Government College of Engineering Jalgaon M.S.

Department of Computer Engineering

T.Y. B. Tech. CSE

CO358U ‘R’ PROGRAMMING LAB

Lab Manuals

List of Experiments

Name of the Student:

Batch, Class and Branch: Third Year Computer

PRN No:

Group A

Sr. No.	Title of Expt.	Date of Performance	Date of Completion	Sign. of Teacher
1.	Introduction to R			
2.	Programming Using R			
3.	Lists and Frames			
4.	Import and Export Files in R			
5.	Mathematical and Statistical Concepts in R			

Group B

Sr. No.	Title of Expt.	Date of Performance	Date of Completion	Sign. of Teacher
1	Write a R program that swaps any two numbers without using any third number.			
2	Write a R program script using for, while and repeat loop that prints the value of i from 1 to 10.			
3	Write a R program script to find the factorial of any given number using a recursive Function			
4	Write a R program that reads the csv file. Find the maximum and minimum values among all three.			
5	Using the various in-built functions plot pie chart, scatter plot, histogram and line charts			

Group C

Sr. No.	Title of Expt.	Date of Performance	Date of Completion	Sign. of Teacher
1	For Iris dataset visualize data using plot() also perform filter(), select(),mutate(), arrange() functions			
2	Write a R program that will identify and remove the missing values from datasets using frequency mean, median or mode options..			
3	Write a R program that will identify outliers and remove outliers from dataset			
4	Using lm() function, perform linear regression on the dataset			
5	Write a R script to predict classification of values using decision trees			

Group A

Government College of Engineering, Jalgaon **(An Autonomous Institute of Government of Maharashtra)**	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.1

AIM: Introduction to R Language

THEORY:

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

Features of R

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R −

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

How to install Rstudio on windows 10?

step by step process to download and #install r and #rstudio on #windows 10 OS (Operating System) and also how you can run r program in rstudio.

First install R Software Back End

The Comprehensive R Archive Network (r-project.org) https//cran.r-project.org

Link 1 : https://cran.r-project.org/bin/window...

Then install R Studio IDE Front End Download

the RStudio IDE - RStudio

https://rstudio.com/products/rstudio/download/

Link 2 : https://rstudio.com/products/rstudio/...

QUESTIONS:

1. **How you will install R in windows?**

Ans: 1. Download R:

- Go to CRAN.
- Click "Download R for Windows" → "base" → Download the latest version.

2. Install R:

- Run the downloaded .exe file.
- Follow the installation steps (default settings are fine).
- Click "Finish" after installation.

3. Verify Installation:

- Open R and type:

  version

2. **State and explain features of R?**

Ans:

1. Open-Source and Free: R is an open-source software, freely available under the GNU General Public License.

2. Statistical and Data Analysis Capabilities: Provides powerful functions for statistical analysis, including:

- Linear and non-linear modeling.
- Time-series analysis.
- Classification and clustering.

3. Data Visualization: Supports advanced graphical capabilities with packages like: ○ ggplot2 for customized plots.

- ○ plotly for interactive graphs.

4. Extensive Package Ecosystem: Thousands of packages available through CRAN to extend functionality for: ○ Machine learning (caret, randomForest).

- ○ Text mining (tm, text2vec).

5. Highly Flexible and Extensible: Users can create their own functions and packages to enhance R's capabilities.

6. Data Manipulation: Powerful packages such as: ○ dplyr for data manipulation.

- ○ tidyr for data cleaning and transformation.

7. Integration with Other Languages: R can integrate with: ○ C, C++, and Java for performance enhancement.

- ○ Python via reticulate package.

8. Community Support and Documentation: Large and active user community with extensive documentation and online forums.

9. Cross-Platform Compatibility: Available for Windows, macOS, and Linux.

10. Reproducible Research: Supports Markdown and Shiny for interactive web applications and reproducible reports.

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.2

AIM: Programming Using R

THEORY:

R - Data Types

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

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

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

Vectors

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

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

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

Lists

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

apple

```
# Create a list. listlist1 <-
list(c(2,5,3),21.3,sin)

# Print the list.
print(list1)
```

Matrices

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

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

Arrays

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

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

QUESTION:

State and explain various data types in R ?

Ans: 1. Decimal numbers.

x <- 3.14 # "numeric"

2. Integer: Whole numbers (use L).

y <- 5L # "integer"a<- 3. Character: Text or strings. z <- "Hello" # "character" 4. Logical: Boolean values. a <- TRUE # "logical"

5. Complex: Numbers with real & imaginary parts.

b <- 2 + 3i # "complex"

6. Raw: Stores raw bytes.

c <- charToRaw("Hello") # "raw"

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.3

AIM: Lists and Frames in R

THEORY:

R - Lists

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function. Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

```
# Create a list containing strings, numbers, vectors and a logical #
values. list_data <- list("Red", "Green", c(21,32,11), TRUE,
51.23, 119.1) print(list_data)
```

Naming List Elements

The list elements can be given names and they can be accessed using these names.

