to the longer one.
      vi. Example:

```
# Creating two data frames
df1 <- data.frame(Name = c("Alice", "Bob", "Charlie"),
Age = c(25, 30, 22))
df2 <- data.frame(Score = c(95, 88, 75))
# Combining them using cbind
combined <- cbind(df1, df2)
print(combined)
```

Output:

```
   Name     Age   Score
1  Alice    25    95
2  Bob      30    88
3  Charlie  22    75
```

b. `rbind` (Row Bind):

i. `rbind` stands for "row bind," and it is used to combine objects by stacking them on top of each other, creating a new object with more rows.

ii. Syntax: `rbind (object1, object2, ...)`

iii. Each object can be a data frame, matrix, or vector.

iv. If you provide multiple objects to `rbind`, it will combine them by rows, aligning the rows from each object together.

v. Like `cbind`, if the objects have different numbers of columns, `rbind` will use recycling to match the shorter object's width to the longer one.

vi. Example:

```
# Creating two data frames
df1 <- data. frame(Name = c("Alice", "Bob"),
Age = c(25, 30))
df2 <- data.frame(Name = c("Charlie"),
Age = c(22))
# Combining them using rbind
combined <- rbind(df1, df2)
print(combined)
```

Output:

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

vii. In summary, `cbind` and `rbind` are versatile functions in R that allow you to combine data frames or matrices either by columns or by rows, depending on your data manipulation needs.

viii. These functions are commonly used in data preprocessing, data merging, and reshaping tasks.

2. **Reading and Writing Data**

Reading and writing data are fundamental tasks in data analysis using R. R provides various functions and packages to handle different data formats, allowing you to import and export data efficiently. Here's a detailed overview of reading and writing data using R:

Reading Data:

a. **Reading from Text Files:**

i. `read.csv()`, `read.table()`, and `read.delim()` are commonly used functions to read data from text files (e.g., CSV, TSV, plain text).

ii. Example:

```
# Read data from a CSV file
 data <- read.csv("data.csv")
 # Read data from a tab-separated file (TSV)
 data <- read.table("data.tsv", sep = "\t", header = TRUE)
```

iii. You can specify the delimiter and whether the file has a header row using parameters like `sep` and `header`.

b. **Reading from Excel Files:**

i. R has packages like `readxl` and `openxlsx` for reading data from Excel files.

ii. Example:

```
library(readxl)
 # Read data from an Excel file
 data <- read_excel("data.xlsx")
```

c. **Reading from Other Data Formats:**

i. R supports a wide range of data formats. To read from formats like JSON, XML, HDF5, or databases, you'll need specific packages (e.g., `jsonlite`, `XML`, `hdf5r`, or database-specific packages like `DBI`).

ii. Example (using `jsonlite` for JSON):

```
library(jsonlite)
data <- fromJSON("data.json")          # Read data from a JSON file
```

Writing Data:

a. **Writing to Text Files:**
   i. For other data formats (e.g., JSON, XML, HDF5), you'll need specific packages similar to reading.
   ii. Example (using `jsonlite` for JSON):
   ```
   library(jsonlite) # Write data to a JSON file
   toJSON(data, file = "output.json")
   ```

3. **Function (setwd, getwd, data, rm etc):**

In R, several functions and commands are used to manage the working directory, load datasets, and remove objects. Here's a detailed explanation of each of these functions and commands along with examples:

a. **`setwd()` (Set Working Directory):**
   i. `setwd()` is used to set the current working directory to a specified path. The working directory is the folder from which R reads and writes files by default.
   ii. Example:
   ```
   # Set the working directory to a specific path
   setwd("C:/Users/YourUserName/Documents/R_Projects")
   ```

b. **`getwd()` (Get Working Directory):**
   i. `getwd()` is used to retrieve the current working directory.
   ii. Example:
   ```
   # Get the current working directory
   current_dir <- getwd()
   print(current_dir)
   ```

c. **`data()` (Load Datasets):**
   i. `data()` is used to load built-in datasets that come with R packages. These datasets can be used for practice and learning.
   ii. Example:
   ```
   data(iris)      # Load the built-in 'iris' dataset
   head(iris)      # Use the 'iris' dataset in your R session
   ```

d. **`rm()` (Remove Objects):**
   i. `rm()` is used to remove objects (variables, data frames, etc.) from the R workspace. You can specify one or more objects to be removed.
   ii. Example:

```
# Create a variable 'x'
x <- c(1, 2, 3, 4, 5)

# Remove the variable 'x'
rm(x)

# Check if 'x' has been removed
print(exists("x"))
```

You can also remove all objects in the workspace by using `rm(list = ls())`.

```
# Remove all objects in the workspace
rm(list = ls())
```

   iii. These functions and commands are useful for managing your R environment and working with data. Setting the working directory helps you read and write files from the correct location, `data()` makes it easy to access built-in datasets, and `rm()` is essential for cleaning up your workspace by removing unnecessary objects.

4. <u>Attaching and Detaching data:</u>

In R, you can attach and detach datasets to make it easier to work with the variables in those datasets. Attaching a dataset allows you to access its variables directly without specifying the dataset's name each time. Detaching a dataset removes it from the search path.

a. **Attaching Data:**
   i. To attach a dataset in R, you can use the `attach()` function. This function makes the dataset's variables available for direct access without specifying the dataset name.
   ii. Example:

```
# Create a simple dataset
my_data <- data.frame(
```

```
Name = c("Alice", "Bob", "Charlie"),
Age = c(25, 30, 22)
)
# Attach the dataset
attach(my_data)
# Now you can access variables directly
print(Name)
print(Age)
```

b. **Detaching Data:**
   i. To detach a dataset, you can use the `detach()` function. Detaching a dataset removes it from the search path, making its variables inaccessible without specifying the data set's name.
   ii. Example:

```
# Detach the dataset
detach(my_data)
# Attempting to access variables without specifying the dataset will result in an error.
# print(Name) # This will throw an error
```

   iii. Note: It's important to be cautious when using `attach()` and `detach()`. While they can make code more concise, they can also lead to ambiguity and unexpected behavior, especially in larger and more complex scripts. It's often considered good practice to avoid using `attach()` and `detach()` in favor of using the `$` operator or the `with()` function for specific situations.

c. **Using `$` operator:**
   i. You can access variables in a dataset directly using the `$` operator.
   ii. Example:

```
# Create a dataset
my_data <- data.frame(
Name = c("Alice", "Bob", "Charlie"),
Age = c(25, 30, 22))
```

```
# Access variables directly using the $ operator
print(my_data$Name)
print(my_data$Age)
```

d. **Using `with()` Function:**
   i. The `with()` function allows you to temporarily work within a specific environment, making it easier to access variables.
   ii. Example:

```
# Create a dataset
my_data <- data.frame(
 Name = c("Alice", "Bob", "Charlie"),
 Age = c(25, 30, 22)
)
# Use with() to access variables
with(my_data, {
 print(Name)
 print(Age)
})
```

5. **Reading data from the console:**

Reading data from the console in R is a common task when you want to interactively input data while running your script or when you want to read user inputs during the execution of a program. You can use the `readLines()` and `scan()` functions to read data from the console. Here's a detailed explanation of how to do this:

a. **`readLines()` Function:**
   i. The `readLines()` function reads lines of text input from the console and stores them in a character vector. You can specify the number of lines to read or read until an empty line is encountered. It's commonly used for reading multiple lines of text.
   ii. Example:

```
# Read multiple lines of text until an empty line is encountered
input_lines <- readLines(con = "stdin", n = 0L)
```

```
print(input_lines)        # Print the input lines
```

iii. In the above example, `con = "stdin"` specifies that you are reading from the console, and `n = 0L` means to read until an empty line is entered. You can change the value of `n` to read a specific number of lines.

b. `scan()` Function:

i. The `scan()` function is used to read data elements separated by a specified delimiter (default is whitespace) from the console. It's commonly used for reading numeric or character data.

ii. Example:

```
# Read space-separated numeric values from the console
numeric_values <- scan(text = "", what = numeric())
# Print the numeric values
print(numeric_values)
```

iii. In this example, `text = ""` specifies that you are reading from the console, and `what = numeric()` indicates that you want to read numeric values. You can change `what` to `character()` if you want to read character data.

c. Reading Data Line by Line:

i. You can read data line by line by using a loop and `readLines()`. This is useful when you want to read and process multiple lines of input sequentially.

ii. Example:

```
# Initialize an empty vector to store lines of text
lines <- character(0)
# Read lines of text until an empty line is encountered
while (TRUE) {
line <- readLines(con = "stdin", n = 1L)
if (length(line) == 0) break # Exit the loop on an empty line
lines <- c(lines, line)
}
# Print the lines of text
print(lines)
```

Script (Code):

```
# CBinding Vector to DataFrame using Cbind
data_1 <- data.frame(x1 = c(7, 3, 2, 9, 0),        # Column1 of data frame 1
                     x2 = c(4, 4, 1, 1, 8),        # Column2 of data frame 1
                     x3 = c(5, 3, 9, 2, 4))        # Column3 of data frame 1
y1 <- c(9, 8, 7, 6, 5)                             # Create vector
data_new1 <- cbind(data_1, y1)                     # cbind vector to data frame
data_new1

# CBinding 2 dataframes using Cbind
data_2 <- data.frame(z1 = c(1, 5, 9, 4, 0),        # Column 1 of data frame 2
                     z2 = c(0, 9, 8, 1, 6))        # Column 2 of data frame 2
data_new2 <- cbind(data_1, data_2)                 # cbind two data frames in R
data_new2

# R BINDING USING R
# RBINDING VECTOR TO DATAFRAME
x1 <- c(7, 4, 4, 9)                                # Column 1 of data frame 1
x2 <- c(5, 2, 8, 9)                                # Column 2 of data frame 1
x3 <- c(1, 2, 3, 4)                                # Column 3 of data frame 1
data_1 <- data.frame(x1, x2, x3)                   # Create example data frame
vector_1 <- c(9, 8, 7)                             # Create example vector
rbind(data_1, vector_1)                            # rbind vector to data frame

# RBINDING 2 DATAFRAMES
x1 <- c(7, 1)                             # Column 1 of data frame 2
x2 <- c(4, 1)                             # Column 2 of data frame 2
x3 <- c(4, 3)                             # Column 3 of data frame 2
data_2 <- data.frame(x1, x2, x3)          # Create second data frame
rbind(data_1, data_2)                              # rbind two data frames in R

# Reading files using R ReadLines()
con <- file("F:/Pushkar/MCA/Sem-1/DAR/readnew.txt", "r")
```

```
w <- readLines(con)
close(con)
w[2]
w[3]
w[4]

# Writing files using R WriteLines()
sample <- c("Class,Alcohol,Malic acid,Ash","1,14.23,1.71,2.43","1,13.2,1.78,2.14")
writeLines(sample,"F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
a <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
a

# Function in R (Setwd, getwd, data, rm)
setwd("D:/Users/Asus/Desktop/MCA FY/DataAnalytics with R")
getwd()
x <- runif(20)
summary(x)
hist(x)
list.files()                        #Lists all the files in working directory
#q()                                #Quit R. You'll get a prompt to save the
workspace.
ls()
# R program to illustrate
# attach function
# Create example data
txt <- data.frame(
   c1 = c(1, 2, 3, 4, 5),
   c2 = c(6, 7, 8, 9, 0),
   c3 = c(1, 2, 5, 4, 5))

# Try to print c1
c1
# Error: object 'c1' not found
```

```
# attach data
attach(txt)
c1
detach(txt)
c1

# Reading data from console in R
# Taking user input readline()
val <- readline(prompt = "Enter the number: ")

# Printing type of variable
print(paste("Old datatype: ",typeof(val)))

# Converting into integer type
val <- as.integer(val)

# Printing the type of variable
print(paste("New datatype: ",typeof(val)))

# Printing the variable
print(val)

# Reading first input using scan()
cat("Enter the number of rows: ")
nrows <- scan(n = 1, what = integer())

cat("Enter the number of columns: ")
ncols <- scan(n = 1, what = integer())

# Read matrix elements
cat("Enter the matrix elements:\n")
elements <- scan(n = nrows * ncols)
```

```
# Step 3: Reshape into a matrix
matrix_data <- matrix(elements, nrow = nrows, ncol = ncols)
```

Console (Output):

```
> # C BINDING USING R
> # CBinding Vector to DataFrame using Cbind
> data_1 <- data.frame(x1 = c(7, 3, 2, 9, 0),          # Column1 of data frame 1
+                      x2 = c(4, 4, 1, 1, 8),          # Column2 of data frame 1
+                      x3 = c(5, 3, 9, 2, 4))          # Column3 of data frame 1
> y1 <- c(9, 8, 7, 6, 5)                               # Create vector
> data_new1 <- cbind(data_1, y1)                       # cbind vector to data frame
> data_new1
  x1 x2 x3 y1
1  7  4  5  9
2  3  4  3  8
3  2  1  9  7
4  9  1  2  6
5  0  8  4  5
>
> # CBinding 2 dataframes using Cbind
> data_2 <- data.frame(z1 = c(1, 5, 9, 4, 0),          # Column 1 of data frame 2
                  + z2 = c(0, 9, 8, 1, 6))             # Column 2 of data frame 2
> data_new2 <- cbind(data_1, data_2)             # cbind two data frames in R
> data_new2
  x1 x2 x3 z1 z2
1  7  4  5  1  0
2  3  4  3  5  9
3  2  1  9  9  8
4  9  1  2  4  1
5  0  8  4  0  6
>
> # R BINDING USING R
> # RBINDING VECTOR TO DATA FRAME
```

```
> x1 <- c(7, 4, 4, 9)                          # Column 1 of data frame 1
> x2 <- c(5, 2, 8, 9)                          # Column 2 of data frame 1
> x3 <- c(1, 2, 3, 4)                          # Column 3 of data frame 1
> data_1 <- data.frame(x1, x2, x3)             # Create example data frame
> vector_1 <- c(9, 8, 7)                       # Create example vector
> rbind(data_1, vector_1)                      # rbind vector to data frame
  x1 x2 x3
1  7  5  1
2  4  2  2
3  4  8  3
4  9  9  4
5  9  8  7
>
> # RBINDING 2 DATAFRAMES
> x1 <- c(7, 1)                                # Column 1 of data frame 2
> x2 <- c(4, 1)                                # Column 2 of data frame 2
> x3 <- c(4, 3)                                # Column 3 of data frame 2
> data_2 <- data.frame(x1, x2, x3)             # Create second data frame
> rbind(data_1, data_2)                        # rbind two data frames in R
  x1 x2 x3
1  7  5  1
2  4  2  2
3  4  8  3
4  9  9  4
5  7  4  4
6  1  1  3
>
> # Reading files using R ReadLines()
> con <- file("F:/Pushkar/MCA/Sem-1/DAR/readnew.txt", "r")
> w <- readLines(con)
> close(con)
> w[2]
[1] "This is the second line."
```

```
> w[3]
[1] "This is the third line."
> w[4]
[1] "This is the forth line."
>
> # Writing files using R WriteLines()
> sample <- c("Class,Alcohol,Malic acid,Ash","1,14.23,1.71,2.43","1,13.2,1.78,2.14")
> writeLines(sample,"F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
> a <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
> a
  Class Alcohol Malic.acid  Ash
1     1   14.23       1.71 2.43
2     1   13.20       1.78 2.14
>
> # Function in R (Setwd, getwd, data, rm)
> setwd("F:/Pushkar/MCA/Sem-1/DAR")
> getwd()
[1] "F:/Pushkar/MCA/Sem-1/DAR"
> x <- runif(20)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.1463  0.4837  0.6256  0.6125  0.8105  0.9854 
> hist(x)
> list.files()                                   #Lists all the files in working directory
[1]  "Journal"          "Practical  1.R"   "Practical  2.R"   "readnew.txt"      "SalesData.xlsx"
"SalesData1.csv"
[7] "sample.csv"
> # q()                              #Quit R. You'll get a prompt to save the workspace.
> ls()
 [1] "a"           "b"           "con"         "data_1"      "data_2"      "data_new1"  "data_new2"  "df"
"difference"
[10] "elements"   "gender"     "is_valid"   "mat"        "my_list"    "name"       "ncols"      "nrows"
"nums"
```

```
[19]  "power"        "product"     "quotient"    "remainder"   "SalesData"   "SalesData1" "sample"
"sum_result" "today"
[28] "txt"       "val"       "vector_1"   "w"         "x"         "x1"        "x2"        "x3"        "y"
[37] "y1"        "z"
>
> # R program to illustrate
> # Attach function
> # Create example data
> txt <- data.frame(
+     c1 = c(1, 2, 3, 4, 5),
+     c2 = c(6, 7, 8, 9, 0),
+     c3 = c(1, 2, 5, 4, 5))
>
> # Try to print c1
> c1
Error: object 'c1' not found
> # Error: object 'c1' not found
> # Attach data
> attach(txt)
> c1
[1] 1 2 3 4 5
> detach(txt)
> c1
Error: object 'c1' not found
> # Reading data from console in R
> # Taking user input readline()
> val <- readline(prompt = "Enter the number: ")
Enter the number: 5
> # Printing type of variable
> print(paste("Old datatype: ",typeof(val)))
[1] "Old datatype:  character"
>
> # Converting into integer type
```

```
> val <- as.integer(val)
>
> # Printing the type of variable
> print(paste("New datatype: ",typeof(val)))
[1] "New datatype:  integer"
>
> # Printing the variable
> print(val)
[1] 5
> # Reading first input using scan()
> cat("Enter the number of rows: ")
Enter the number of rows: > nrows <- scan(n = 1, what = integer())
1: 3
Read 1 item
> cat("Enter the number of columns: ")
Enter the number of columns: > ncols <- scan(n = 1, what = integer())
1: 3
Read 1 item
> # Read matrix elements
> cat("Enter the matrix elements:\n")
Enter the matrix elements:
> elements <- scan(n = nrows * ncols)
1: 2
2: 3
3: 4
4: 5
5: 6
6: 7
7: 8
8: 9
9: 2
Read 9 items
> # Step 4: Display the matrix
```

```
> print(matrix_data)
     [,1] [,2] [,3]
[1,]    2    5   89
[2,]    3    6    9
[3,]    4    7    8
>
```

Conclusion: In this practical we learned different commands and functions. These commands and functions are fundamental for data analysis and manipulation in R. Properly managing the working directory, loading data, writing results, and handling data objects are crucial skills for any R user which we learned during the practical.

<table>
<tr><td colspan="3">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td>Lab Assignment Number: 3</td></tr>
<tr><td colspan="3">Title of Lab Assignment:
Write commands for Implementation of Data Preprocessing Techniques like:
a. Naming and renaming variables,
b. Adding a new variable,
c. Dealing with missing data,
d. Dealing with categorical data,
e. Data reduction using subsetting</td></tr>
<tr><td colspan="2">DOP: 17-09-2023</td><td>DOS: 28-09-2023</td></tr>
<tr><td>CO Mapped:
CO2</td><td>PO Mapped:
PO1, PO2, PO3, PO4, PO5, PO7, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 3

Aim: Write commands for Implementation of Data Preprocessing Techniques like:

a. Naming and renaming variables,
b. Adding a new variable,
c. Dealing with missing data,
d. Dealing with categorical data,
e. Data reduction using subsetting.

Theory:

Data preprocessing is an important stage in the data analysis pipeline that involves cleaning, organizing, and transforming raw data into an analysis and modeling-ready state. It is crucial in assuring the accuracy and effectiveness of data-driven processes. In this section, we'll go through several data preprocessing techniques in R, such as variable naming and renaming, variable addition, dealing with missing data, handling categorical data, and data reduction using subsetting.

1. **Naming and Renaming Variables:**
   a. **Naming Variables:** Data naming entails giving variables meaningful names that effectively reflect their purpose and substance. Descriptive variable names aid in understanding the context of the data.
   b. **Renaming Variables:** To offer more descriptive and interpretable names, you can rename variables in R using functions such as colnames() or names().
   c. **Example:**
      # Renaming variables in R
      colnames(dataframe) <- c("new_name1", "new_name2", ...)