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))

# Give names to the elements in the list. names(list_data) <- c("1st Quarter", "A_Matrix",
"A Inner list")# Show the list.
print(list_data)
```

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example −

```
# Create a list containing a vector, a matrix and a list. list_data <-
list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Access the first element of the list. print(list_data[1])

# Access the thrid element. As it is also a list, all its elements will be printed. print(list_data[3])

# Access the list element using the name of the element.
print(list_data$A_Matrix)
```

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```
# Create a list containing a vector, a matrix and a list. list_data <-
list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
list("green",12.3))

# Give names to the elements in the list. names(list_data) <- c("1st
Quarter", "A_Matrix", "A Inner list")

# Add element at the end of the list.
list_data[4] <- "New element"
```

```
print(list_data[4])

# Remove the last element. list_data[4] <- NULL

# Print the 4th Element. print(list_data[4])

# Update the 3rd Element.
list_data[3] <- "updated element" print(list_data[3])
```

Merging Lists

You can merge many lists into one list by placing all the lists inside one list() function.

```
# Create two lists. list1 <-
list(1,2,3) list2 <-
list("Sun","Mon","Tue")

# Merge the two lists.
merged.list <- c(list1,list2)

# Print the merged list.
print(merged.list)
```

Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

```
# Create lists. list1
<-
list(1:5)
print(list
1)

list2 <-list(10:14) print(list2)

# Convert the lists to
vectors. v1 <- unlist(list1)
v2
<- unlist(list2)

print(v1
)
print(v2
)
```

R - Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

```
# Create the data frame. emp.data <- data.frame(     emp_id
= c (1:5),     emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
                salary =
c(6□23.3,515.2,611.0,729.0,843.25),
```

```
   start_date = as.Date(c("2012-01-01", "2013-09-23",
      "2014-11-15", "2014-05-11", "2015-03-27□")),
stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

Get the Structure of the Data Frame

The structure of the data frame can be seen by using **str()** function.

```
# Create the data frame. emp.data <- data.frame(     emp_id
= c (1:5),     emp_name =
c("Rick","Dan","Michelle","Ryan","
Gary"),        salary =
c(6□23.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15",
      "2014-05-11", "2015-03-27□")),
stringsAsFactors = FALSE
)
# Get the structure of the data
frame. str(emp.data)
```

QUESTION:

Compare List and Frame in R Ans:

Feature	List	Data Frame
Structure	Collection of elements	Tabular (rows & columns)
Data Type	Can store mixed types	Columns store same type
Access	Access via `[[ ]]` or `$`	Access via `$` or `[]`
Dimension	1-Dimensional	2-Dimensional
Indexing	Single index	Row & column indexing
Flexibility	Can store lists, vectors, etc.	Stores data like a table
Conversion	Can convert to Data Frame	Can be converted to List

Sign of Course Teacher

Government College of Engineering, Jalgaon	
(An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.4

AIM:

Import and Export Files in R

THEORY:

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

R - CSV Files

Reading a CSV File

Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory −

```
data <- read.csv("input.csv")
print(data)
```

R - Excel File

Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with required package name to install the additional packages.

```
install.packages("xlsx")
```

Reading the Excel File

The input.xlsx is read by using the **read.xlsx()** function as shown below. The result is stored as a data frame in the R environment.

```
# Read the first worksheet in the file input.xlsx. data
<- read.xlsx("input.xlsx", sheetIndex = 1)
print(data)
```

QUESTION:

Explain how you will import and export the data file in R.

Ans: Import Data:

1. CSV File:data <- read.csv("file.csv")
2. Excel File:

Install and load the package

install.packages("readxl")

library(readxl)

data <- read_excel("file.xlsx")

3. Text File: data <- read.table("file.txt", header = TRUE, sep = "\t")

Export Data:

1. To CSV: write.csv(data, "output.csv", row.names = FALSE)
2. To Excel:

Install and load the package

install.packages("writexl")

library(writexl)

write_xlsx(data, "output.xlsx")

3. To Text File:write.table(data, "output.txt", sep = "\t", row.names = FALSE)

Sign of Course Teacher

Government College of Engineering, Jalgaon	
(An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.5

AIM: Mathematical and Statistical Concepts in R

THEORY:

R - Mean, Median and Mode

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

The functions we are discussing in this chapter are mean, median and mode. Mean

It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function **mean()** is used to calculate this in R.

Syntax

The basic syntax for calculating mean in R is − mean(x,

trim = 0, na.rm = FALSE, ...)

Following is the description of the parameters used −

- **x** is the input vector.
- **trim** is used to drop some observations from both end of the sorted vector.
- **na.rm** is used to remove the missing values from the input vector.

```
Example # Create a vector.  x <-
c(12,7□,3,4.2,18,2,54,-21,8,-5)


# Find Mean.
result.mean <- mean(x)print(result.mean)
```

Applying Trim Option

When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.

When trim = 0.3, 3 values from each end will be dropped from the calculations to find mean.

In this case the sorted vector is (−21, −5, 2, 3, 4.2, 7□, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (−21,−5,2) from left and (12,18,54) from right.

```
# Create a vector.
x <- c(12,7□,3,4.2,18,2,54,-21,8,-5)
# Find Mean. result.
mean <- mean(x,trim = 0.3)
print(result.mean)
```

Median

The middle most value in a data series is called the median. The **median()** function is used in R to calculate this value. Syntax

The basic syntax for calculating median in R is − median(x,

na.rm = FALSE)

Following is the description of the parameters used −

- **x** is the input vector.
- **na.rm** is used to remove the missing values from the input vector.

Example

```
# Create the vector.
x <- c(12,7□,3,4.2,18,2,54,-21,8,5)

# Find the median.
median.result <- median(x)
print(median.result)
```

Mode

The mode is the value that has highest number of occurrences in a set of data. Unike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

Example

```
# Create the function.
getmode <- function(v) {
uniqv <- unique(v)
uniqv[which.max(tabulate(match(v,uniqv)))]
}
# Create the vector with numbers.
v <-c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)
# Calculate the mode using the user function.
result <- getmode(v)
print(result)
# Create the vector with characters.
charv <- c("o","it","the","it","it")
# Calculate the mode using the user function.
result <- getmode(charv)
print(result)
```

R - Pie Charts

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is − pie(x,

labels, radius, main, col, clockwise)

Following is the description of the parameters used −

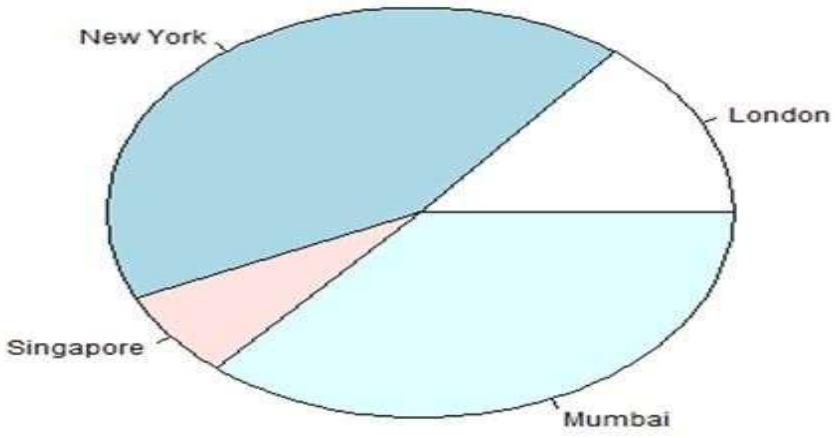
- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city.png")

# Plot the chart.
pie(x,labels)

# Save the file.
dev.off()
```

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



Pie Chart Title and Colors

We can expand the features of the chart by adding more parameters to the function. We will use parameter **main** to add a title to the chart and another parameter is **col** which will make use of rainbow

colour pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use length(x).

Example

The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city_title_colours.jpg")

# Plot the chart with title and rainbow color pallet.
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

# Save the file.
dev.off()
```

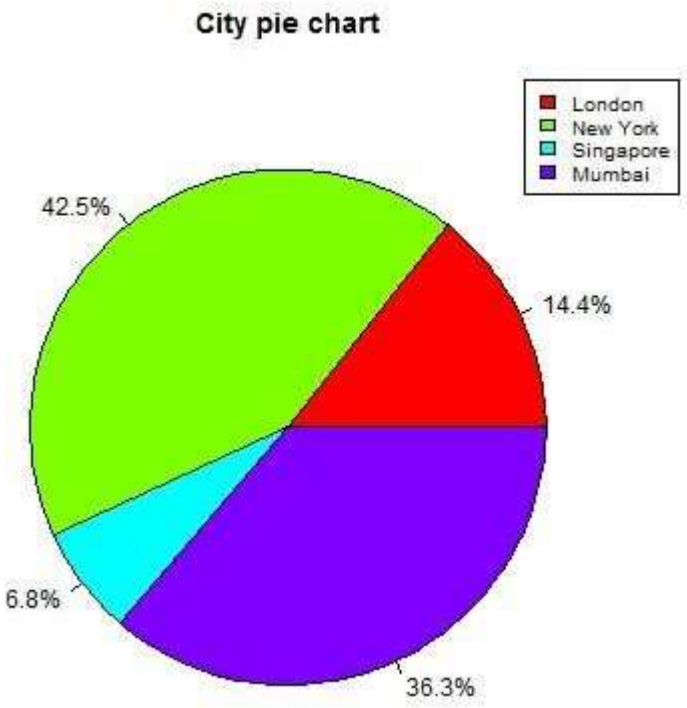
When we execute the above code, it produces the following result −

Slice Percentages and Chart Legend

We can add slice percentage and a chart legend by creating additional chart variables.

```
# Create data for the graph.
x <- c(21, 6□2, 10,53)
labels <- c("London","New York","Singapore","Mumbai")

piepercent<- round(100*x/sum(x), 1)

# Give the chart file a name.
png(file = "city_percentage_legends.jpg")

# Plot the chart. pie(x, labels = piepercent, main = "City pie chart",col =
rainbow(length(x))) legend("topright", c("London","New
York","Singapore","Mumbai"), cex = 0.8,    fill = rainbow(length(x)))

# Save the file. dev.off()
```

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



3D Pie Chart

A pie chart with 3 dimensions can be drawn using additional packages. The package **plotrix** has a function called **pie3D()** that is used for this.