2. **Adding a New Variable:**
   a. Adding a new variable to your dataset is a critical step in data preparation and feature engineering. This technique enables you to add new information or change current data in order to improve the quality and usefulness of your dataset for analysis or modeling.
   b. Example: # Creating a new variable based on existing ones
      dataframe$new_variable <- dataframe$column1 + dataframe$column2

3. **Dealing with Missing Data:**
    a. **Identification of Missing Data:** Start by identifying missing values in your dataset using functions like is.na() or complete.cases() in R.
    Example:
    ```
    # Identify missing values in a column
    missing_values <- is.na(dataframe$column_with_missing)
    ```

    b. **Handling Missing Data:**
    1) <u>Removal:</u> You can remove rows with missing values using complete.cases()
    Example:
    ```
    dataframe <- dataframe[complete.cases(dataframe), ]
    ```
    2) <u>Imputation:</u> Fill in missing values with appropriate substitutes. Common imputation methods include mean, median, mode imputation, or more advanced techniques like K-nearest neighbors (KNN) imputation or regression imputation.
    Example:
    ```
    # Impute missing values with the mean of the column
    dataframe$column_with_missing <- ifelse
    (is.na(dataframe$column_with_missing),
    mean(dataframe$column_with_missing, na.rm = TRUE),
    dataframe$column_with_missing)
    ```

4. **Dealing with Categorical Data:**
    1) **Categorical Data Encoding:** Categorical data needs to be converted into numerical format for most machine learning algorithms. Common methods include:
    a) One-Hot Encoding: Create binary columns for each category.
    Example:
    ```
    library(dummies)
    dataframe <- dummy.data.frame(dataframe, names =
    c("categorical_column"))
    ```

2) **Label Encoding:** Assign unique integers to each category.
   Example:
   dataframe$categorical_column <- as.factor(dataframe$categorical_column)
   dataframe$categorical_column <- as.integer(dataframe$categorical_column)

5. Data Reduction Using Subsetting in R:

1) **Data Subsetting:** Data reduction involves selecting a subset of relevant data for analysis. In R, you can use subsetting to filter rows and columns based on specific conditions.
   Example:
   # Selecting rows where a condition is met
   subset_data <- dataframe[dataframe$column > 5,]
   # Selecting specific columns
   subset_data <- dataframe[, c("column1", "column2")]

2) **Advanced Subsetting Techniques:** You can combine multiple conditions, use logical operators (AND, OR), and even create complex filtering conditions.
   Example:
   subset_data <- dataframe[dataframe$column1 > 5 & dataframe$column2 == "CategoryA",]

<u>Code:</u>

1. **Naming and Renaming Variables**

   To Rename Variables:

```
# Create a sample data frame
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, 73)
)
```

   a. **Rename a Single Variable:**

```
names(data_frame)[names(data_frame) == "Points"] <- "Ranks"
data_frame
```

   Output:

```
> data_frame <- data.frame(
+   Name = c("John", "Sam", "Tom"),
+   Marks = c(20, 25, 23),
+   Points = c(80, 87, 73))
> names(data_frame)[names(data_frame) == "Points"] <- "Ranks"
> data_frame
  Name Marks Ranks
1 John    20    80
2  Sam    25    87
3  Tom    23    73
> 
```

   b. **Rename multiple variables:**

```
data_frame <- data_frame %>%
  rename(Score = Marks,
         rank = Points)
data_frame
```

   Output:

```
> data_frame
  Name Score Ranks
1 John    20    80
2  Sam    25    87
3  Tom    23    73
> 
```

c. **To Change Variable Names (without altering the data):**

names(data_frame) <- c("Name", "Marks Scored", "Rank Secured")

data_frame

Output:

```
> data_frame
  Name Marks Scored Rank Secured
1 John           20           80
2  Sam           25           87
3  Tom           23           73
>
```

2. **Adding a New Variable**

a. **Using $ operator**

data_frame$Gender <- c("M","M","M")

data_frame

Output:

```
> data_frame
  Name Marks Scored Rank Secured Gender
1 John           20           80      M
2  Sam           25           87      M
3  Tom           23           73      M
>
```

b. **Using [ ] notation:**

data_frame['Gender'] <- c("M","M","M")

data_frame

Output:

```
> data_frame
  Name Marks Scored Rank Secured Gender
1 John           20           80      M
2  Sam           25           87      M
3  Tom           23           73      M
>
```

c. Using cbind() Function:

Gender <- c ("M","M","M")

data_frame <- cbind(data_frame, Gender)

data_frame

Output:

```
> data_frame
  Name Marks Scored Rank Secured Gender
1 John           20           80      M
2  Sam           25           87      M
3  Tom           23           73      M
>
```

d. Add a New Column from the Existing:

data_frame$rank <- data_frame$Points-2

data_frame

Output:

```
> data_frame
  Name Marks Points rank
1 John    20     80   78
2  Sam    25     87   85
3  Tom    23     73   71
>
```

3. Dealing with missing data

a. Checking for Missing Data:

1) **Checking with not missing any data**

data_frame <- data.frame(

Name = c("John", "Sam", "Tom"),

Marks = c(20, 25, 23),

Points = c(80, 87, 73)

)

any(is.na(data_frame))

Output:

```
> any(is.na(data_frame))
[1] FALSE
>
```

2) **Checking with missing data**

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA)
)
any(is.na(data_frame))
```

Output:

```
> any(is.na(data_frame))
[1] TRUE
>
```

b. Handling Missing Data:

1) **Removing Row With missing Values**

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA),
  Favourite = c("Messi", "Ronaldo", "Messi")
)
# Remove rows with missing values
data_frame <- na.omit(data_frame)
data_frame
```

Output:

```
> data_frame
  Name Marks Points Favourite
1 John    20     80     Messi
2  Sam    25     87   Ronaldo
>
```

2) **Fill missing values with a specific value**

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA),
)
# Fill missing values with a specific value
data_frame$Points[is.na(data_frame$Points)] <- 100
data_frame
```

Output:

```
> data_frame
  Name Marks Points
1 John    20     80
2  Sam    25     87
3  Tom    23    100
>
```

4. Dealing with Categorical Data

a. Converting Categorical to Dummy Variables (One-Hot Encoding):

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA),
  Favourite = c("Messi", "Ronaldo", "Messi")
)
# Perform one-hot encoding (dummy variable creation) for the categorical variable
dummyData <- dummyVars(~ Favourite, data = data_frame)
data_frame <- data.frame(predict(dummyData, newdata = data_frame))
data_frame
```

Output:

```
> data_frame
  FavouriteMessi FavouriteRonaldo
1              1                0
2              0                1
3              1                0
```

b. Converting Categorical to Numeric Using Factorization:

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA),
  Favourite = c(1, 2, 3)
)
# Perform one-hot encoding (dummy variable creation) for the categorical variable
data_frame$Favourite <- as.numeric(factor(data_frame$Favourite))
data_frame
```

Output:

```
> data_frame
  Name Marks Points Favourite
1 John    20     80         1
2  Sam    25     87         2
3  Tom    23     NA         3
>
```

5. Data Reduction Using Subsetting:

a. Subsetting Rows Based on a Condition:

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA)
)
```

```
# Define the condition and desired value
variable_condition <- "Name"  # Change this to your desired condition
desired_value <- "John"      # Change this to your desired value

# Subset the data frame based on the condition
subset_data <- data_frame[data_frame[, variable_condition] == desired_value, ]
subset_data
```

Output:

```
> subset_data
  Name Marks Points
1 John    20     80
> |
```

b. Subsetting Columns:

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA)
)

# Subset data_frame to include only "Age" and "Score"
subset_data <- data_frame[, c("Marks", "Points")]
subset_data
```

Output:

```
> subset_data
  Marks Points
1    20     80
2    25     87
3    23     NA
> |
```

c. Random Sampling:

```
data_frame <- data.frame(
  Name = c("John", "Sam", "Tom"),
  Marks = c(20, 25, 23),
  Points = c(80, 87, NA)
)

# Specify the desired sample size (e.g., 2 for a small sample)
sample_size <- 2
random_sample <- data_frame[sample(nrow(data_frame), sample_size), ]
random_sample
```

Output:

```
> random_sample
  Name Marks Points
3  Tom    23     NA
1 John    20     80
> 
```

<u>Conclusion:</u>

Data preprocessing is a multidimensional process that includes a variety of approaches and procedures for cleaning, enhancing, and preparing data for analysis or modeling. Handling variable names correctly, introducing new variables, addressing missing data, encoding categorical data, and decreasing data via subsetting are all key phases in the data preprocessing pipeline, ensuring the data is in an optimal shape for meaningful analysis or machine learning activities.

<table>
<tr><td colspan="3">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td>Lab Assignment Number: 4</td></tr>
<tr><td colspan="3">Title of Lab Assignment: Demonstrate data reduction and manipulation techniques.</td></tr>
<tr><td colspan="2">DOP: 13/10/2023</td><td>DOS: 13/10/2023</td></tr>
<tr><td>CO Mapped:
CO3</td><td>PO Mapped:
PO4, PO5, PO7, PO8, PO9, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 4

Aim: Implementation of Data reduction using subsetting, implementation and usage Dplyr & Tidyverse, select, transmute, arrange, filter, group-by on dataset.

Description:

Data reduction is a crucial step in data analysis, where you reduce the size or complexity of a dataset to focus on specific subsets of data, relevant variables, or to prepare data for further analysis. The `dplyr` and `tidyverse` packages in R offer powerful tools for data reduction, including functions like `select`, `transmute`, `arrange`, `filter`, and `group_by`.

Here's a breakdown of the mentioned functions in the context of data reduction:

1. **`select`:** This function allows you to choose specific columns from your dataset, thereby reducing the number of variables under consideration. By selecting only the columns of interest, you can make your dataset more manageable and relevant for your analysis.
2. **`transmute`:** While `select` allows you to choose columns, `transmute` is used to create new variables based on existing ones. You can compute new variables or transformations, which can be useful for summarization or further analysis. This can also help in reducing the dataset's dimensionality.
3. **`arrange`:** Sometimes it's necessary to reorder rows within your dataset, perhaps to examine data in a particular order. `arrange` is used to sort rows based on one or more variables, making it easier to visualize or analyze data that's in a specific order.
4. **`filter`:** Data reduction often involves filtering out rows that don't meet specific criteria. With `filter`, you can extract a subset of your data that fits a particular condition, reducing the dataset to only the relevant observations.
5. **`group_by`:** In some cases, data reduction is about aggregating data based on specific variables. `group_by` is used to create groups within your data based on a variable, allowing you to perform summary operations on these groups.

When performing data reduction, you're essentially focusing on specific aspects of your dataset that are relevant to your analysis objectives, thereby making your analysis more efficient and meaningful. Here's a summary of the steps involved in data reduction using these functions:

1. **Data Preparation:** Load your dataset and ensure it's in a suitable format for analysis.

2. **Select Relevant Columns:** Use `select` to choose the variables that are pertinent to your analysis. This reduces the dimensionality of your dataset.
3. **Create New Variables:** If necessary, use `transmute` to compute new variables or transformations that might be helpful for your analysis.
4. **Filter Data:** Use `filter` to subset your data based on specific conditions or criteria. This reduces the dataset to only the relevant observations.
5. **Arrange Data:** If the order of data is essential, use `arrange` to sort the data based on one or more variables.
6. **Group and Summarize Data:** If you need to aggregate your data, use `group_by` in combination with summarization functions to compute summary statistics for each group.

Code (Script):

```
# Load required libraries
library(dplyr)

# Load the mtcars dataset
data(mtcars)

# View the first few rows of the dataset
head(mtcars)

# Select specific columns
selected_columns <- mtcars %>%
  select(mpg, hp, wt)

# Filter rows based on conditions
filtered_data <- mtcars %>%
  filter(cyl == 6, gear == 4)

# Arrange rows based on a variable
arranged_data <- mtcars %>%
  arrange(desc(mpg))
```

```
# Group the data by a variable and summarize it
grouped_and_summarized <- mtcars %>%
  group_by(cyl) %>%
  summarize(mean_mpg = mean(mpg), mean_hp = mean(hp))

# Create new variables using transmute
transmuted_data <- mtcars %>%
  transmute(mpg_per_hp = mpg / hp, wt_miles_per_gallon = wt / mpg)

# View the resulting datasets
head(selected_columns)
head(filtered_data)
head(arranged_data)
grouped_and_summarized
head(transmuted_data)
```

Output:

```
> # Load the mtcars dataset
> data(mtcars)
>
> # View the first few rows of the dataset
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> # Select specific columns
> selected_columns <- mtcars %>%
+   select(mpg, hp, wt)
>
> # Filter rows based on conditions
> filtered_data <- mtcars %>%
+   filter(cyl == 6, gear == 4)
> # Arrange rows based on a variable
> arranged_data <- mtcars %>%
+   arrange(desc(mpg))
> # Group the data by a variable and summarize it
> grouped_and_summarized <- mtcars %>%
```

```
+  group_by(cyl) %>%
+  summarize(mean_mpg = mean(mpg), mean_hp = mean(hp))
> # Create new variables using transmute
> transmuted_data <- mtcars %>%
+  transmute(mpg_per_hp = mpg / hp, wt_miles_per_gallon = wt / mpg)
> # View the resulting datasets
> head(selected_columns)
                   mpg  hp    wt
Mazda RX4         21.0 110 2.620
Mazda RX4 Wag     21.0 110 2.875
Datsun 710        22.8  93 2.320
Hornet 4 Drive    21.4 110 3.215
Hornet Sportabout 18.7 175 3.440
Valiant           18.1 105 3.460
> head(filtered_data)
               mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Merc 280      19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C     17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
> head(arranged_data)
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
> grouped_and_summarized
# A tibble: 3 × 3
    cyl mean_mpg mean_hp
  <dbl>    <dbl>   <dbl>
1     4     26.7    82.6
2     6     19.7   122. 
3     8     15.1   209. 
> head(transmuted_data)
                  mpg_per_hp wt_miles_per_gallon
Mazda RX4          0.1909091           0.1247619
Mazda RX4 Wag      0.1909091           0.1369048
Datsun 710         0.2451613           0.1017544
Hornet 4 Drive     0.1945455           0.1502336
Hornet Sportabout  0.1068571           0.1839572
Valiant            0.1723810           0.1911602
```

Conclusion:

Here code demonstrates the effective use of dplyr and tidyverse functions for data reduction in R. By employing select, transmute, arrange, filter, and group_by, the code showcases how to reduce dataset size and complexity, focusing on specific data subsets and variables of interest. This data reduction process streamlines data analysis and improves the ability to extract valuable insights from the data.

<table>
<tr><td colspan="4">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td colspan="2">Lab Assignment Number: 5</td></tr>
<tr><td colspan="4">Title of Lab Assignment: Write commands for Working with different types of R Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts.</td></tr>
<tr><td colspan="2">DOP: 13-10-2023</td><td colspan="2">DOS: 19-10-2023</td></tr>
<tr><td>CO Mapped:
CO4</td><td colspan="2">PO Mapped:
PO1, PO2, PO3, PO4, PO5, PO7, PO8, PO9, PO12, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 5

Aim: Write commands for Working with different types of R Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts.

Description:

1. **Histogram:**
    a. Write commands for Working with different types of R Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts
    b. Example:
    ```
    data <- c(22, 30, 35, 40, 42, 45, 50, 55, 60, 65)
    # Create a histogram
    hist(data,
    main = "Histogram Example",
    xlab = "Values",
    col = "blue",
    border = "black",
    breaks = 5) # You can customize the number of bins
    ```
    c. Here,
        - **`data`:** The data you want to create a histogram for.
        - **`main`:** The title of the histogram.
        - **`xlab`:** Label for the x-axis.
        - **`col`:** Color of the bars.
        - **`border`:** Color of the border of the bars.
        - **`breaks`:** Number of bins.

2. **Boxplots:**
    a. Boxplots are used to visualize the distribution and spread of a dataset. You can create a boxplot using the `boxplot()` function:
    b. Example:
    ```
    data <- c(22, 30, 35, 40, 42, 45, 50, 55, 60, 65)
    # Create a boxplot
    boxplot(data,
    ```

```
    main = "Boxplot Example",
    col = "lightblue",
    horizontal = TRUE) # Create a horizontal boxplot
```

c. Here,
   - **`data`:** The data for which you want to create a boxplot.
   - **`main`:** The title of the boxplot.
   - **`col`:** Color of the boxes.
   - **`horizontal`:** Set to `TRUE` for a horizontal boxplot.

3. **Bar Charts:**
   a. Bar charts are used to display categorical data. You can create bar charts using the `barplot()` function or the `ggplot2` package. Here's a basic example using the `barplot()` function:
   b. Example:

```
categories <- c("Category A", "Category B", "Category C")
values <- c(10, 20, 30)
# Create a bar chart
barplot(values,
    names.arg = categories,
    main = "Bar Chart Example",
    col = "green")
```

   c. Here,
      - **`values`:** Numeric values for the bars.
      - **`names.arg`:** Names for the categories.
      - **`main`:** The title of the bar chart.
      - **`col`:** Color of the bars.

4. **Line Graphs:**
   a. Line graphs are used to visualize trends and relationships between data points over time or a continuous variable. You can create line graphs using the `plot()` function.
   b. Example:

```
x <- 1:10
y <- x^2
```

```
# Create a line graph
plot(x, y,
    type = "l", # "l" for lines
    main = "Line Graph Example",
    xlab = "Time",
    ylab = "Value",
    col = "red")
```

c. Here,
    - **`x` and `y`:** The data for the x and y axes.
    - **`type`:** "l" for a line graph.
    - **`main`:** The title of the line graph.
    - **`xlab` and `ylab`:** Labels for the x and y axes.
    - **`col`:** Color of the line.

5. **Scatterplots:**
    a. Scatterplots are used to show relationships between two variables. You can create scatterplots using the `plot()` function.
    b. Example:

```
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 6, 8, 10)
# Create a scatterplot
plot(x, y,
    main = "Scatterplot Example",
    xlab = "X-Axis",
    ylab = "Y-Axis",
    col = "blue")
```

c. Here,
    - **`x` and `y`:** The data for the x and y axes.
    - **`main`:** The title of the scatterplot.
    - **`xlab` and `ylab`:** Labels for the x and y axes.
    - **`col`:** Color of the points.

6. **Pie Charts:**
    a. Pie charts are used to represent parts of a whole. You can create pie charts using the `pie()` function.
    b. Example:
    ```
    data <- c(10, 20, 30)
    labels <- c("Category A", "Category B", "Category C")
    # Create a pie chart
    pie(data,
        labels = labels,
        main = "Pie Chart Example",
        col = rainbow(length(data)))
    ```
    c. Here,
        - **`data`:** A vector of values for each segment.
        - **`labels`:** Labels for each segment.
        - **`main`:** The title of the pie chart.
        - **`col`:** Color palette for the segments.

Code: (Script File)

```
setwd("F:/Pushkar/MCA/Sem-1/DAR")

data <- read.csv("SalesData1.csv")
data

# Create a histogram
hist(data$toothpaste, main = "Histogram", xlab = "X-axis label", col = "blue", border = "black")

# Create a boxplot
boxplot(data$bathingsoap, main = "Boxplot", ylab = "Y-axis label", col = "green")

# Create a bar chart
barplot(table(data$total_units), main = "Bar Chart", xlab = "X-axis label", ylab = "Y-axis label", col = "purple")
```

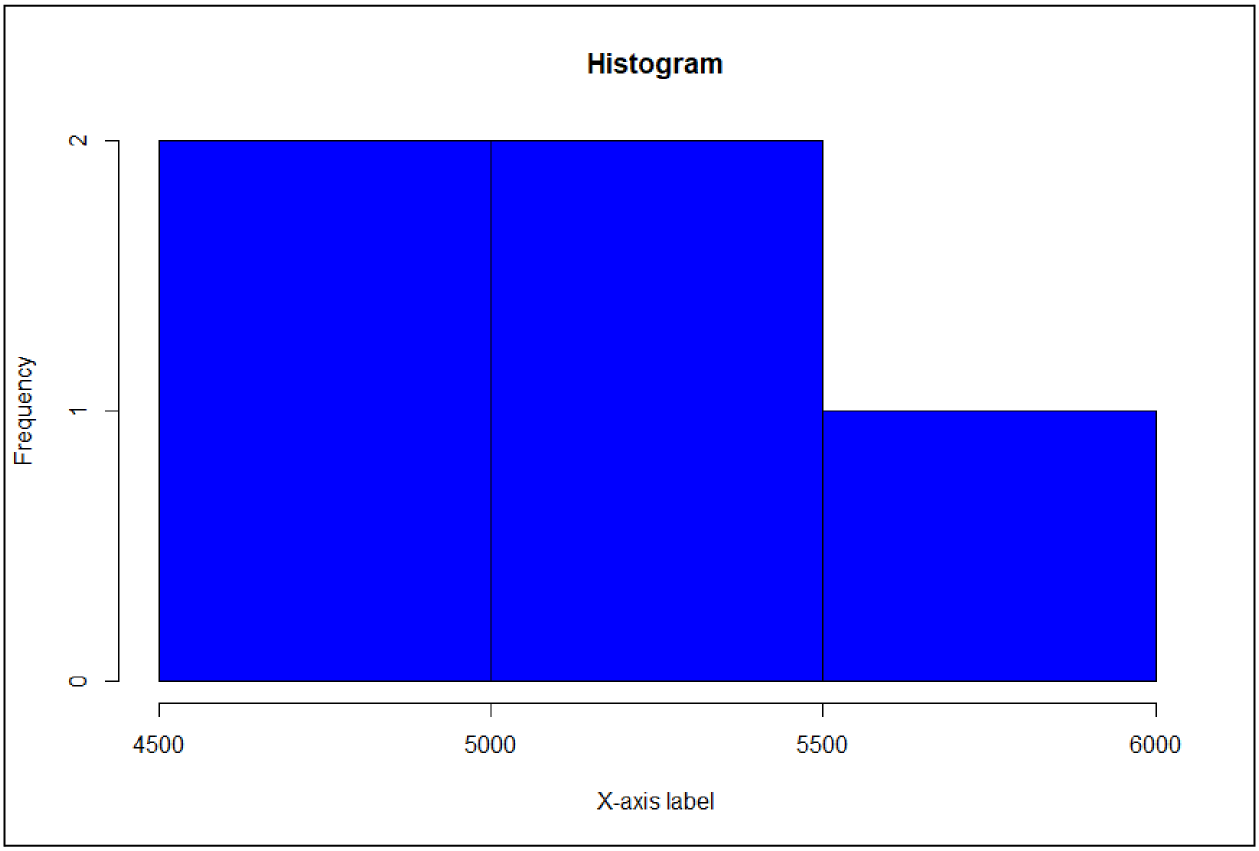
```
# Create a line graph
plot(data$total_units, data$total_profit, type = "l", col = "red", main = "Line Graph", xlab =
"X-axis label", ylab = "Y-axis label")

# Create a scatterplot
plot(data$total_units, data$total_profit, col = "orange", main = "Scatterplot", xlab =
"X-axis label", ylab = "Y-axis label")

# Create a pie chart
slices <- c(30, 10, 20, 15, 25)
lbls <- c("A", "B", "C", "D", "E")
pie(slices, labels = lbls, main = "Pie Chart")
```

Output:

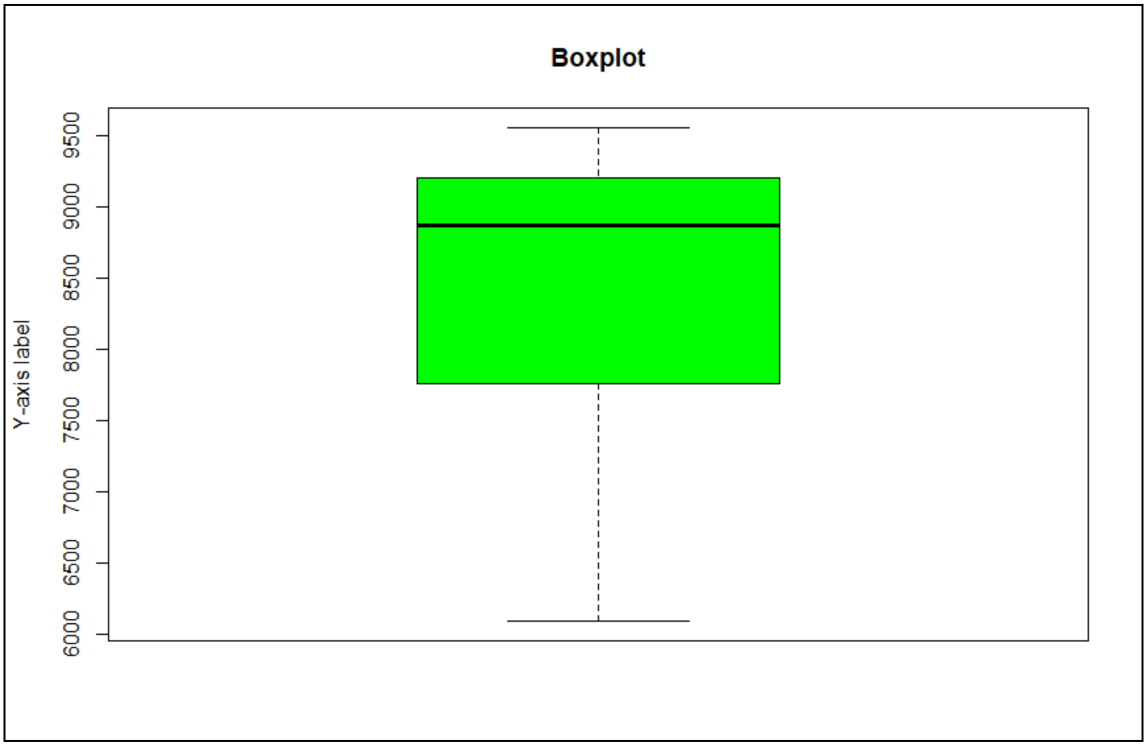
```
> setwd("F:/Pushkar/MCA/Sem-1/DAR")
> data <- read.csv("SalesData1.csv")
> data
  month_number facecream facewash toothpaste bathingsoap shampoo moisturizer
total_units total_profit
1            1      2500     1500       5200        9200    1200        1500
21100        211000
2            2      2630     1200       5100        6100    2100        1200
18330        183300
3            3      2140     1340       4550        9550    3550        1340
22470        224700
4            4      3400     1130       5870        8870    1870        1130
22270        222700
5            5      3600     1740       4560        7760    1560        1740
20960        209600
> # Create a histogram
> hist(data$toothpaste, main = "Histogram", xlab = "X-axis label", col =
"blue", border = "black")
> # Create a boxplot
> boxplot(data$bathingsoap, main = "Boxplot", ylab = "Y-axis label", col =
"green")
> # Create a bar chart
> barplot(table(data$total_units), main = "Bar Chart", xlab = "X-axis label",
ylab = "Y-axis label", col = "purple")
> # Create a line graph
> plot(data$total_units, data$total_profit, type = "l", col = "red", main =
"Line Graph", xlab = "X-axis
```

```
+ label", ylab = "Y-axis label")
> # Create a scatterplot
> plot(data$total_units, data$total_profit, col = "orange", main =
"Scatterplot", xlab = "X-axis label",
+     ylab = "Y-axis label")
> # Create a pie chart
> slices <- c(30, 10, 20, 15, 25)
> lbls <- c("A", "B", "C", "D", "E")
> pie(slices, labels = lbls, main = "Pie Chart")
```

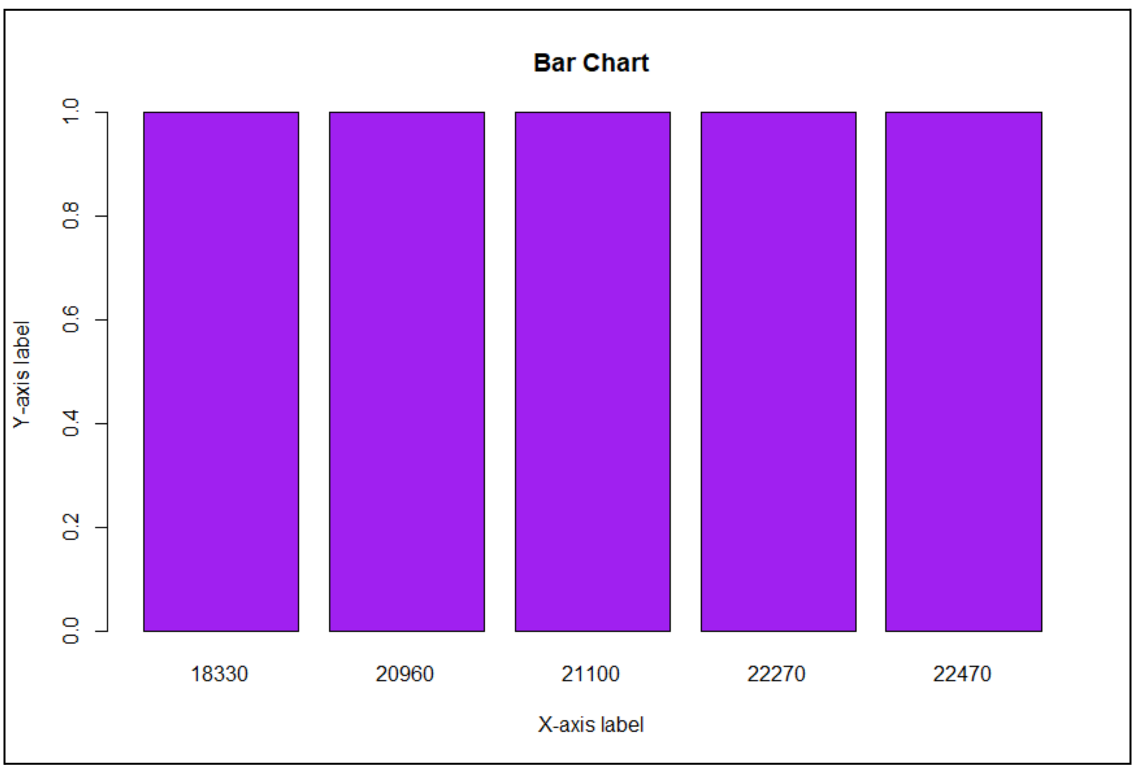
Histogram:



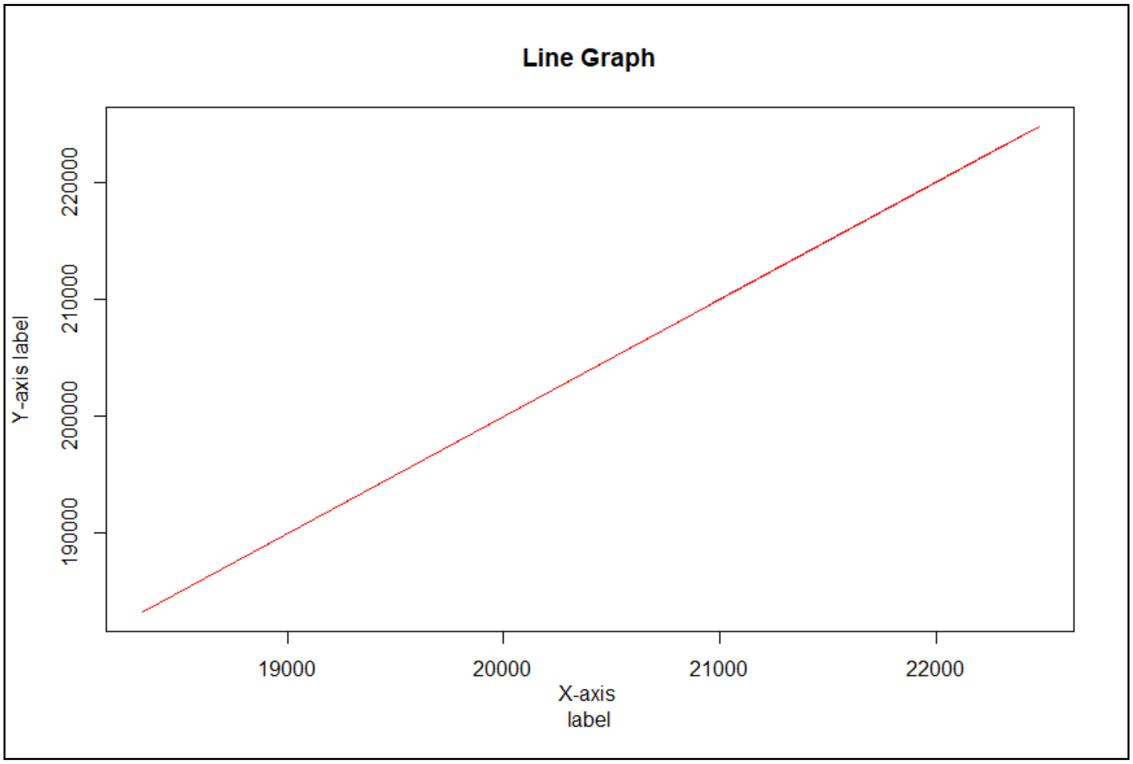
Boxplot:



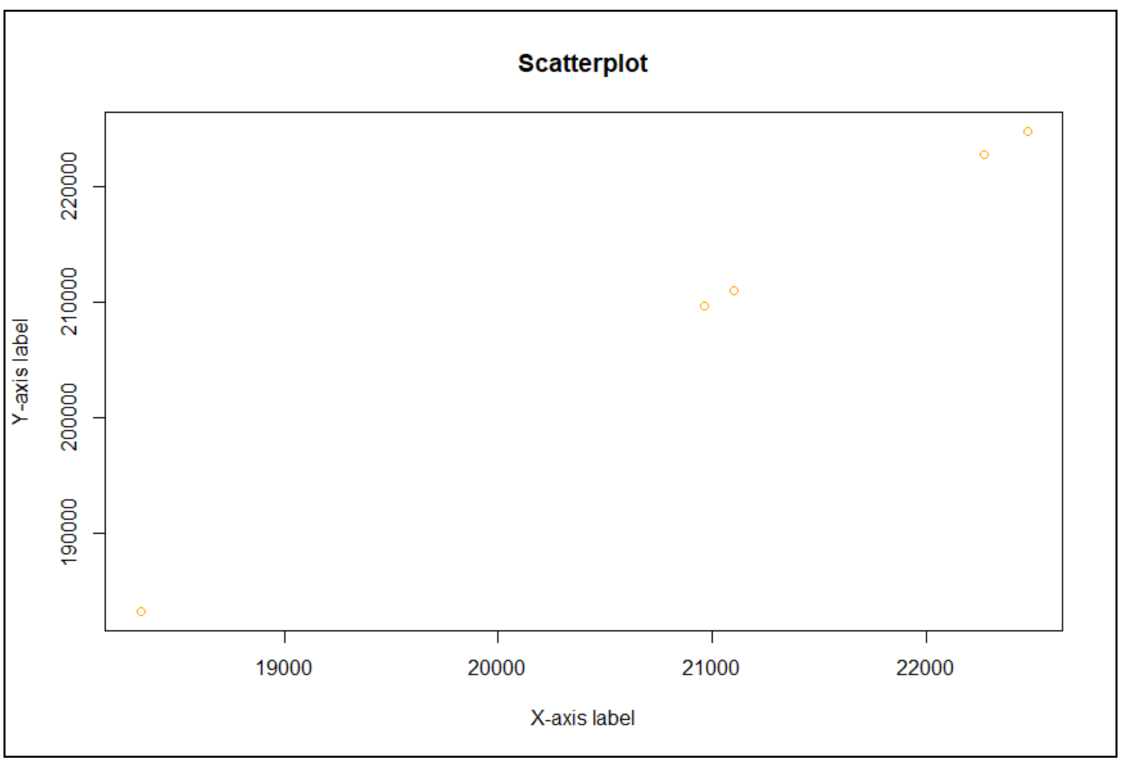
Bar Chart:



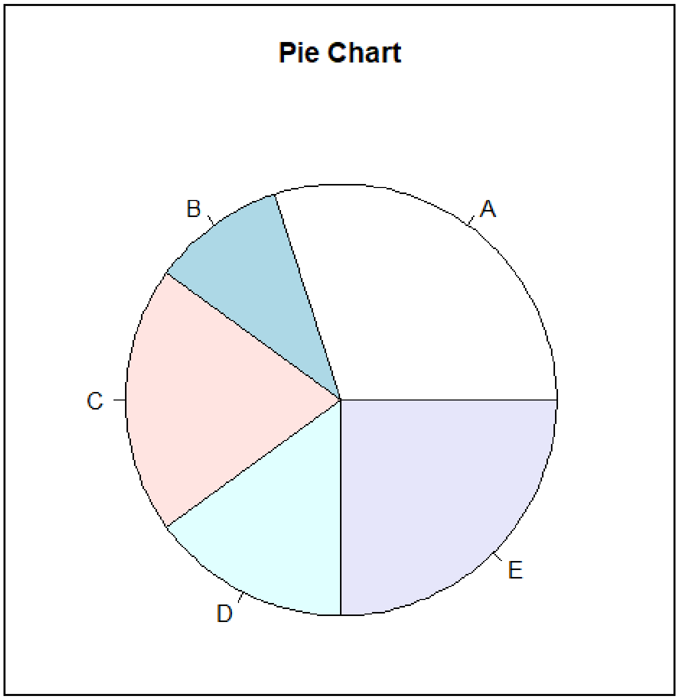
Line Graph:



Scatter Plot:



Pie Chart:



Conclusion:

In this practical, we learned different commands to perform data visualization operation on data using R programming.

<table>
<tr><td colspan="3">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td>Lab Assignment Number: 6</td></tr>
<tr><td colspan="3">Title of Lab Assignment: Implement data Visualization With Ggplot2.</td></tr>
<tr><td colspan="2">DOP: 22-10-2023</td><td>DOS: 27-10-2023</td></tr>
<tr><td>CO Mapped:
CO4</td><td>PO Mapped:
PO1, PO2, PO3, PO4, PO5, PO7, PO8, PO9, PO12, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 6

Aim: Implement data Visualization With Ggplot2.

Description:

Data visualization is a critical aspect of data analysis, allowing you to present complex data in a clear and understandable way. ggplot2 is a popular data visualization package in R that offers a versatile and flexible approach to creating a wide range of plots. This note provides an overview of the process for implementing data visualization with ggplot2.

1. **Load and Install ggplot2:**

   If you haven't already, install and load the ggplot2 package using the following commands:

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

2. **Data Preparation:**

   Start with a well-structured and clean dataset. Ensure your data is organized, and any necessary data wrangling or transformations have been applied. In ggplot2, you typically work with data frames.

3. **Basic ggplot Structure:**

   The fundamental structure of a ggplot2 visualization includes the following components:

   - Data: Specify the data frame that contains your dataset.
   - Aesthetics (aes): Map data variables to visual properties such as x and y coordinates, color, size, and shape.
   - Geometric Objects (Geoms): Choose a geom to determine how the data will be visually represented (e.g., points, lines, bars).
   - Layers: Add layers to the plot, including additional geoms, statistical transformations, facets, themes, and annotations.

4. **Creating a Simple Scatter Plot:**

   To create a basic scatter plot, use the following template:

   ```
   ggplot(data = your_data, aes(x = variable1, y = variable2)) + geom_point()
   ```

- `data`: The data frame.
- `aes`: Aesthetics, mapping variables to visual properties.
- `geom_point()`: Geom for a scatter plot.

5. **Customization:**
   Customize your plot to make it more informative and visually appealing. Use various functions to:
   - Set titles and axis labels with `labs()`.
   - Customize axis scales with `scale_x_()` and `scale_y_()`.
   - Adjust plot themes with `theme()`.
   - Annotate data points with labels using `geom_text()` and `geom_label()`.
   - Add statistical summaries with functions like `geom_smooth()`.

6. **Faceting:**
   Create small multiples or facet your plot using the `facet_wrap()` or `facet_grid()` functions. This is useful when you want to compare subsets of your data in separate panels.

7. **Export Your Plot:**
   Save your plot to a file using the `ggsave()` function.
   For example:
   ggsave("my_plot.png", plot = your_plot, width = 6, height = 4, units = "in")

8. **Advanced Visualizations:**
   ggplot2 supports a wide range of visualizations, including bar charts, line graphs, box plots, and more. Each type of plot requires specific geoms and customization options. You can also use dplyr for advanced data transformations.

9. **Learning Resources:**
   To master ggplot2, consider reading "ggplot2: Elegant Graphics for Data Analysis" by Hadley Wickham. Online tutorials, documentation, and the R community provide valuable resources for learning and troubleshooting.

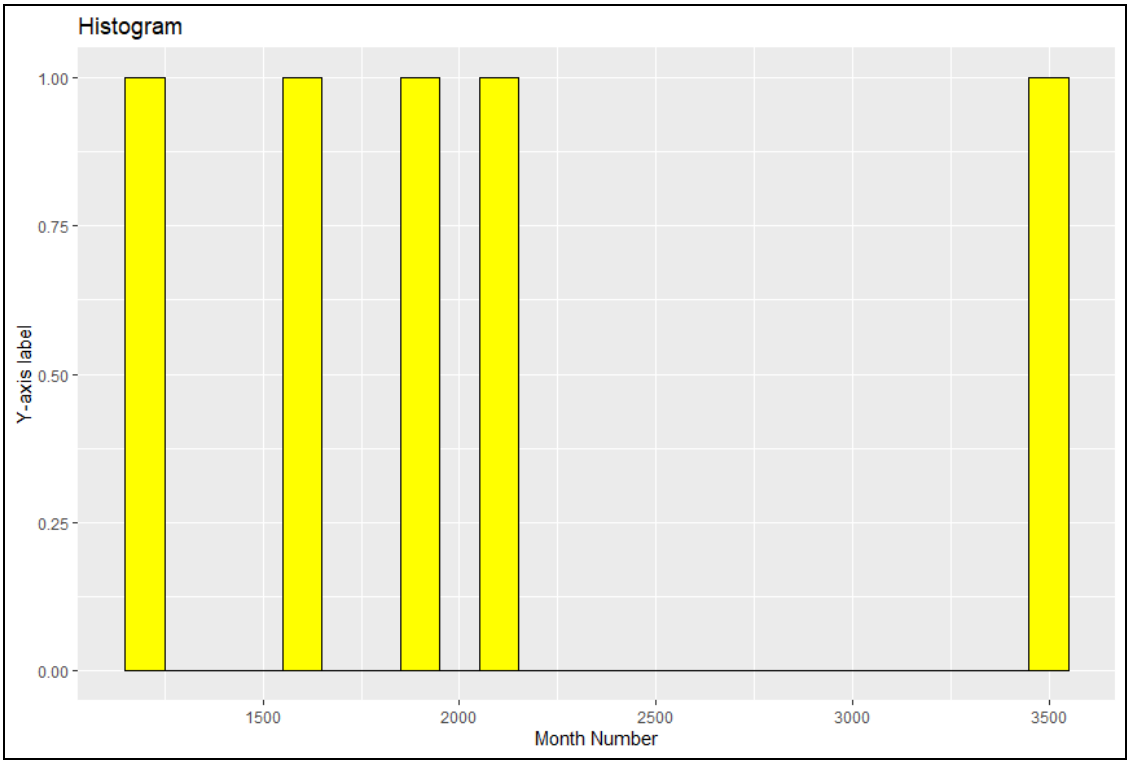
Code:

```
install.packages("ggplot2")  # Install and load necessary packages
library(ggplot2)
data <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/SalesData1.csv")
data
# Create a histogram
ggplot(data, aes(x = shampoo)) +
  geom_histogram(binwidth = 100, fill = "yellow", color = "black") +
  labs(title = "Histogram", x = "Month Number", y = "Y-axis label")

# Create a boxplot
ggplot(data, aes(x = 'Group' , y = facewash)) +
  geom_boxplot(fill = "green", color = "black") +
  labs(title = "Boxplot", x = "Group", y = "Facewash")

# Create a bar chart
ggplot(data, aes(x = shampoo)) +
  geom_bar(fill = "purple", color = "black") +
  labs(title = "Bar Chart", x = "X-Shampoo", y = "Y-axis label")

# Create a line graph
ggplot(data, aes(x = shampoo, y = facewash)) +
  geom_line(color = "red") +
  labs(title = "Line Graph", x = "Shampoo", y = "Facewash")

# Create a scatterplot
ggplot(data, aes(x = shampoo, y = facewash)) +
  geom_point(color = "orange") +
  labs(title = "Scatterplot", x = "Shampoo", y = "Facewash")

# Create a pie chart
pie(data$total_units, labels = data$labels, main = "Pie Chart")
```

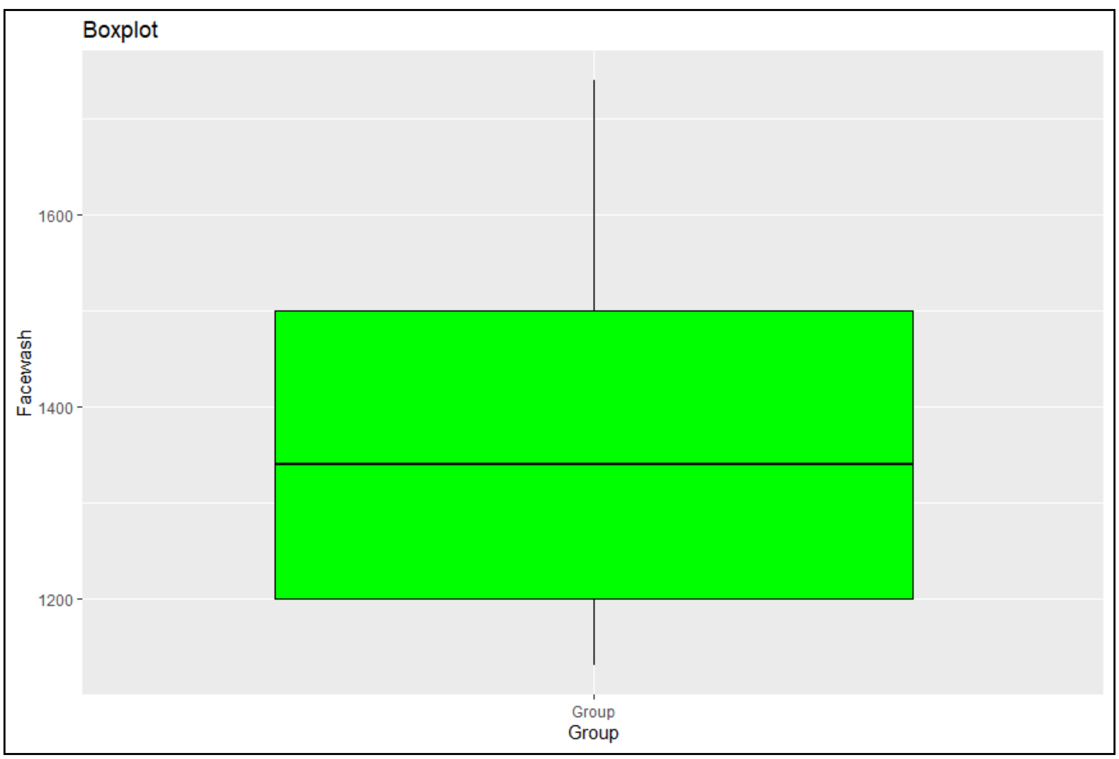
Output;

```
> library(ggplot2)
>
> data <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/SalesData1.csv")
> data
  month_number facecream facewash toothpaste bathingsoap shampoo moisturizer
total_units total_profit
1            1      2500     1500        5200        9200    1200         1500
21100        211000
2            2      2630     1200        5100        6100    2100         1200
18330        183300
3            3      2140     1340        4550        9550    3550         1340
22470        224700
4            4      3400     1130        5870        8870    1870         1130
22270        222700
5            5      3600     1740        4560        7760    1560         1740
20960        209600
> # Create a histogram
> ggplot(data, aes(x = shampoo)) +
+   geom_histogram(binwidth = 100, fill = "yellow", color = "black") +
+   labs(title = "Histogram", x = "Month Number", y = "Y-axis label")
> # Create a boxplot
> ggplot(data, aes(x = 'Group' , y = facewash)) +
+   geom_boxplot(fill = "green", color = "black") +
+   labs(title = "Boxplot", x = "Group", y = "Facewash")
> # Create a bar chart
> ggplot(data, aes(x = shampoo)) +
+   geom_bar(fill = "purple", color = "black") +
+   labs(title = "Bar Chart", x = "X-Shampoo", y = "Y-axis label")
> # Create a line graph
> ggplot(data, aes(x = shampoo, y = facewash)) +
+   geom_line(color = "red") +
+   labs(title = "Line Graph", x = "Shampoo", y = "Facewash")
> # Create a scatterplot
> ggplot(data, aes(x = shampoo, y = facewash)) +
+   geom_point(color = "orange") +
+   labs(title = "Scatterplot", x = "Shampoo", y = "Facewash")
> # Create a pie chart
> pie(data$total_units, labels = data$labels, main = "Pie Chart")
```

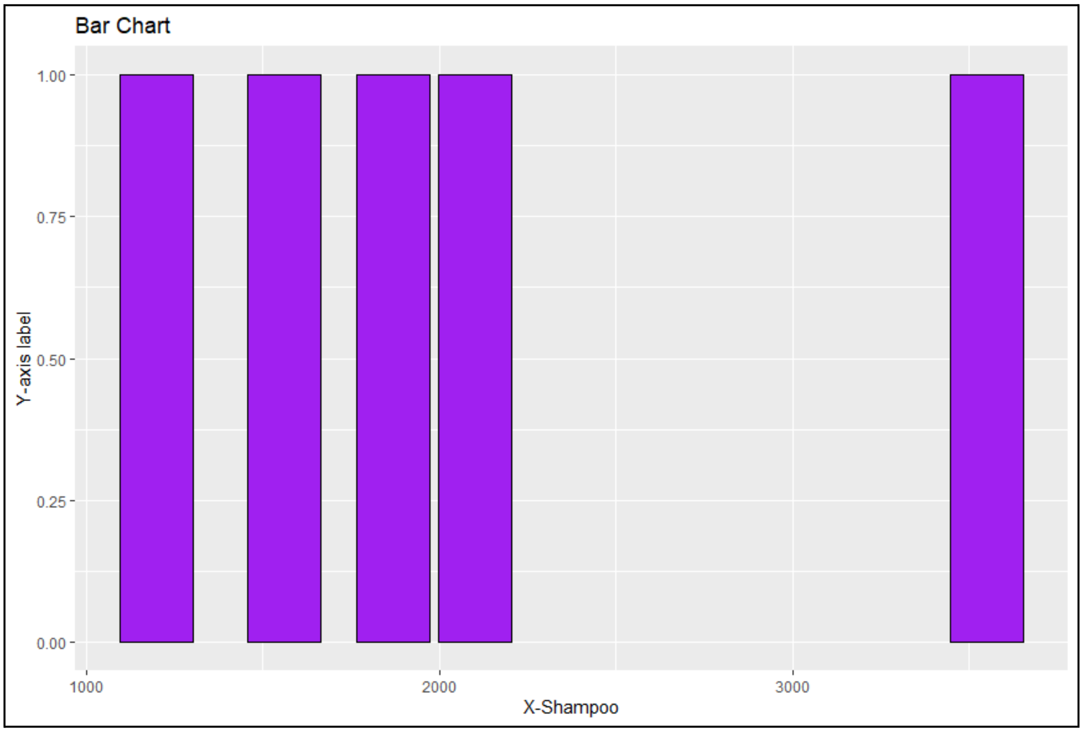
Histogram using Ggplot2:.



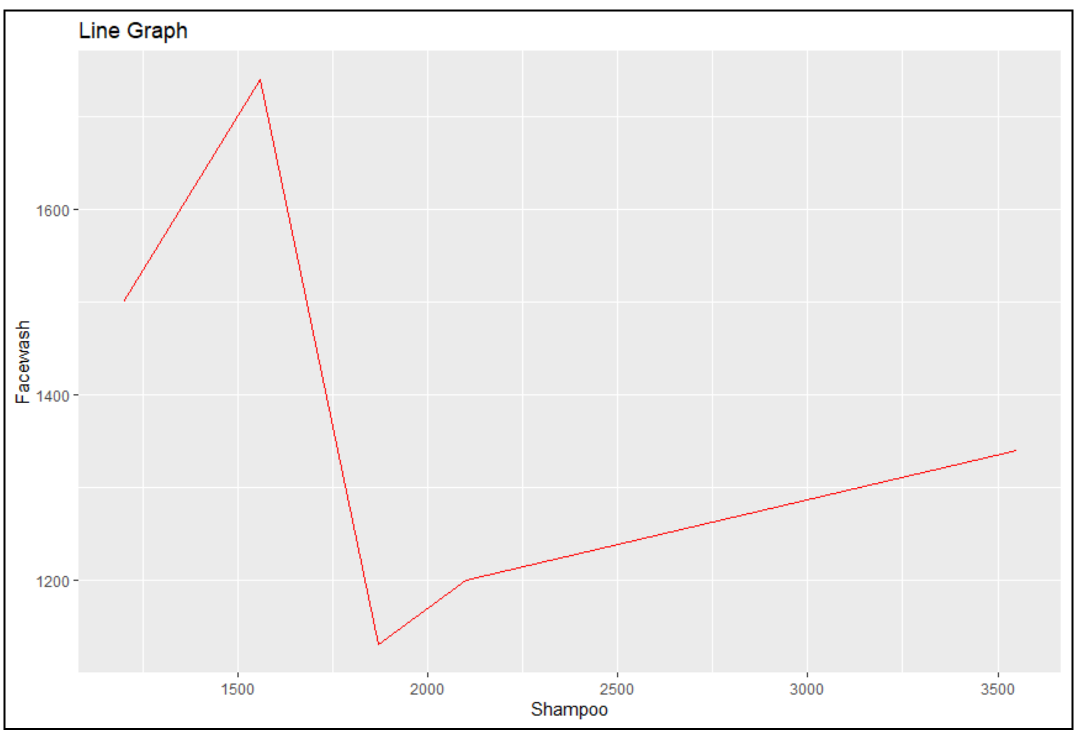
Boxplot using Ggplot2:



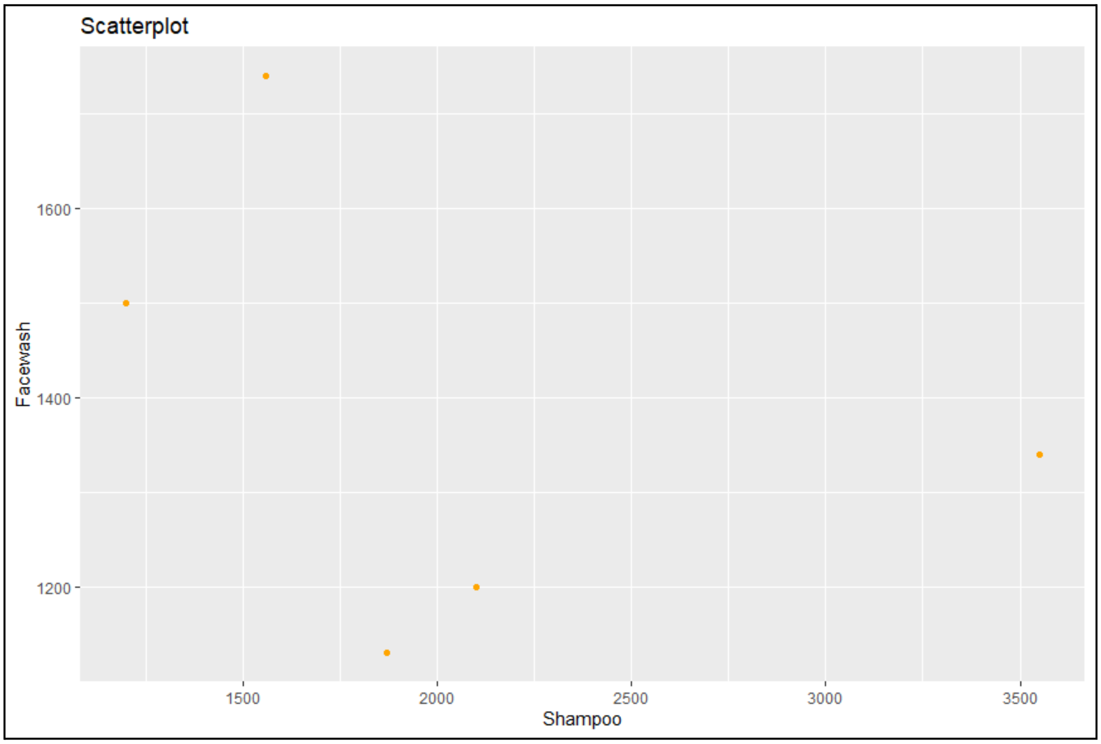
BarChart using Ggplot2:



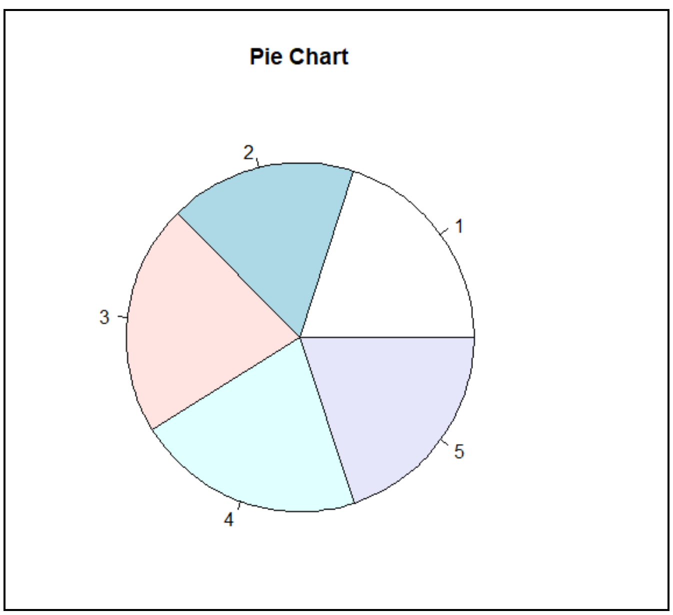
Line Graph using Ggplot2:



ScatterPlot using Ggplot2:



PieChart using Ggplot2:



<u>**Conclusion:**</u> In this practical we learned implementation of data visualization using ggplot2.

<table>
<tr><td colspan="4">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td colspan="2">Lab Assignment Number: 7</td></tr>
<tr><td colspan="4">Title of Lab Assignment: Implement commands for drawing various Correlation Plots and learn the process of EDA.</td></tr>
<tr><td colspan="2">DOP: 14-10-2023</td><td colspan="2">DOS: 20-10-2023</td></tr>
<tr><td>CO Mapped:
CO5</td><td colspan="2">PO Mapped:
PO1, PO2, PO3, PO4, PO5,
PO7, PO12, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 7

Aim: Implement commands for drawing various Correlation Plots and learn the process of EDA.

Description:

Exploratory Data Analysis (EDA) is a crucial step in understanding and visualizing your data to uncover insights and patterns. One of the key components of EDA is exploring the relationships between variables, often visualized through correlation plots.

EDA Process:

1. **Load Required Libraries:**
   a. Before you begin, you need to load the necessary libraries.
   b. Example:
      # Load necessary libraries
      library(ggplot2) For data visualization
      library(corrplot) For correlation plots

2. **Load and Examine Data:**
   a. Load your dataset into R and examine its structure and summary statistics to get an initial understanding of the data.
   b. Example:
      # Load your dataset
      data <- read.csv("your_data.csv")
      # Explore the structure and summary statistics
      str(data)
      summary(data)

3. **Correlation Matrix and Plot:**
   a. Create a correlation matrix to understand the relationships between numeric variables. Then, generate a correlation plot.
   b. Example:
      # Calculate the correlation matrix
      correlation_matrix <- cor(data, method = "pearson")

```
# Create a correlation plot using corrplot
corrplot(correlation_matrix, method = "color")
```

c. The correlation plot will display the strength and direction of correlations using colors.

4. **Scatterplot Matrix:**
   a. To visualize relationships between pairs of numeric variables, create a scatterplot matrix.
   b. Example:
   ```
   # Create a scatterplot matrix using ggplot2
   pairs(data)
   ```
   c. This will produce a matrix of scatterplots showing pairwise relationships between numeric variables.

5. **Heatmap:**
   a. If you want to visualize the relationships between variables, including both numeric and categorical, create a heatmap.
   b. Example:
   ```
   # Create a heatmap using ggplot2
   ggplot(data, aes(x = your_variable1, y = your_variable2, fill =
   your_numeric_variable)) + geom_tile()
           + labs(title = "Heatmap", x = "Variable 1", y = "Variable 2", fill = "Numeric
   Variable")
   ```
   c. Adjust `your_variable1`, `your_variable2`, and `your_numeric_variable` based on your dataset.

6. **Pairwise Scatterplots:**
   a. To create scatterplots between pairs of numeric variables, you can use the `scatterplot` function from the `car` package.
   b. Example:
   ```
   # Install and load the car package
   install.packages("car")
   library(car)
   ```

```
#Create pairwise scatterplots
scatterplotMatrix(data)
```

c. This will display a grid of scatterplots for your numeric variables.

Remember that the actual variable names and data may vary depending on your dataset. The EDA process should be tailored to your specific data and research questions. The goal is to gain insights, detect patterns, and identify potential relationships in your data through various exploratory plots and visualizations.

Code (Script):