```
# Get the library. library(plotrix)

# Create data for the graph. x
<- c(21, 62, 10,53)
lbl <- c("London","New York","Singapore","Mumbai")

# Give the chart file a name. png(file
= "3d_pie_chart.jpg")

# Plot the chart. pie3D(x,labels = lbl,explode = 0.1, main = "Pie
Chart of Countries ")

# Save the file. dev.off()
```

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

R - Bar Charts

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is − barplot(H,xlab,ylab,main, names.arg,col)

Following is the description of the parameters used −

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

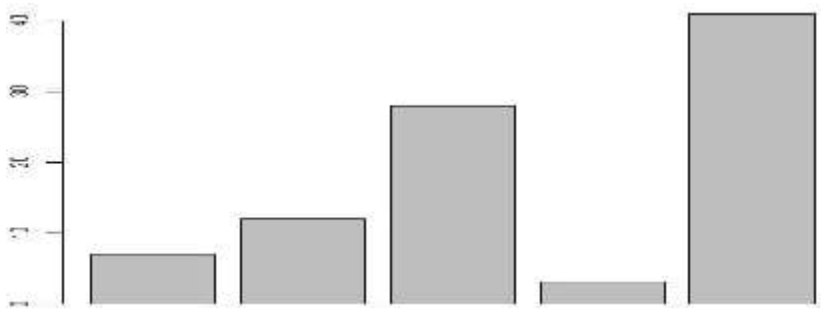
Example

A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the
chart H <-
c(7□,12,28,3,41)

# Give the chart file a name
png(file = "barchart.png")

# Plot the bar chart
barplot(H)

# Save the file dev.off()
```

When we execute above code, it produces following result −



Bar Chart Labels, Title and Colors

The features of the bar chart can be expanded by adding more parameters. The **main** parameter is used to add **title**. The **col** parameter is used to add colors to the bars. The **args.name** is a vector having same number of values as the input vector to describe the meaning of each bar. Example

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart
H <- c(7,12,28,3,41)
M <- c("Mar","Apr","May","Jun","Jul")

# Give the chart file a name
png(file = "barchart_months_revenue.png")

# Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue", main="Revenue
chart",border="red")

# Save the file dev.off()
```

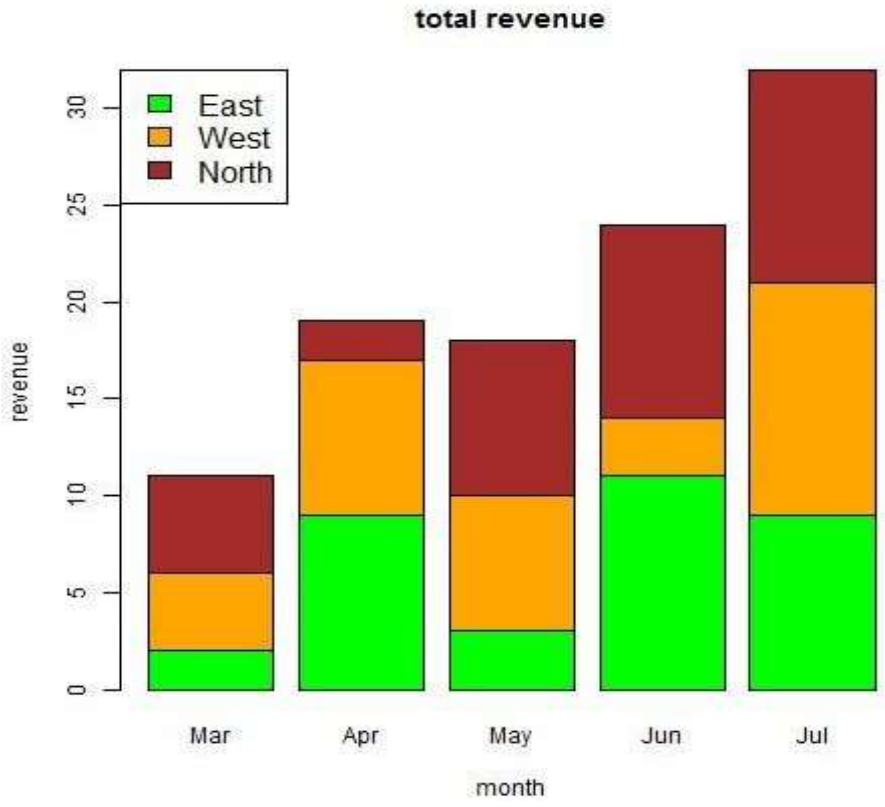
When we execute above code, it produces following result −

Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

```
# Create the input vectors. colors =
c("green","orange","brown") months <-
c("Mar","Apr","May","Jun","Jul") regions <-
c("East","West","North")

# Create the matrix of the values.
Values <- matrix(c(2,9□,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow = 3, ncol = 5, byrow = TRUE)

# Give the chart file a name png(file
= "barchart_stacked.png")

# Create the bar chart
barplot(Values, main = "total revenue", names.arg = months, xlab = "month", ylab = "revenue", col
= colors)

# Add the legend to the chart
legend("topleft", regions, cex = 1.3, fill = colors)

# Save the file dev.off()
```



R - Boxplots

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile

in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.

Syntax

The basic syntax to create a boxplot in R is − boxplot(x,
data, notch, varwidth, names, main)

Following is the description of the parameters used −

- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Example

We use the data set "mtcars" available in the R environment to create a basic boxplot. Let's look at the columns "mpg" and "cyl" in mtcars.