```
install.packages("ggplot2")
install.packages("corrplot")
install.packages("GGally")
library(GGally)
library(ggplot2)
library(corrplot)

data <- read.csv("company-sales.csv")
data

# Explore the structure and summary statistics
str(data)
summary(data)

# Calculate the correlation matrix
correlation_matrix <- cor(data, method = "pearson")
correlation_matrix

# Create a correlation plot using corrplot
corrplot(correlation_matrix, method = "color")

# Create a scatterplot matrix using ggplot2
ggpairs(data, title = "Scatterplot Matrix")
```

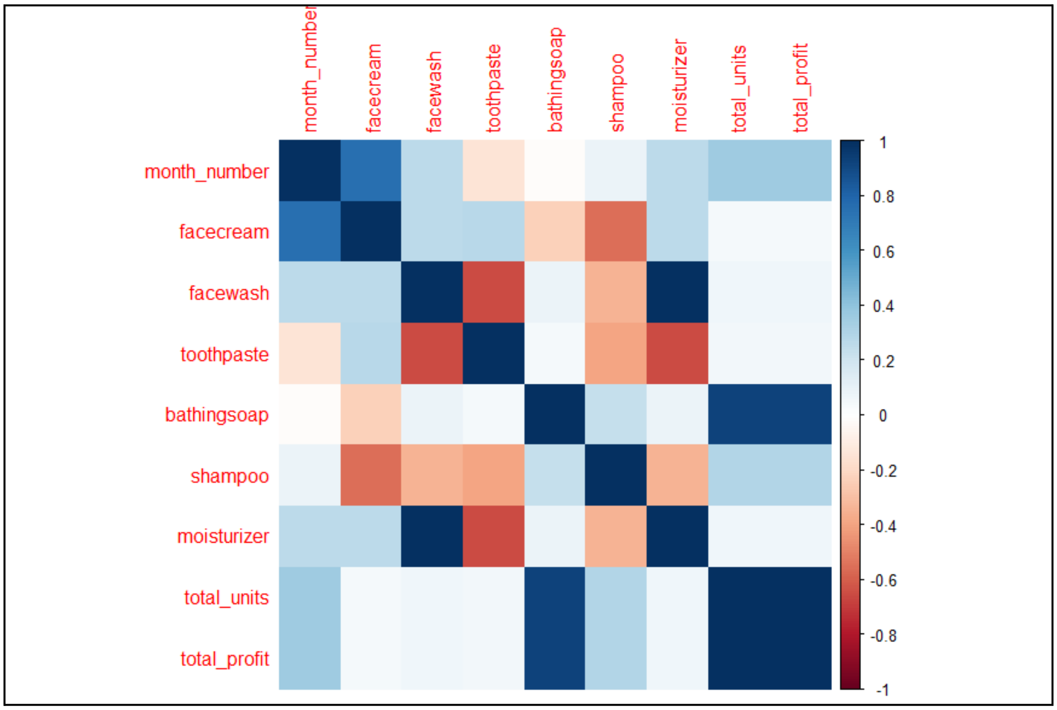
```
# Create boxplots or violin plots for numeric variables
ggplot(data, aes(x = "Group", y = facewash)) + geom_boxplot(fill = "blue") +
  labs(title = "Boxplot or Violin Plot", x = "Categorical Variable", y = "Numeric Variable")

# Create histograms
ggplot(data, aes(x = facewash)) + geom_histogram(binwidth = 40, fill = "blue") +
  labs(title = "Histogram", x = "Numeric Variable")

# Create a heatmap using ggplot2
ggplot(data, aes(x = total_units, y = bathingsoap)) + geom_tile(aes(fill = total_profit)) +
  labs(title = "Heatmap", x = "Categorical Variable 1", y = "Categorical Variable 2", fill =
       "Numeric Variable")
```

Output:

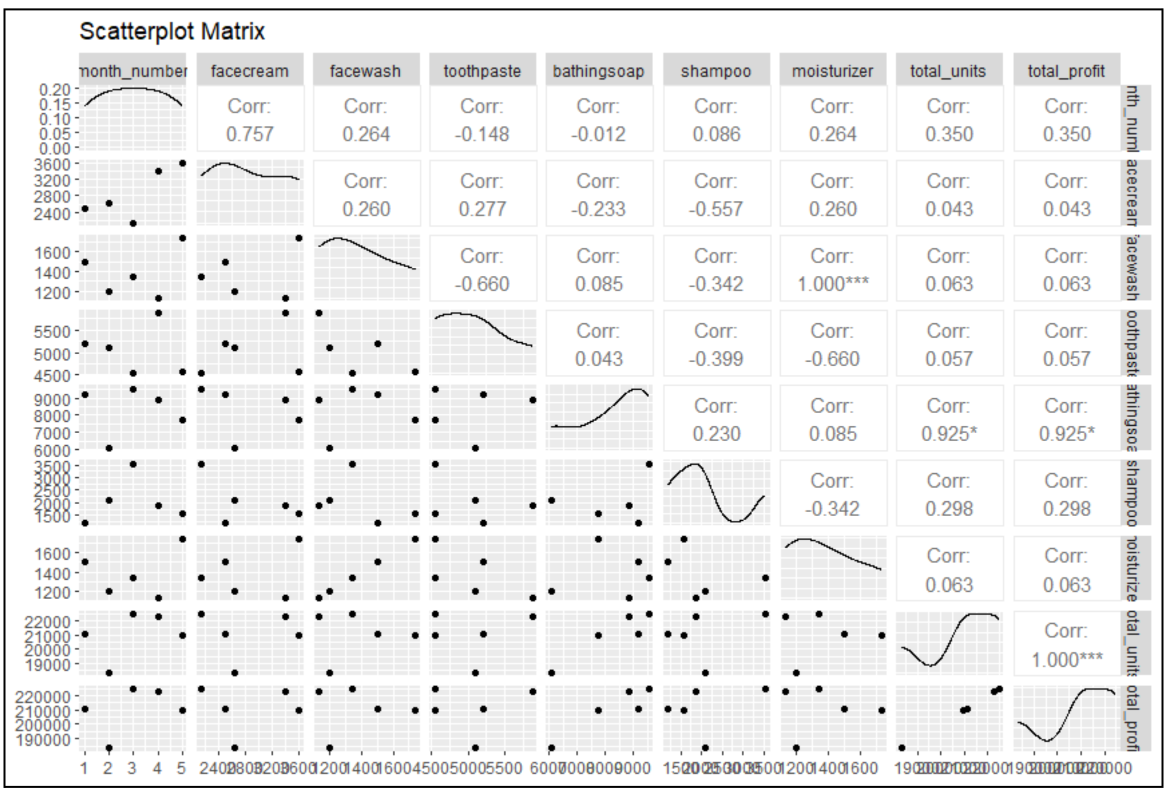
```
> library(GGally)
> library(ggplot2)
> library(corrplot)
> data <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/SalesData1.csv")
> data
  month_number facecream facewash toothpaste bathingsoap shampoo moisturizer
total_units total_profit
1            1      2500     1500       5200        9200    1200        1500
21100       211000
2            2      2630     1200       5100        6100    2100        1200
18330       183300
3            3      2140     1340       4550        9550    3550        1340
22470       224700
4            4      3400     1130       5870        8870    1870        1130
22270       222700
5            5      3600     1740       4560        7760    1560        1740
20960       209600
> # Explore the structure and summary statistics
> str(data)
'data.frame':	5 obs. of  9 variables:
 $ month_number: int  1 2 3 4 5
 $ facecream   : int  2500 2630 2140 3400 3600
 $ facewash    : int  1500 1200 1340 1130 1740
 $ toothpaste  : int  5200 5100 4550 5870 4560
 $ bathingsoap : int  9200 6100 9550 8870 7760
 $ shampoo     : int  1200 2100 3550 1870 1560
 $ moisturizer : int  1500 1200 1340 1130 1740
 $ total_units : int  21100 18330 22470 22270 20960
 $ total_profit: int  211000 183300 224700 222700 209600
> summary(data)
  month_number   facecream        facewash       toothpaste     bathingsoap
shampoo      moisturizer    total_units
 Min.   :1     Min.   :2140    Min.   :1130    Min.   :4550    Min.   :6100
Min.   :1200   Min.   :1130   Min.   :18330
 1st Qu.:2     1st Qu.:2500    1st Qu.:1200    1st Qu.:4560    1st Qu.:7760    1st
Qu.:1560   1st Qu.:1200   1st Qu.:20960
 Median :3     Median :2630    Median :1340    Median :5100    Median :8870
Median :1870   Median :1340   Median :21100
 Mean   :3     Mean   :2854    Mean   :1382    Mean   :5056    Mean   :8296
Mean   :2056   Mean   :1382   Mean   :21026
 3rd Qu.:4     3rd Qu.:3400    3rd Qu.:1500    3rd Qu.:5200    3rd Qu.:9200    3rd
Qu.:2100   3rd Qu.:1500   3rd Qu.:22270
 Max.   :5     Max.   :3600    Max.   :1740    Max.   :5870    Max.   :9550
Max.   :3550   Max.   :1740   Max.   :22470
  total_profit
 Min.   :183300
```

```
 1st Qu.:209600
 Median :211000
 Mean   :210260
 3rd Qu.:222700
 Max.   :224700
> # Calculate the correlation matrix
> correlation_matrix <- cor(data, method = "pearson")
> correlation_matrix
                month_number   facecream    facewash  toothpaste bathingsoap
shampoo moisturizer total_units
month_number     1.00000000  0.75684574  0.26438960 -0.14800837 -0.01243202
0.08598715  0.26438960  0.35038895
facecream        0.75684574  1.00000000  0.26039407  0.27724218 -0.23326099
-0.55680046  0.26039407  0.04310301
facewash         0.26438960  0.26039407  1.00000000 -0.65960883  0.08537187
-0.34226770  1.00000000  0.06274722
toothpaste      -0.14800837  0.27724218 -0.65960883  1.00000000  0.04333450
-0.39860046 -0.65960883  0.05743385
bathingsoap     -0.01243202 -0.23326099  0.08537187  0.04333450  1.00000000
0.23048226  0.08537187  0.92482277
shampoo          0.08598715 -0.55680046 -0.34226770 -0.39860046  0.23048226
1.00000000 -0.34226770  0.29848723
moisturizer      0.26438960  0.26039407  1.00000000 -0.65960883  0.08537187
-0.34226770  1.00000000  0.06274722
total_units      0.35038895  0.04310301  0.06274722  0.05743385  0.92482277
0.29848723  0.06274722  1.00000000
total_profit     0.35038895  0.04310301  0.06274722  0.05743385  0.92482277
0.29848723  0.06274722  1.00000000
             total_profit
month_number   0.35038895
facecream      0.04310301
facewash       0.06274722
toothpaste     0.05743385
bathingsoap    0.92482277
shampoo        0.29848723
moisturizer    0.06274722
total_units    1.00000000
total_profit   1.00000000
```

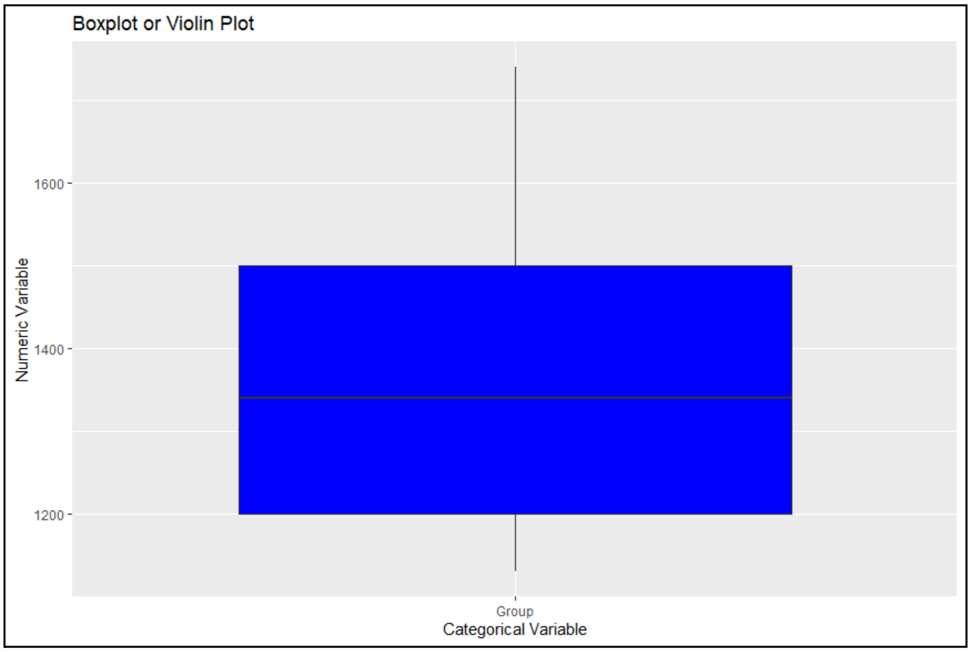
```
> # Create a correlation plot using corrplot
> corrplot(correlation_matrix, method = "color")
```



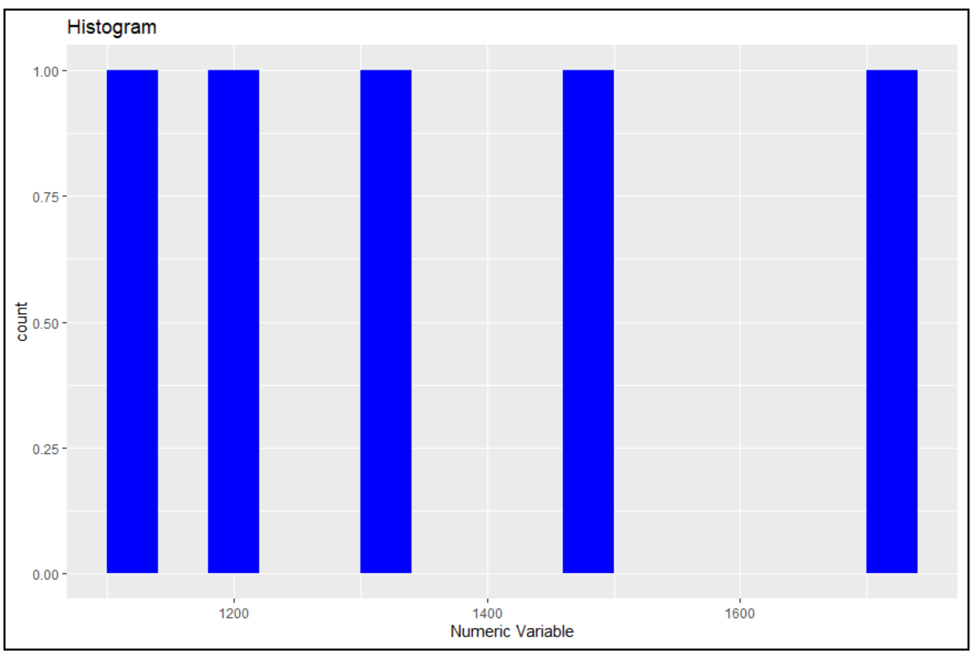
```
> # Create a scatterplot matrix using ggplot2
> ggpairs(data, title = "Scatterplot Matrix")
```



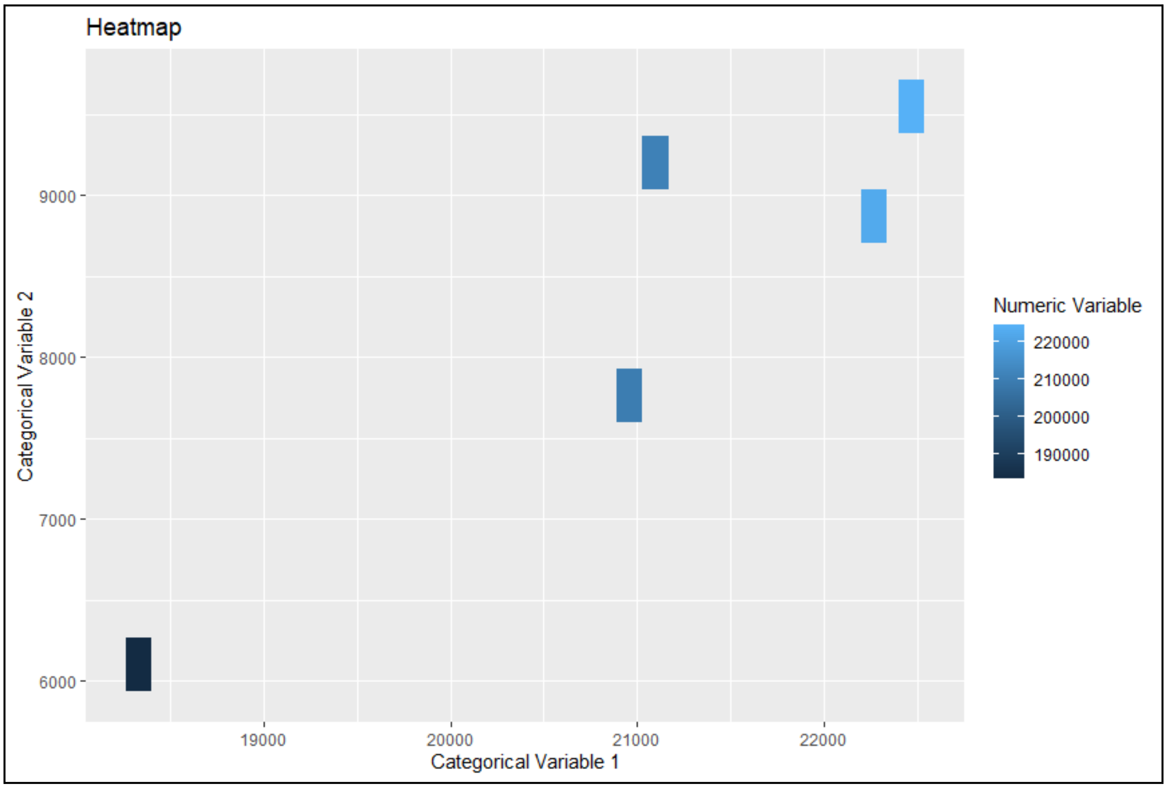
```
> # Create boxplots or violin plots for numeric variables
> ggplot(data, aes(x = "Group", y = facewash)) +
+   geom_boxplot(fill = "blue") +
+   labs(title = "Boxplot or Violin Plot", x = "Categorical Variable", y =
"Numeric Variable")
```




```
> # Create histograms
> ggplot(data, aes(x = facewash)) +
+   geom_histogram(binwidth = 40, fill = "blue") +
+   labs(title = "Histogram", x = "Numeric Variable")
```