```
input <- mtcars[,c('mpg','cyl')]
print(head(input))
```

When we execute above code, it produces following result −

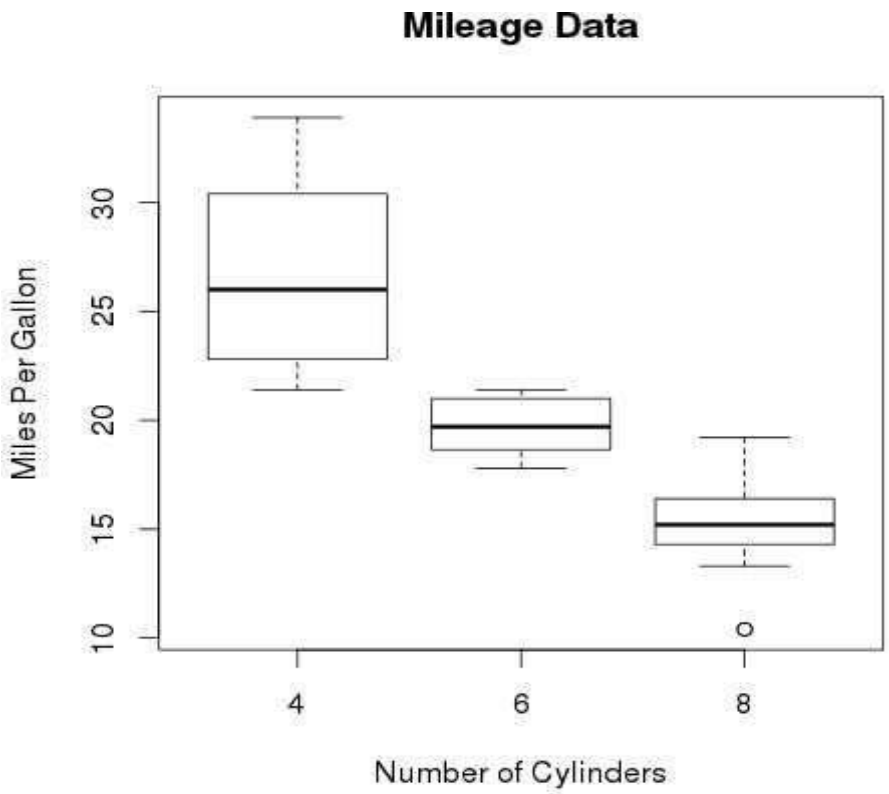
```
mpg  cyl
Mazda RX4         21.0  6
Mazda RX4 Wag     21.0  6
Datsun 710        22.8  4
Hornet 4 Drive    21.4  6
Hornet Sportabout 18.7  8
Valiant           18.1  6
```

Creating the Boxplot

The below script will create a boxplot graph for the relation between mpg (miles per gallon) and cyl (number of cylinders).

```
# Give the chart file a name. png(file
= "boxplot.png")

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
ylab = "Miles Per Gallon", main = "Mileage Data")

# Save the file. dev.off()
```

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



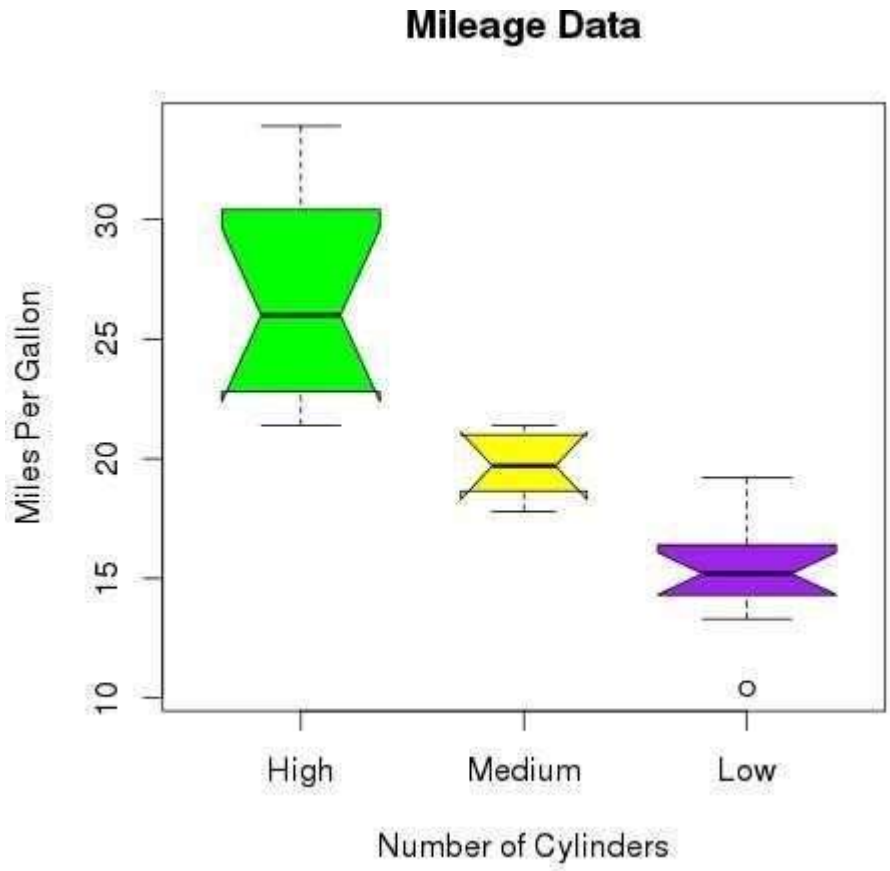
Boxplot with Notch

We can draw boxplot with notch to find out how the medians of different data groups match with each other.

The below script will create a boxplot graph with notch for each of the data group.