```
> # Create a heatmap using ggplot2
> ggplot(data, aes(x = total_units, y = bathingsoap)) +
+  geom_tile(aes(fill = total_profit)) +
+  labs(title = "Heatmap", x = "Categorical Variable 1", y = "Categorical
Variable 2", fill = + "Numeric Variable")
```



Conclusion:

In this practical we learned the EDA(Exploratory Data Analytics) process by which we can gain insights, detect patterns, and identify potential relationships in our data through various exploratory plots and visualizations.

<table>
<tr><td colspan="3">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td>Lab Assignment Number: 8</td></tr>
<tr><td colspan="3">Title of Lab Assignment: Implementation of Normal and Binomial Distribution, Univariate and Bivariate analysis.</td></tr>
<tr><td colspan="2">DOP: 26-10-2023</td><td>DOS: 27-10-2023</td></tr>
<tr><td>CO Mapped:
CO6</td><td>PO Mapped:
PO1, PO2, PO3, PO4, PO5, PO7, PO12, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 8

Aim: Implementation of Normal and Binomial Distribution, Univariate and Bivariate analysis.

Description:

1. **Normal distribution:**

   Normal distribution, also known as the Gaussian distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean. In graphical form, the normal distribution appears as a "bell curve".

   It is generally observed that data distribution is normal when there is a random collection of data from independent sources. The graph produced after plotting the value of the variable on the x-axis and the count of the value on the y-axis is a bell-shaped curve graph. The graph signifies that the peak point is the mean of the data set and half of the values of the data set lie on the left side of the mean and the other half lies on the right part of the mean telling about the distribution of the values. The graph is a symmetric distribution.

   In R, there are 4 built-in functions to generate normal distribution:

   - dnorm()
     dnorm(x, mean, sd)
   - pnorm()
     pnorm(x, mean, sd)
   - qnorm()
     qnorm(p, mean, sd)
   - rnorm()
     rnorm(n, mean, sd)

2. **Binomial Distribution:**

   Binomial distribution is a common discrete distribution used in statistics, as opposed to a continuous distribution, such as normal distribution. This is because binomial distribution only counts two states, typically represented as 1 (for a success) or 0 (for a failure), given a number of trials in the data.

Binomial distribution in R is a probability distribution used in statistics. The binomial distribution is a discrete distribution and has only two outcomes i.e. success or failure. All its trials are independent, the probability of success remains the same and the previous outcome does not affect the next outcome. The outcomes from different trials are independent. Binomial distribution helps us to find the individual probabilities as well as cumulative probabilities over a certain range.

It is also used in many real-life scenarios such as in determining whether a particular lottery ticket has won or not, whether a drug is able to cure a person or not, it can be used to determine the number of heads or tails in a finite number of tosses, for analyzing the outcome of a die, etc.

We have four functions for handling binomial distribution in R namely:

- dbinom()
  dbinom(k, n, p)
- pbinom()
  pbinom(k, n, p)
  where n is total number of trials, p is probability of success, k is the value at which the probability has to be found out.
- qbinom()
  qbinom(P, n, p)
  Where P is the probability, n is the total number of trials and p is the probability of success..3
- rbinom()
  rbinom(n, N, p)

dbinom() Function:

This function is used to find probability at a particular value for a data that follows binomial distribution i.e. it finds:

P(X = k)

Syntax:

dbinom(k, n, p)

3. **Univariate Analysis:**

Univariate analysis explores each variable in a data set, separately. It looks at the range of values, as well as the central tendency of the values. It describes the pattern of response to the variable. It describes each variable on its own. Descriptive statistics describe and summarize data.

This type of data consists of only one variable. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it. The example of a univariate data can be height.

Suppose that the heights of seven students of a class is recorded(figure 1), there is only one variable that is height and it is not dealing with any cause or relationship. The description of patterns found in this type of data can be made by drawing conclusions using central tendency measures (mean, median and mode), dispersion or spread of data (range, minimum, maximum, quartiles, variance and standard deviation) and by using frequency distribution tables, histograms, pie charts, frequency polygon and bar charts.

4. **Bivariate Analysis:**

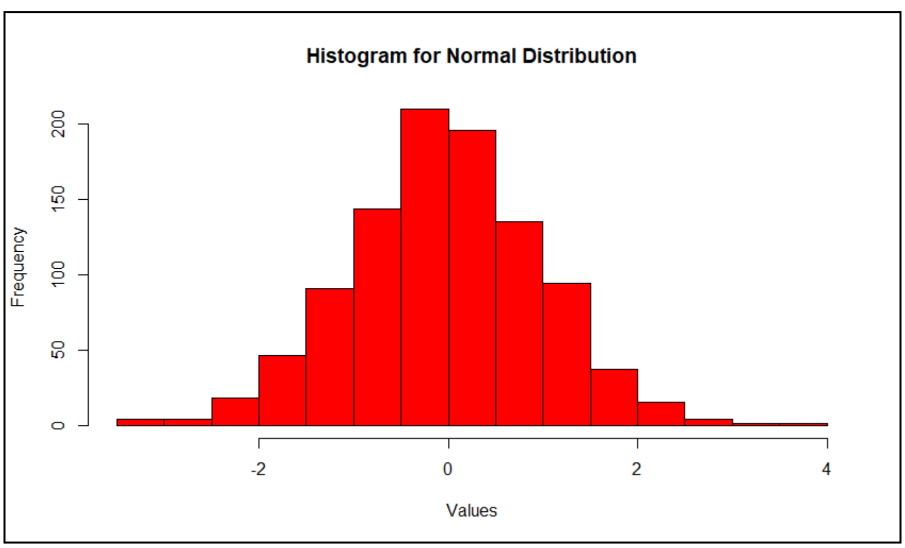
This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship among the two variables. Examples of bivariate data can be temperature and ice cream sales in summer season.Suppose the temperature and ice cream sales are the two variables of a bivariate data. Here, the relationship is visible from the table that temperature and sales are directly proportional to each other and thus related because as the temperature increases, the sales also increase. Thus bivariate data analysis involves comparisons, relationships, causes and explanations. These variables are often plotted on the X and Y axis on the graph for better understanding of data and one of these variables is independent while the other is dependent.

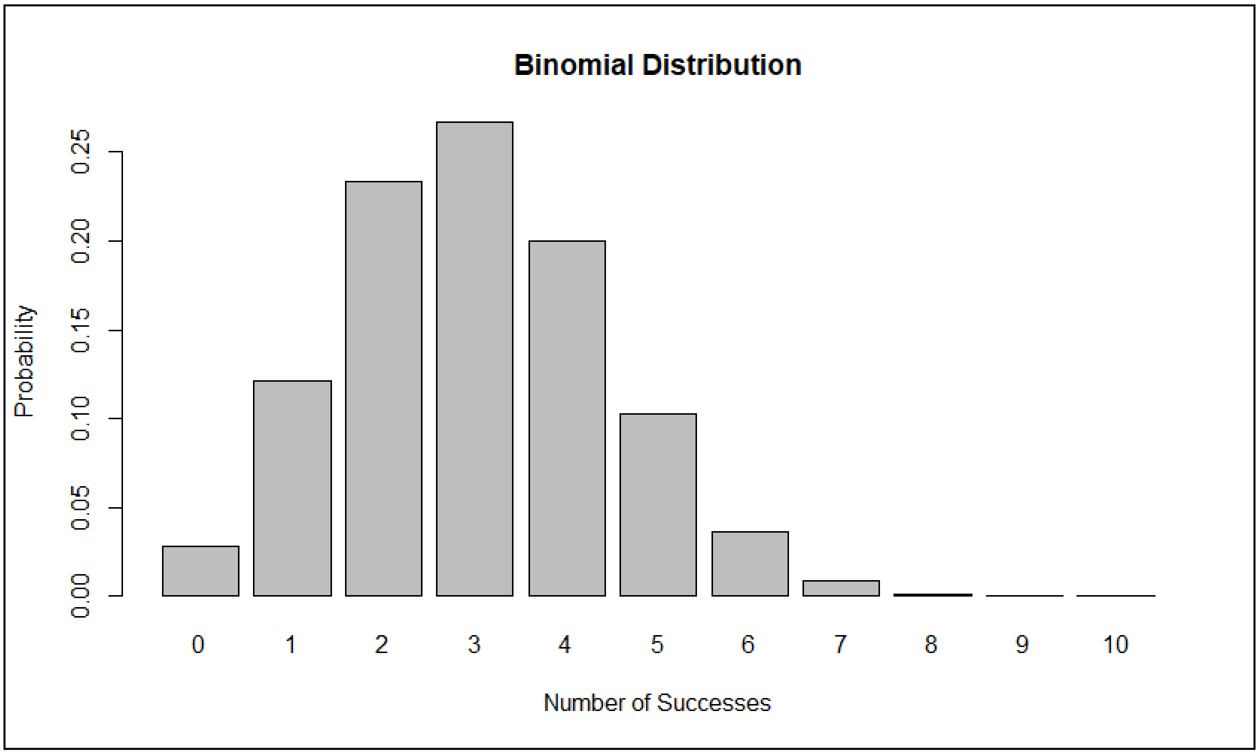
Code (Script):

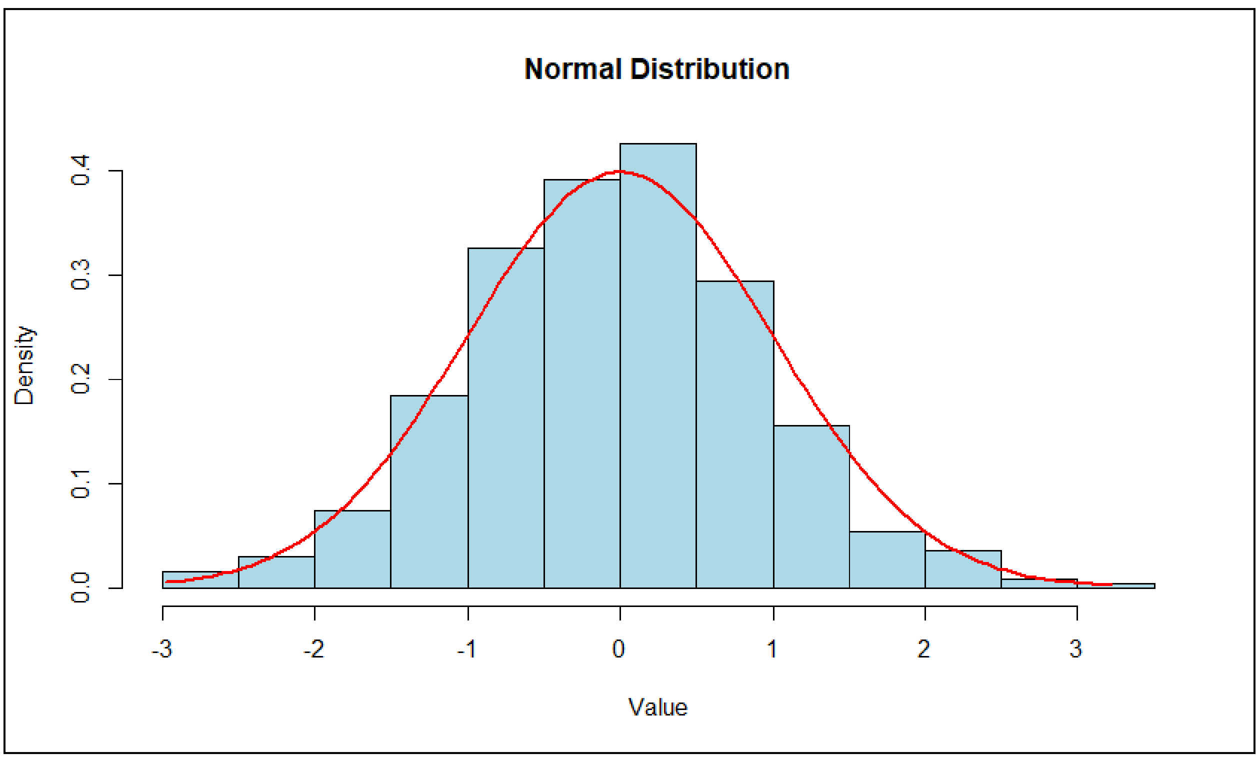
```
# Genereating Random data for normal distribution.
sample<- rnorm(100, mean=0, sd=1)

#calculating summary statistics.
mean_data<-mean(sample)
sd_data<- sd

# Creating a Histogram for data visualization.
hist(data, main="Histogram for Normal Distribution", xlab = "Values", ylab =
    "Frequency", col = "Red")

# Generate random data from a normal distribution
mean_value <- 0
sd_value <- 1
n <- 1000
data <- rnorm(n, mean = mean_value, sd = sd_value)

# Create a histogram
hist(data, main = "Normal Distribution", xlab = "Value", prob = TRUE, col = "lightblue")

# Overlay the probability density function (PDF)
x <- seq(min(data), max(data), length = 100)
y <- dnorm(x, mean = mean_value, sd = sd_value)
lines(x, y, col = "red", lwd = 2)

#Binomial Distribution
# Parameters for the binomial distribution
n_trials <- 10
probability_success <- 0.3
x_values <- 0:n_trials

# Compute the probability mass function
pmf <- dbinom(x_values, size = n_trials, prob = probability_success)
```

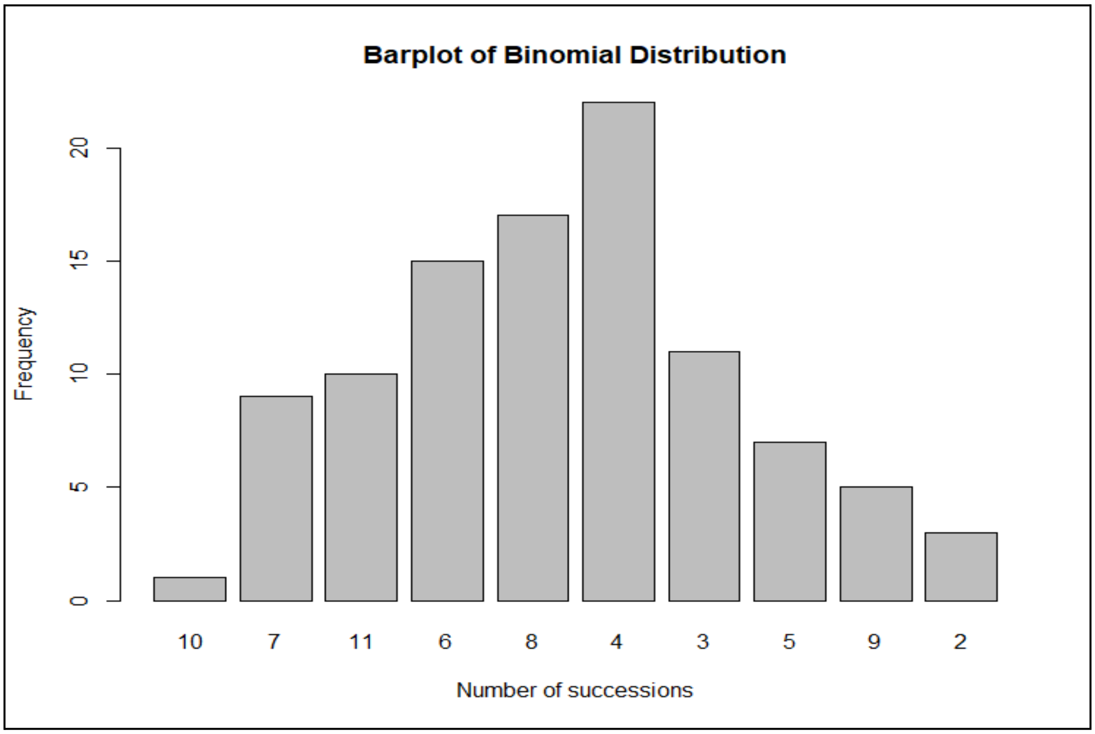
```
# Create a bar plot
barplot(pmf, names.arg = x_values, main = "Binomial Distribution", xlab = "Number of
Successes", ylab = "Probability")

# Generating data set for Binomil distribution.
data<- rbinom(100, size = 20, prob = 0.3)

#summary statistics.
mean_val<- mean(data)
sd_val<- sd(data)

# Visualiazation for the PMF of binomial distribution.
barplot(table(data), names.arg = unique(data), main="Barplot of Binomial Distribution",
    xlab="Number of successions", ylab="Frequency")

# Generating Random data for 2 variables in Normal distribution
x<-rnorm(100, mean=0, sd=1)
y<- rnorm(100, mean = 0, sd=3)

# Creating a scatterplot for Bivariate analysis.
plot(x,y, main="Scatter Plot for Bivariate Analysis", xlab = "X", ylab="Y")
```

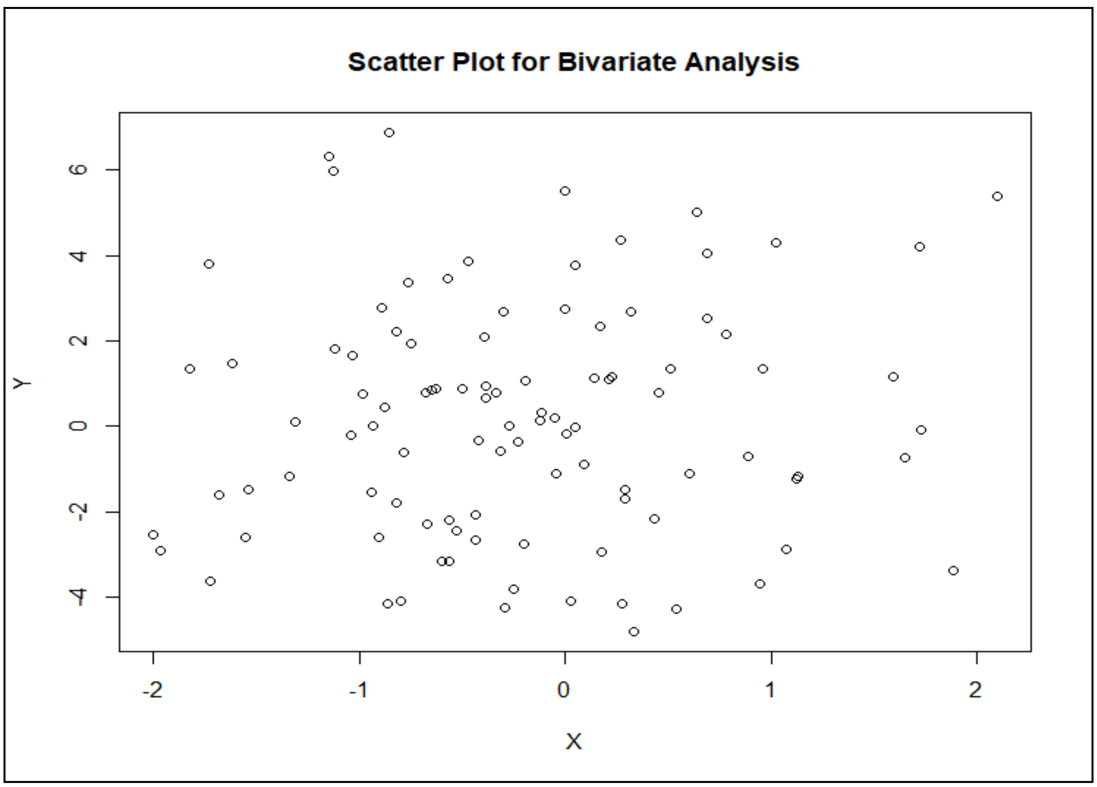
Output:




Binomial Distribution
Probability
0.00
0.05
0.10
0.15
0.20
0.25
0 1 2 3 4 5 6 7 8 9 10
Number of Successes


Normal Distribution
Density
0.0
0.1
0.2
0.3
0.4
-3 -2 -1 0 1 2 3
Value





Conclusion: Demonstrated the implementation of Normal and Binomial Distribution, Univariate and Bivariate analysis.

<table>
<tr><td colspan="6">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="3">Roll Number: 45</td><td colspan="3">Lab Assignment Number: 9</td></tr>
<tr><td colspan="6">Title of Lab Assignment: Implementation and analysis of Apriori Algorithm using Market Basket Analysis.</td></tr>
<tr><td colspan="3">DOP: 21-10-2023</td><td colspan="3">DOS: 27-10-2023</td></tr>
<tr><td colspan="2">CO Mapped:
CO6</td><td colspan="2">PO Mapped:
PO1, PO2, PO3, PO4, PO5, PO7, PO12, PSO1, PSO2</td><td colspan="2">Signature:</td></tr>
</table>

Practical No. 9

Aim: Implementation and analysis of Apriori Algorithm using Market Basket Analysis.

Theory:

Market Basket Analysis is a form of frequent itemset mining that examines consumer purchasing patterns by identifying relationships between the many goods in their "shopping baskets." By getting insight into which goods are commonly purchased together by customers, businesses may build marketing strategies based on the finding of these relationships. Market Basket Analysis is a method of determining the value of a market basket.

MBA is most often used to help in cross-selling and up-selling. If you know that customers who buy trousers also buy belts, for example, you may advertise the belts on the same page or offer them as part of a bundle to try to boost sales. You may also advertise one product while seeing an increase in the other. Customers' purchase patterns are depicted using "Association Rules" in Market Basket Analysis. A rule's interestingness is determined by two metrics: support and confidence.

Example:

Tea_powder => sugar [support = 4%, confidence = 70%]

a. A support of 2% for the above rule states that 2% of all the transactions under analysis show that tea powder and sugar are purchased together.
   support(B => C) = P(B U C)

b. A confidence of 70% means that 70% of the customers who purchased tea powder also bought the sugar.

c. Lift is a metric that helps us figure out if combining two products increases our chances of making a sale.

Packages / Functions Used:

a. **arules:** It is used for displaying, manipulating, and analyzing transaction data and patterns (frequent itemsets and association rules)

b. **inspect():** It summarizes all relevant options, plots and statistics that should be usually considered.

c. **apriori():** From a given collection of transaction data, apriori() creates the most relevant set of rules. It also demonstrates the rules' support, confidence, and lifting. The relative strength of the rules may be determined using these three criteria.

Problem Statement: Implementation and analysis of Apriori Algorithm using Market Basket Analysis.

Code (Script):

```
#reference:-https://www.analyticsvidhya.com/blog/2021/10/end-to-end-introduction-to-marketbasket-analysis-in-r/

# Install the libaries
# install.packages('arules')

# Load the libraries
library(arules)

# Load the data set
data(Groceries)

# Get the rules
grocery_rules = apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))
grocery_rules # shows that it is a set of 410 rules

# Show top 10 rules out of total 410 rules,
# Show only 2 decimal digits after the decimal
options(digits = 2)
inspect(grocery_rules[1:10])

# Sorting rules by confidence
grocery_rules = sort(grocery_rules, by = "confidence", decreasing = TRUE)
```

```
inspect(grocery_rules[1:10])

# What type of customers will buy whole milk? (whole milk is rhs)
whole_milk_rules = apriori(data = Groceries,
                           parameter = list(supp = 0.001, conf = 0.08),
                           appearance = list(default = "lhs", rhs = "whole milk")
                          )

inspect(whole_milk_rules[1:10])
whole_milk_rules = sort(whole_milk_rules, by = "confidence", decreasing = TRUE)

inspect(whole_milk_rules[1:10])

# If a customer buys "whole milk" then what else will they buy? (whole milk is lhs)
whole_milk_rules = apriori(data = Groceries,
                           parameter = list(supp = 0.001, conf = 0.08, minlen = 2),
                           appearance = list(default = "rhs", lhs = "whole milk")
                          )

inspect(whole_milk_rules[1:10])

whole_milk_rules = sort(whole_milk_rules, by = "confidence", decreasing = TRUE)

inspect(whole_milk_rules[1:10])
```

Output:

Get the rules

```
Console  Terminal  Background Jobs
R 4.3.1 · ~/
> # Load the libraries
> library(arules)
>
> # Load the data set
> data(Groceries)
>
> # Get the rules
> grocery_rules = apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.8    0.1    1 none FALSE            TRUE       5   0.001      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 9

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.01s].
writing ... [410 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> grocery_rules # shows that it is a set of 410 rules
set of 410 rules
> |
```

Sort the rules in decreasing order of confidence

```
Console  Terminal  Background Jobs
R 4.3.1 · ~/
> # Show top 10 rules out of total 410 rules,
> # show only 2 decimal digits after the decimal
> options(digits = 2)
> inspect(grocery_rules[1:10])
     lhs                                                 rhs                support confidence coverage lift count
[1]  {rice, sugar}                                    => {whole milk}       0.0012  1          0.0012   3.9  12
[2]  {canned fish, hygiene articles}                  => {whole milk}       0.0011  1          0.0011   3.9  11
[3]  {root vegetables, butter, rice}                  => {whole milk}       0.0010  1          0.0010   3.9  10
[4]  {root vegetables, whipped/sour cream, flour}     => {whole milk}       0.0017  1          0.0017   3.9  17
[5]  {butter, soft cheese, domestic eggs}             => {whole milk}       0.0010  1          0.0010   3.9  10
[6]  {citrus fruit, root vegetables, soft cheese}     => {other vegetables} 0.0010  1          0.0010   5.2  10
[7]  {pip fruit, butter, hygiene articles}            => {whole milk}       0.0010  1          0.0010   3.9  10
[8]  {root vegetables, whipped/sour cream, hygiene articles} => {whole milk} 0.0010  1          0.0010   3.9  10
[9]  {pip fruit, root vegetables, hygiene articles}   => {whole milk}       0.0010  1          0.0010   3.9  10
[10] {cream cheese , domestic eggs, sugar}            => {whole milk}       0.0011  1          0.0011   3.9  11
>
> # Sorting rules by confidence
> grocery_rules = sort(grocery_rules, by = "confidence", decreasing = TRUE)
> inspect(grocery_rules[1:10])
     lhs                                                 rhs                support confidence coverage lift count
[1]  {rice, sugar}                                    => {whole milk}       0.0012  1          0.0012   3.9  12
[2]  {canned fish, hygiene articles}                  => {whole milk}       0.0011  1          0.0011   3.9  11
[3]  {root vegetables, butter, rice}                  => {whole milk}       0.0010  1          0.0010   3.9  10
[4]  {root vegetables, whipped/sour cream, flour}     => {whole milk}       0.0017  1          0.0017   3.9  17
[5]  {butter, soft cheese, domestic eggs}             => {whole milk}       0.0010  1          0.0010   3.9  10
[6]  {citrus fruit, root vegetables, soft cheese}     => {other vegetables} 0.0010  1          0.0010   5.2  10
[7]  {pip fruit, butter, hygiene articles}            => {whole milk}       0.0010  1          0.0010   3.9  10
[8]  {root vegetables, whipped/sour cream, hygiene articles} => {whole milk} 0.0010  1          0.0010   3.9  10
[9]  {pip fruit, root vegetables, hygiene articles}   => {whole milk}       0.0010  1          0.0010   3.9  10
[10] {cream cheese , domestic eggs, sugar}            => {whole milk}       0.0011  1          0.0011   3.9  11
> |
```

What type of customers will buy whole milk? (whole milk is rhs)

```
Console  Terminal  Background Jobs
R 4.3.1 · ~/
> # What type of customers will buy whole milk? (whole milk is rhs)
> whole_milk_rules = apriori(data = Groceries,
+                            parameter = list(supp = 0.001, conf = 0.08),
+                            appearance = list(default = "lhs", rhs = "whole milk")
+                           )
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
       0.08    0.1    1 none FALSE            TRUE       5   0.001      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 9

set item appearances ...[1 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.01s].
writing ... [3765 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(whole_milk_rules[1:10])
     lhs                    rhs          support confidence coverage lift count
[1]  {}                  => {whole milk} 0.2555  0.26       1.0000   1.0  2513
[2]  {honey}             => {whole milk} 0.0011  0.73       0.0015   2.9    11
[3]  {soap}              => {whole milk} 0.0011  0.42       0.0026   1.7    11
[4]  {cocoa drinks}      => {whole milk} 0.0013  0.59       0.0022   2.3    13
[5]  {pudding powder}    => {whole milk} 0.0013  0.57       0.0023   2.2    13
[6]  {cooking chocolate} => {whole milk} 0.0013  0.52       0.0025   2.0    13
[7]  {nuts/prunes}       => {whole milk} 0.0012  0.36       0.0034   1.4    12
[8]  {potato products}   => {whole milk} 0.0012  0.43       0.0028   1.7    12
[9]  {artif. sweetener}  => {whole milk} 0.0011  0.34       0.0033   1.3    11
[10] {canned fruit}      => {whole milk} 0.0013  0.41       0.0033   1.6    13
> whole_milk_rules = sort(whole_milk_rules, by = "confidence", decreasing = TRUE)
> inspect(whole_milk_rules[1:10])
     lhs                                                 rhs          support confidence coverage lift count
[1]  {rice, sugar}                                    => {whole milk} 0.0012  1          0.0012   3.9  12
[2]  {canned fish, hygiene articles}                  => {whole milk} 0.0011  1          0.0011   3.9  11
[3]  {root vegetables, butter, rice}                  => {whole milk} 0.0010  1          0.0010   3.9  10
[4]  {root vegetables, whipped/sour cream, flour}     => {whole milk} 0.0017  1          0.0017   3.9  17
[5]  {butter, soft cheese, domestic eggs}             => {whole milk} 0.0010  1          0.0010   3.9  10
[6]  {pip fruit, butter, hygiene articles}            => {whole milk} 0.0010  1          0.0010   3.9  10
[7]  {root vegetables, whipped/sour cream, hygiene articles} => {whole milk} 0.0010  1          0.0010   3.9  10
[8]  {pip fruit, root vegetables, hygiene articles}   => {whole milk} 0.0010  1          0.0010   3.9  10
[9]  {cream cheese , domestic eggs, sugar}            => {whole milk} 0.0011  1          0.0011   3.9  11
[10] {curd, domestic eggs, sugar}                     => {whole milk} 0.0010  1          0.0010   3.9  10
> |
```

If a customer buys "whole milk" then what else will they buy? (whole milk is lhs)

Console Terminal Background Jobs

R 4.3.1 · ~/

```
> # If a customer buys "whole milk" then what else will they buy? (whole milk is lhs)
> whole_milk_rules = apriori(data = Groceries,
+                           parameter = list(supp = 0.001, conf = 0.08, minlen = 2),
+                           appearance = list(default = "rhs", lhs = "whole milk")
+                         )
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
       0.08    0.1    1 none FALSE            TRUE       5   0.001      2     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 9

set item appearances ...[1 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [23 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(whole_milk_rules[1:10])
     lhs             rhs                     support confidence coverage lift count
[1]  {whole milk} => {beef}                  0.021   0.083      0.26     1.6  209
[2]  {whole milk} => {curd}                  0.026   0.102      0.26     1.9  257
[3]  {whole milk} => {pork}                  0.022   0.087      0.26     1.5  218
[4]  {whole milk} => {frankfurter}           0.021   0.080      0.26     1.4  202
[5]  {whole milk} => {brown bread}           0.025   0.099      0.26     1.5  248
[6]  {whole milk} => {margarine}             0.024   0.095      0.26     1.6  238
[7]  {whole milk} => {butter}                0.028   0.108      0.26     1.9  271
[8]  {whole milk} => {newspapers}            0.027   0.107      0.26     1.3  269
[9]  {whole milk} => {domestic eggs}         0.030   0.117      0.26     1.9  295
[10] {whole milk} => {fruit/vegetable juice} 0.027   0.104      0.26     1.4  262
> whole_milk_rules = sort(whole_milk_rules, by = "confidence", decreasing = TRUE)
> inspect(whole_milk_rules[1:10])
     lhs             rhs                  support confidence coverage lift count
[1]  {whole milk} => {other vegetables}   0.075   0.29       0.26     1.5  736
[2]  {whole milk} => {rolls/buns}         0.057   0.22       0.26     1.2  557
[3]  {whole milk} => {yogurt}             0.056   0.22       0.26     1.6  551
[4]  {whole milk} => {root vegetables}    0.049   0.19       0.26     1.8  481
[5]  {whole milk} => {tropical fruit}     0.042   0.17       0.26     1.6  416
[6]  {whole milk} => {soda}               0.040   0.16       0.26     0.9  394
[7]  {whole milk} => {bottled water}      0.034   0.13       0.26     1.2  338
[8]  {whole milk} => {pastry}             0.033   0.13       0.26     1.5  327
[9]  {whole milk} => {whipped/sour cream} 0.032   0.13       0.26     1.8  317
[10] {whole milk} => {citrus fruit}       0.031   0.12       0.26     1.4  300
> 
```

Conclusion: Demonstrated the implementation and analysis of Apriori Algorithm using Market Basket Analysis.

<table>
<tr><td colspan="3">Name of Student: Pushkar Sane</td></tr>
<tr><td colspan="2">Roll Number: 45</td><td>Lab Assignment Number: 10</td></tr>
<tr><td colspan="3">Title of Lab Assignment: Implementation and analysis of Linear regression through graphical methods.</td></tr>
<tr><td colspan="2">DOP: 24-10-2023</td><td>DOS: 27-10-2023</td></tr>
<tr><td>CO Mapped:
CO6</td><td>PO Mapped:
PO1, PO2, PO3, PO4, PO5, PO7, PO12, PSO1, PSO2</td><td>Signature:</td></tr>
</table>

Practical No. 10

Aim: Implementation and analysis of Linear regression through graphical methods.

Theory:

Linear Regression using R

Regression analysis is used to establish relationships between two variables.

Simple linear regression is used to estimate the relationship between **two** quantitative variables.

You can use simple linear regression when you want to know:

1. How strong the relationship is between two variables (e.g. the relationship between rainfall and soil erosion).
2. The value of the **dependent variable** at a certain value of the **independent variable**. (e.g. the amount of soil erosion at a certain level of rainfall).

Predictor or independent Variable: The **values** that are gathered through **experiments** is known as a predictor variable.

Response or dependent or predicted Variable: The **values that are derived from predictor variables** are known as response variables.

Linear regression is a regression model that uses a **straight line** to describe the relationship between variables. It finds the **line of best fit through your data** by searching for the value of the regression coefficient(s) that minimizes the total error of the model.

In **linear** regression predictor and response variables are related through an equation where exponent (power) of both these variables is **1**.

A **non-linear relationship** where the exponent of a variable is not equal to 1 creates a curve.

The equation for linear regression is **y=ax+b**.

Here,

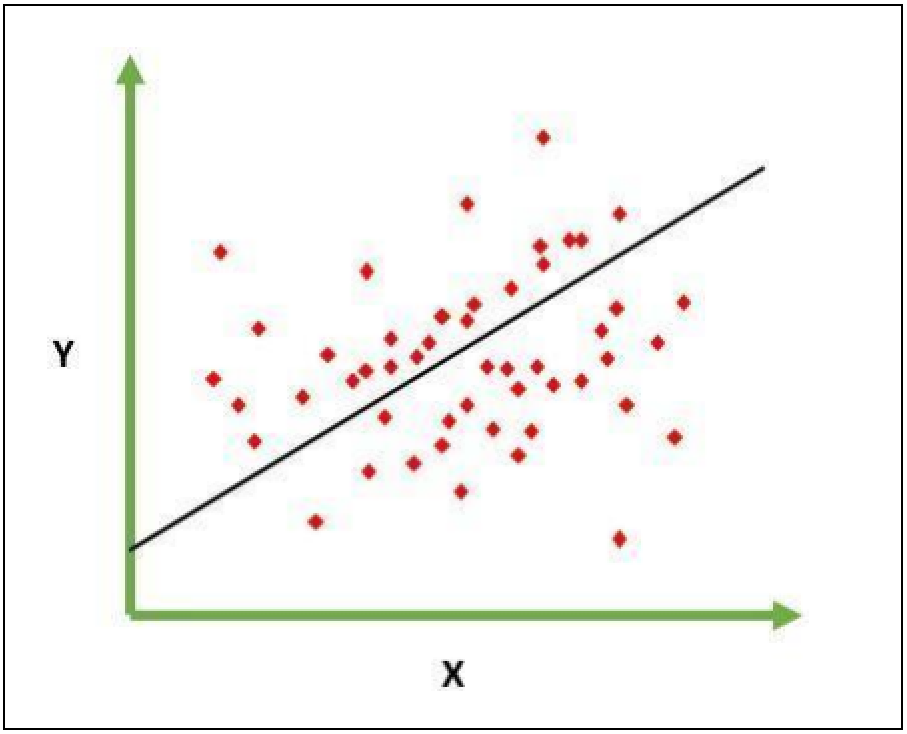
y is the response variable

x is the predictor variable

a is regression coefficient

b is y-intercept

The **linear regression** technique involves the continuous dependent variable and the independent variables can be continuous or discrete. By using best fit straight line linear regression sets up a relationship between dependent variable (Y) and one or more independent variables (X). In other words, there exists a linear relationship between independent and dependent variables.



In the above diagram, you see that the points can be anywhere in the plane of a graph in and around a straight line.

There are two main types of linear regression:

a. **Simple linear regression** uses only **one independent** variable.
b. **Multiple linear regression** uses **two or more independent** variables.

Simple linear regression:

Example :

You are a social researcher interested in the relationship between income and happiness. You survey 500 people whose incomes range from $15k to $75k and ask them to rank their happiness on a scale from 1 to 10. The income values are divided by 10,000 to make the

income data match the scale of the happiness scores (so a value of $2 represents $20,000, $3 is $30,000, etc.)
Your independent variable (income) and dependent variable (happiness) are both quantitative, so you can do a regression analysis to see if there is a linear relationship between them.

Multiple linear regression:
It is used to estimate the relationship between **two or more independent variables** and **one dependent variable.**

Equation can be of the form: **y= ax + bz + c**
y is the response variable
x and **z** are the predictor variable
a and **b** is regression coefficient
c is y-intercept

You can use multiple linear regression when you want to know:

1. How strong the relationship is between two or more independent variables and one dependent variable (e.g. how rainfall (1st independent variable), temperature (2nd independent variable), and amount of fertilizer added (3rd independent variable), affect crop growth (dependent variable).
2. The value of the dependent variable at a certain value of the independent variables (e.g. the expected yield of a crop at certain levels of rainfall, temperature, and fertilizer addition).

Example:
You are a public health researcher interested in social factors that influence heart disease. You survey 500 towns and gather data on the percentage of people in each town who smoke, the percentage of people in each town who bike to work, and the percentage of people in each town who have heart disease.
Because you have two independent variables and one dependent variable, and all your variables are quantitative, you can use multiple linear regression to analyze the relationship between them.

How to do the practical in R

install.packages("ggplot2")

library(ggplot2)

A system for 'declaratively' creating graphics, based on "The Grammar of Graphics". You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Step 1: Load the data into R

Simple regression

incomedata = read.csv("income.data for linear regression.csv")

summary(incomedata)

Because both our variables are quantitative, when we run this function we see a table in our console with a numeric summary of the data. This tells us the minimum, median, mean, and maximum values of the independent variable (income) and dependent variable (happiness):

```
       X             income        happiness    
 Min.   :  1.0   Min.   :1.506   Min.   :0.266  
 1st Qu.:125.2   1st Qu.:3.006   1st Qu.:2.266  
 Median :249.5   Median :4.424   Median :3.473  
 Mean   :249.5   Mean   :4.467   Mean   :3.393  
 3rd Qu.:373.8   3rd Qu.:5.992   3rd Qu.:4.503  
 Max.   :498.0   Max.   :7.482   Max.   :6.863  
```

Step 2: Make sure your data meet the assumptions

We can use R to check that our data meet the 3 main assumptions for linear regression.

Simple regression:

1. **Independence of observations:**

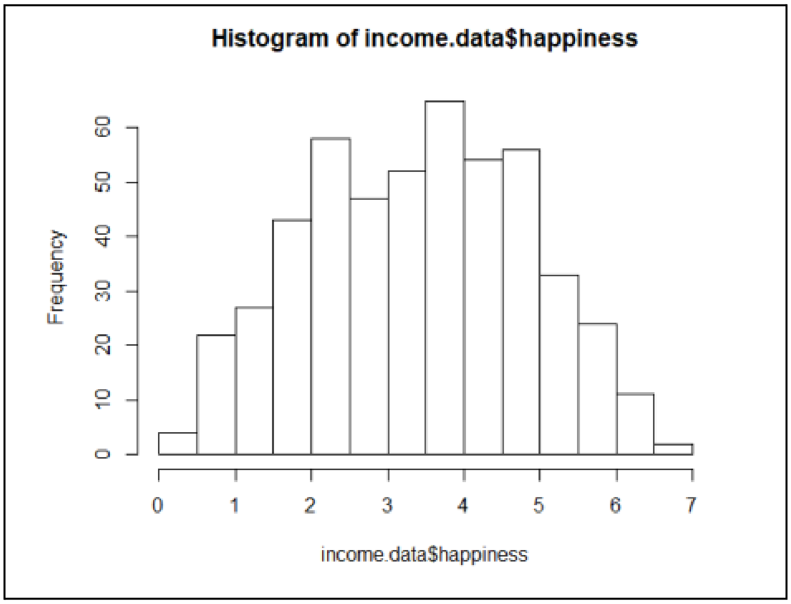
   Because we only have **one independent variable and one dependent variable**, we don't need to test for any hidden relationships among variables.

   If you know that you have autocorrelation within variables (i.e. multiple observations of the same test subject), then do not proceed with a simple linear regression! Use a structured model, like a linear mixed-effects model, instead.

2. **Normality:**

   To check whether the **dependent variable** follows a normal distribution, use the hist() function.

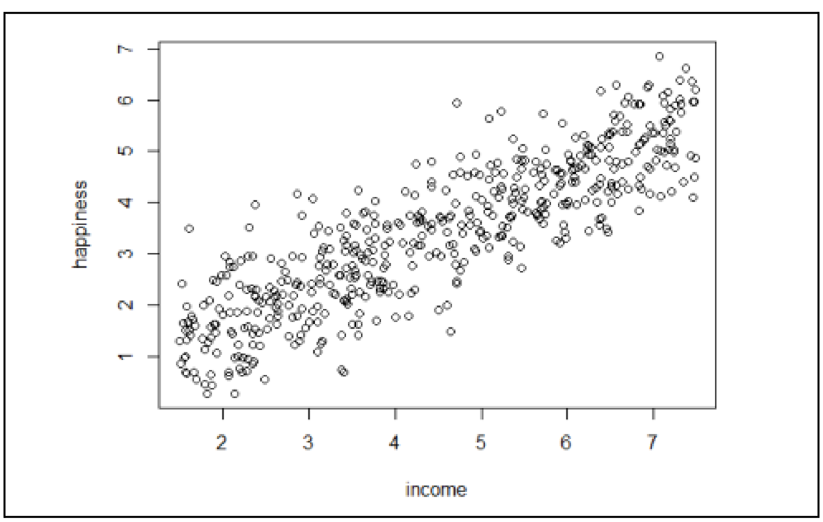
   hist(incomedata$happiness)



3. **Linearity:**

   The **relationship between the independent and dependent variable must be linear**. We can test this visually with a **scatter plot** to see if the **distribution of data points could be described with a straight line**.

   plot(happiness ~ income, data = incomedata)



   The relationship looks **roughly linear**, so we can proceed with the linear model.

Step 3: Perform the linear regression analysis.

Simple regression: income and happiness

Let's see if there's a linear relationship between income and happiness in our survey of 500 people with incomes ranging from $15k to $75k, where happiness is measured on a scale of 1 to 10.

To perform a simple linear regression analysis and check the results, you need to run two lines of code. The first line of code makes the linear model, and the second line prints out the summary of the model:

The lm() function

In R, the lm(), or "linear model," function can be used to create a simple regression model. The lm() function accepts a number of arguments. The following list explains the two most commonly used parameters.

- **formula:** describes the model

  Note that the formula argument follows a specific format. For simple linear regression, this is **"YVAR ~ XVAR"** where *YVAR is the dependent, or predicted or target variable and XVAR is the independent, or predictor variable*.
- **data:** the variable that contains the dataset

  lm([target variable] ~[predictor variables], data = [data source])

  income.happiness.lm <- lm(happiness ~ income, data = incomedata)

  summary(income.happiness.lm)

  The output looks like this:

```
Call:
lm(formula = happiness ~ income, data = income.data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.02479 -0.48526  0.04078  0.45898  2.37805

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.20427    0.08884   2.299   0.0219 *
income       0.71383    0.01854  38.505   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7181 on 496 degrees of freedom
Multiple R-squared:  0.7493,	Adjusted R-squared:  0.7488
F-statistic:  1483 on 1 and 496 DF,  p-value: < 2.2e-16
```

This output table **first repeats the formula** that was used to generate the results ('Call'), then **summarizes the model residuals** ('Residuals'), which give an idea of how well the model fits the real data.

Next is the **'Coefficients'** table.

The **first row gives the estimates of the y-intercept**, and the **second row gives the regression coefficient** of the model.

Row 1 of the table is labeled <u>(Intercept)</u>. This is the **y-intercept** of the regression equation, with a value of 0.20. You can plug this into your regression equation if you want to predict happiness values across the range of income that you have observed:

*happiness = 0.20 + 0.71*income ± 0.018*

Row 2 in the 'Coefficients' table is **income**. This is the row that describes the estimated effect of income on reported happiness:

The <u>Estimate</u> column is the estimated **effect**, also called the **regression coefficient** or r2 value. The number in the table (0.713) tells us that for every one unit increase in income (where one unit of income = $10,000) there is a corresponding 0.71-unit increase in reported happiness (where happiness is a scale of 1 to 10).

The <u>Std. Error</u> column displays the **standard error** (*In statistics, a sample mean deviates from the actual mean of a population; this deviation is the standard error of the mean.*) of the estimate. This number shows how much variation there is in our estimate of the relationship between income and happiness.

The **Pr(>| t |)** column shows the **p-value**. Because the p-value is so low ($p < 0.001$), we can **reject the null hypothesis** and **conclude that income has a statistically significant effect on happiness**.

The last three lines of the model summary are statistics about the model as a whole.

The most important thing to notice here is the p-value of the model.

Here it is significant ($p < 0.001$), which means that this model is a good fit for the **observed data**.

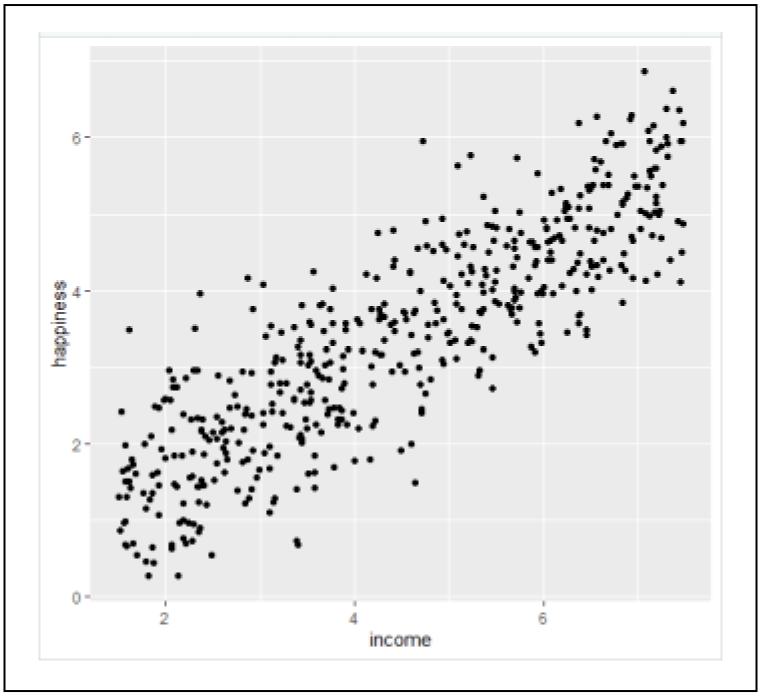
Step 4: Visualize the results with a graph

Simple regression

Follow 4 steps to visualize the results of your simple linear regression.

1. **Plot the data points on a graph**

   income.graph <- ggplot(incomedata, aes(x=income, y=happiness))+
   geom_point()
   income.graph



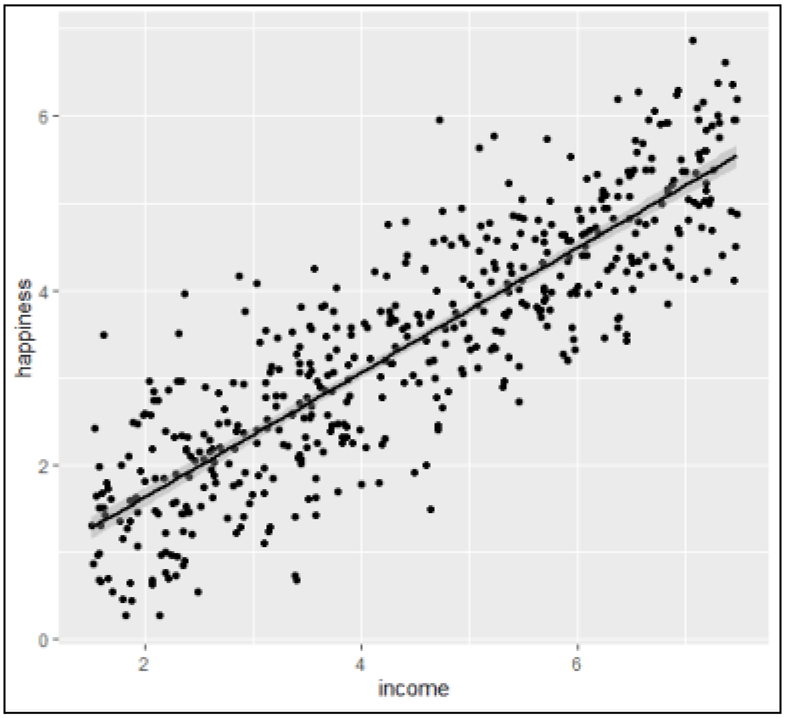
ggplot2 is a plotting package that makes it simple to create complex plots from data in a data frame.

geom_point() is used to create scatterplots. The scatterplot is most useful for displaying the relationship between two continuous variables.

2. **Add the linear regression line to the plotted data**

   Add the regression line using geom_smooth() and typing in lm as your method for creating the line. This will add the line of the linear regression as well as the standard error of the estimate (in this case +/- 0.01) as a light gray stripe surrounding the line:

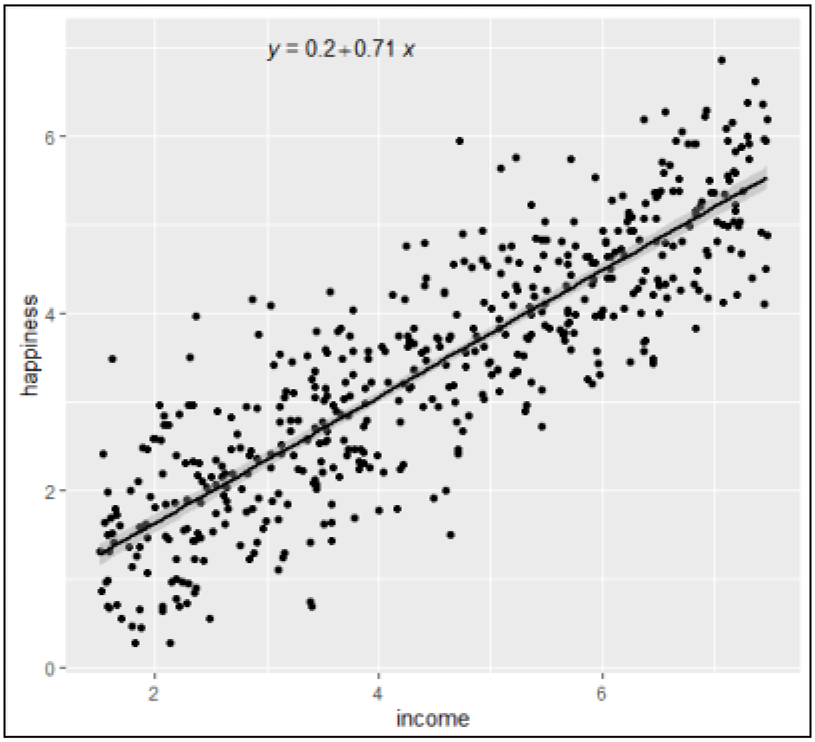
   income.graph <- income.graph + geom_smooth(method="lm", col="black")
   income.graph



3. **Add the equation for the regression line.**

```
install.packages("ggpubr") // to use stat_regline_equation() function
library(ggpubr)
income.graph <- income.graph +
                    stat_regline_equation(label.x = 3, label.y = 7)
income.graph
```

label.x, label.y	numeric coordinates (in data units) to be used for absolute positioning of the label.



Problem Statement: Implementation and analysis of Linear regression through graphical methods.

Code (Script):

```
# Linear regression
# simple linear regression
# Step 1: Load the data into R
incomedata = read.csv("C:\\Users\\ADMIN\\Desktop\\practicals\\dar\\practical 10\\income.data

for linear regression.csv")
summary(incomedata)

# Step 2: Make sure your data meet the assumptions
# Normality
hist(incomedata$happiness)

# Linearity
plot(happiness ~ income, data = incomedata)
```

```
# Step 3: Perform the simple linear regression analysis
income.happiness.lm <- lm(happiness ~ income, data = incomedata)
summary(income.happiness.lm)

# Step 4: Visualize the results with a graph
library(ggplot2)
# scatter plot
income.graph<-ggplot(incomedata, aes(x=income, y=happiness))+ geom_point()
income.graph

# Add the linear regression line to the plotted data
income.graph <- income.graph + geom_smooth(method="lm")
income.graph

# Add the equation for the regression line.
library(ggpubr)
income.graph <- income.graph +
  stat_regline_equation(label.x = 3, label.y = 7)
income.graph
```

Output:

Step 1:

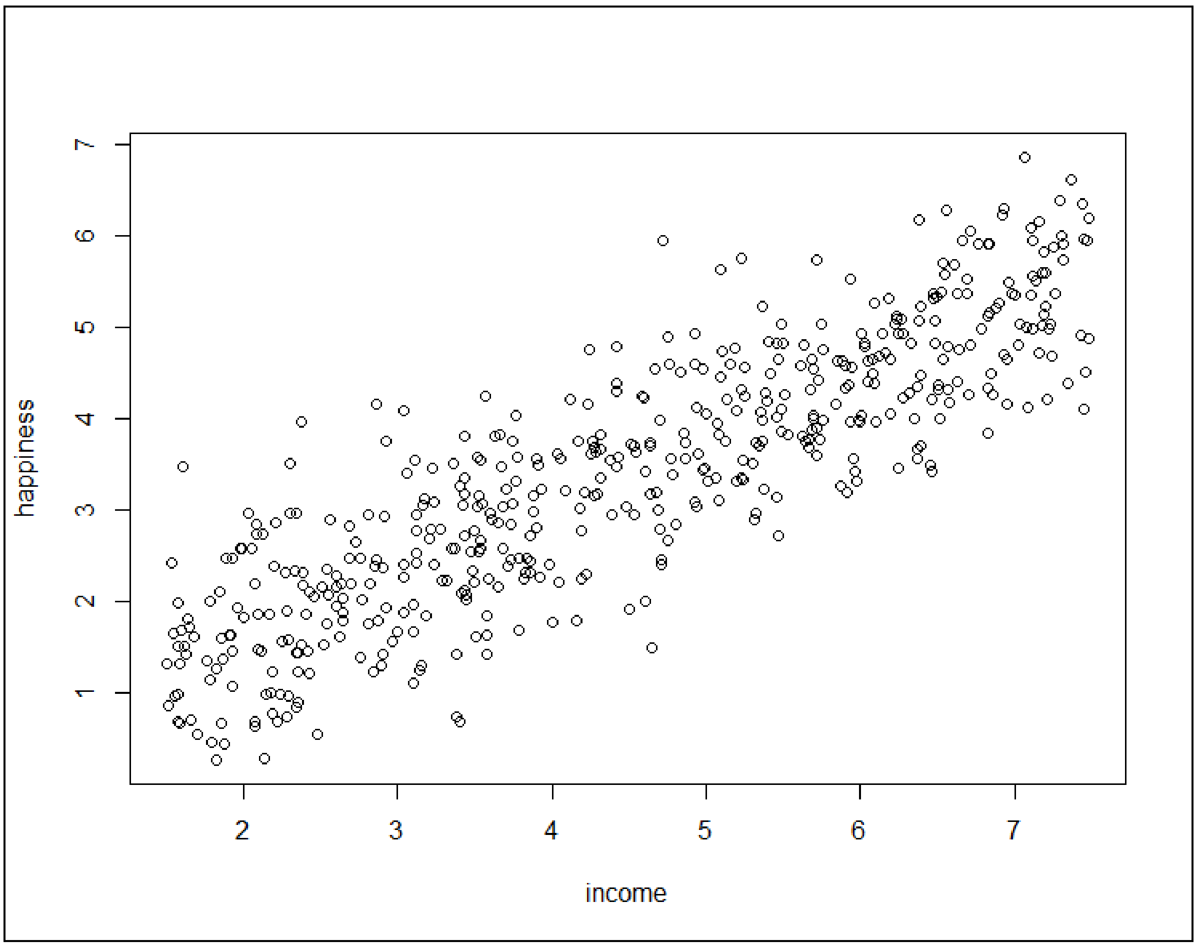
```
Console   Terminal ×   Background Jobs ×
R 4.3.1 · ~/
> # Linear regression
> # simple linear regression
> # Step 1: Load the data into R
> incomedata = read.csv("F:/Pushkar/MCA/Sem-1/DAR/income.data for linear regression.csv")
> summary(incomedata)
       X             income          happiness    
 Min.   :  1.0   Min.   :1.506   Min.   :0.266  
 1st Qu.:125.2   1st Qu.:3.006   1st Qu.:2.266  
 Median :249.5   Median :4.424   Median :3.473  
 Mean   :249.5   Mean   :4.467   Mean   :3.393  
 3rd Qu.:373.8   3rd Qu.:5.992   3rd Qu.:4.503  
 Max.   :498.0   Max.   :7.482   Max.   :6.863  
>
```

Step 2:

Console Terminal Background Jobs

R 4.3.1 · ~/

```
> # Step 2: Make sure your data meet the assumptions
> # Normality
> hist(incomedata$happiness)
>
> # Linearity
> plot(happiness ~ income, data = incomedata)
>
```

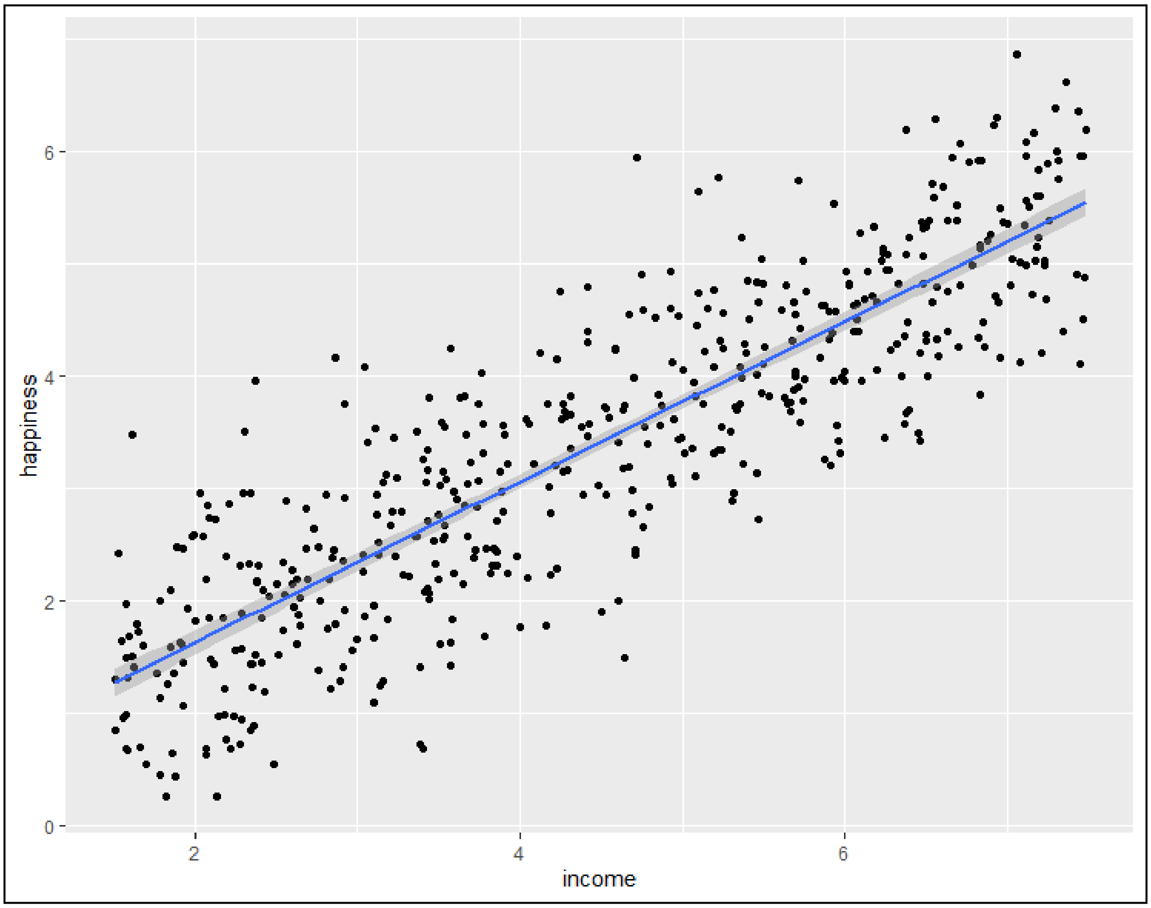


Step 3:

```
Console   Terminal ×   Background Jobs ×
R 4.3.1 · ~/
> # Step 3: Perform the simple linear regression analysis
> income.happiness.lm <- lm(happiness ~ income, data = incomedata)
> summary(income.happiness.lm)

Call:
lm(formula = happiness ~ income, data = incomedata)

Residuals:
     Min       1Q   Median       3Q      Max 
-2.02479 -0.48526  0.04078  0.45898  2.37805 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.20427    0.08884   2.299   0.0219 *  
income       0.71383    0.01854  38.505   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7181 on 496 degrees of freedom
Multiple R-squared:  0.7493,	Adjusted R-squared:  0.7488 
F-statistic:  1483 on 1 and 496 DF,  p-value: < 2.2e-16

>
```

Step 4:

```
Console   Terminal ×   Background Jobs ×
R 4.3.1 · ~/
> # Step 4: Visualize the results with a graph
> library(ggplot2)
> # scatter plot
> income.graph<-ggplot(incomedata, aes(x=income, y=happiness))+ geom_point()
> income.graph
> 
> # Add the linear regression line to the plotted data
> income.graph <- income.graph + geom_smooth(method="lm")
> income.graph
`geom_smooth()` using formula = 'y ~ x'
> 
> # Add the equation for the regression line.
> library(ggpubr)
> income.graph <- income.graph +
+   stat_regline_equation(label.x = 3, label.y = 7)
> income.graph
`geom_smooth()` using formula = 'y ~ x'
> 
```



Conclusion: : We implemented commands for drawing various Correlation Plots and learnt the process of EDA.