```
# Give the chart file a name.
png(file = "boxplot_with_notch.png")

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars,
xlab = "Number of Cylinders",
ylab = "Miles Per Gallon",     main =
"Mileage Data",    notch = TRUE,
varwidth = TRUE,     col =
c("green","yellow","purple"),
   names = c("High","Medium","Low")
)
# Save the file. dev.off()
```

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



R - Histograms

Frequency distribution

in statistics provides the information of the number of occurrences (**frequency**) of distinct values distributed within a **given** period of time or interval, in a list, table, or graphical representation.
Grouped and Ungrouped are two **types** of **Frequency Distribution**.

There are different types of frequency distributions.

- Grouped **frequency distribution**.
- Ungrouped **frequency distribution**.
- Cumulative **frequency distribution**.
- Relative **frequency distribution**.
- Relative cumulative **frequency distribution**.

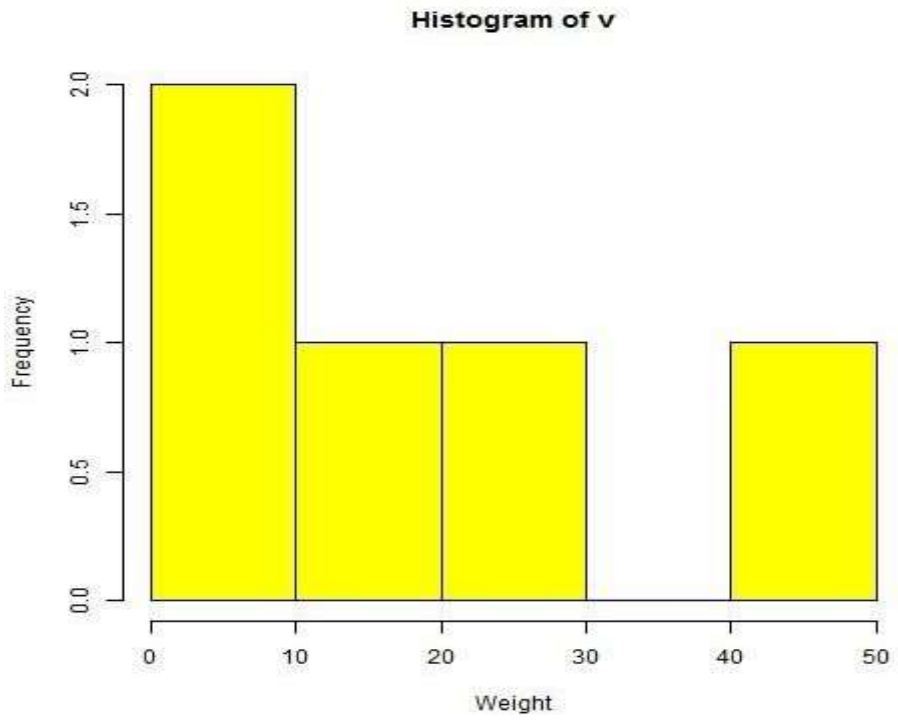
A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.
R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms. Syntax
The basic syntax for creating a histogram using R is − hist(v,main,xlab,xlim,ylim,breaks,col,border)
Following is the description of the parameters used −

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

Example
A simple histogram is created using input vector, label, col and border parameters.
The script given below will create and save the histogram in the current R working directory.

```
# Create data for the graph.
v <-  c(9□,13,21,8,36,22,12,41,31,33,19□)

# Give the chart file a
name. png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

# Save the file.
dev.off()
```

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



Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

The width of each of the bar can be decided by using breaks.

```
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram_lim_breaks.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
breaks = 5)

# Save the file.
dev.off()
```

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

R - Line Graphs

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is − plot(v,type,col,xlab,ylab)

Following is the description of the parameters used −

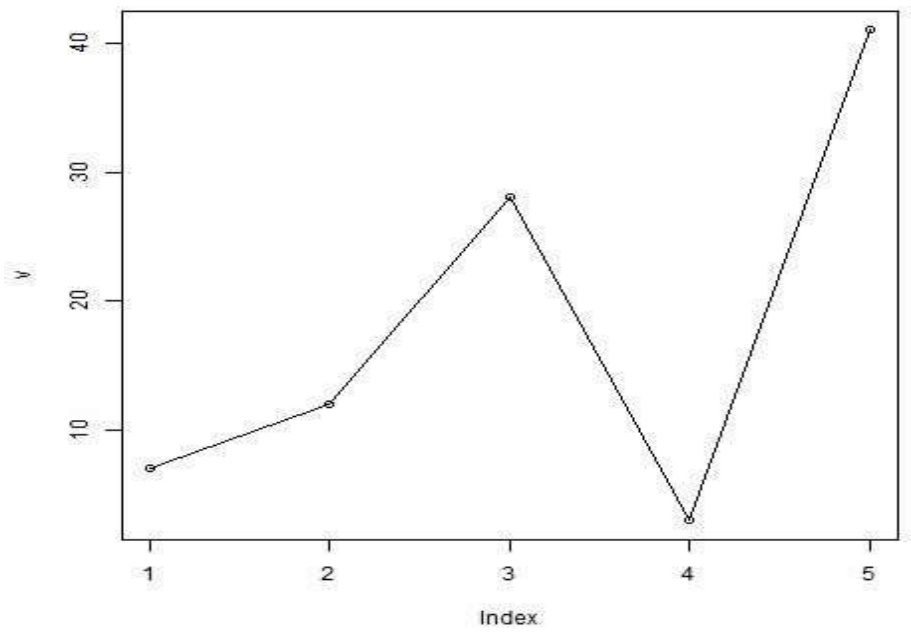
- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

Example

A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory.

```
# Create the data for the
chart. v <- c(7□,12,28,3,41)


# Give the chart file a name.
png(file = "line_chart.jpg")


# Plot the bar chart.
plot(v,type = "o")

# Save the file.
dev.off()
```

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



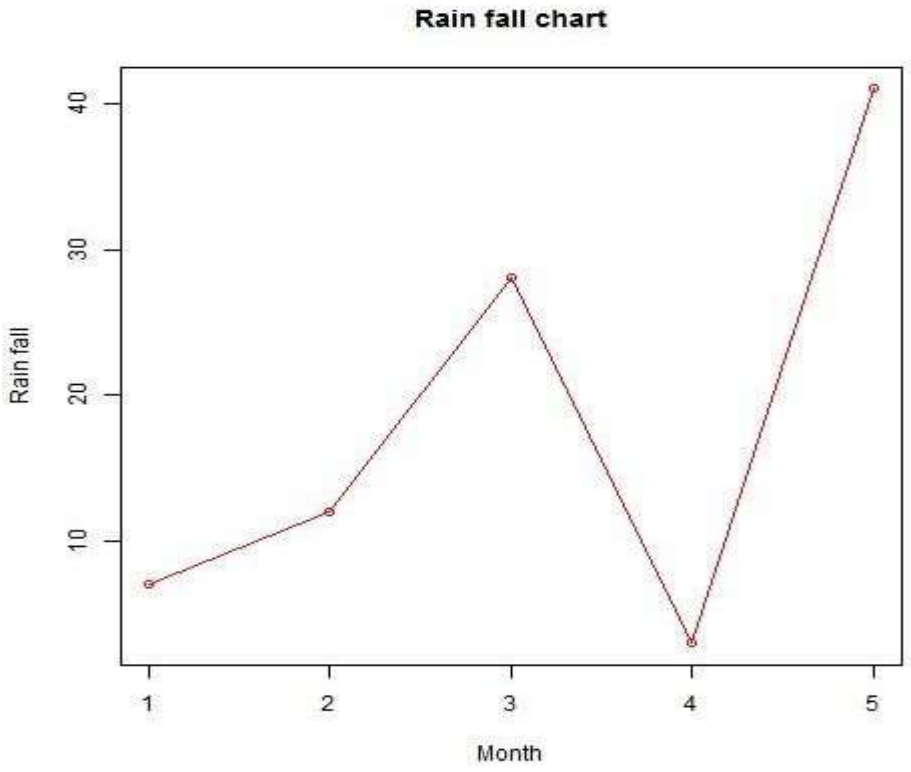
Line Chart Title, Color and Labels

The features of the line chart can be expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.

Example

```
# Create the data for the
chart. v <- c(7□,12,28,3,41)

# Give the chart file a name.
png(file ="line_chart_label_colored.jpg")

# Plot the bar chart.
plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",
main = "Rain fall chart")

# Save the file.
dev.off()
```

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



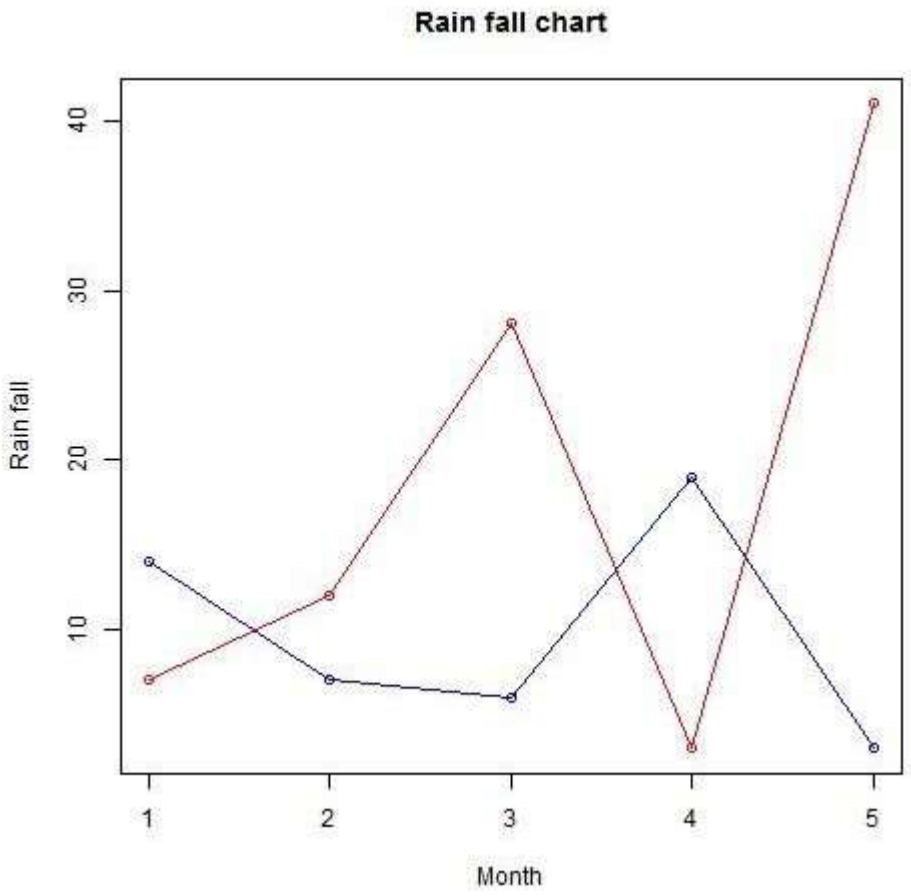
Multiple Lines in a Line Chart

More than one line can be drawn on the same chart by using the **lines()**function.

After the first line is plotted, the lines() function can use an additional vector as input to draw the second line in the chart,

```
# Create the data for the
chart. v <- c(7,12,28,3,41)
t <- c(14,7,6,19,3)

# Give the chart file a name.
png(file =
"line_chart_2_lines.jpg")

# Plot the bar chart.
plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",
main = "Rain fall chart")

lines(t, type = "o", col = "blue")

 # Save the file.
dev.off()
```

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



R - Scatterplots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is − plot(x,

y, main, xlab, ylab, xlim, ylim, axes)

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

Example

We use the data set **"mtcars"** available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

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

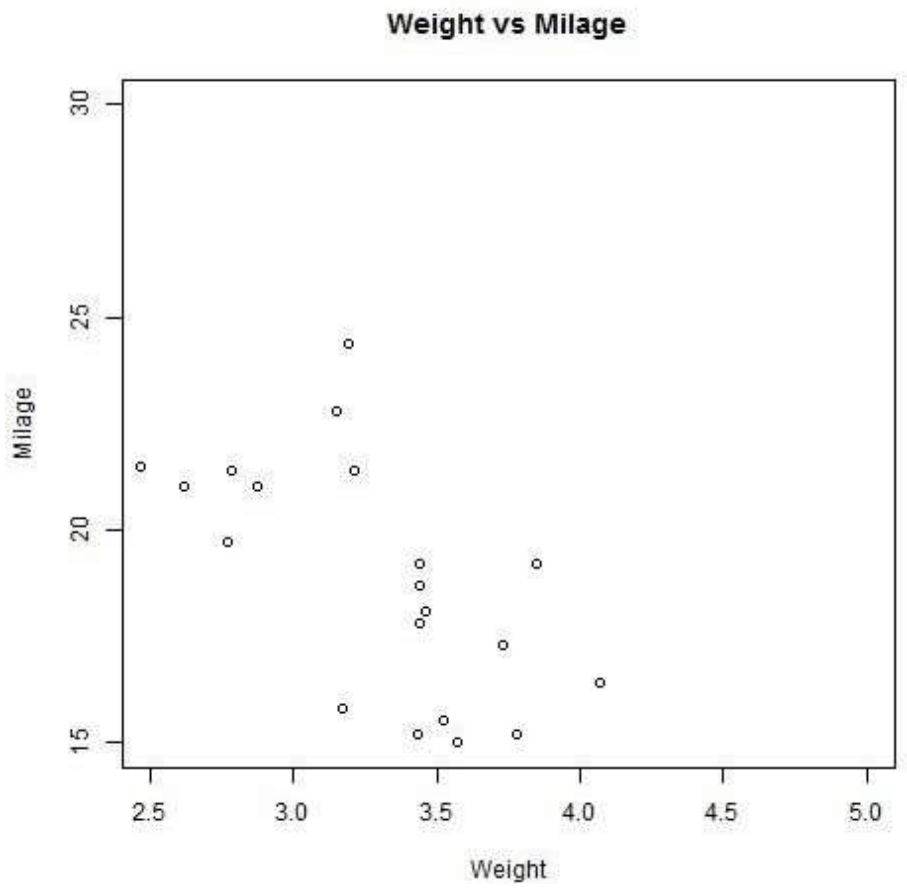
```
wt      mpg
Mazda RX4           2.620  21.0
Mazda RX4 Wag       2.875  21.0
Datsun 7□10         2.320  22.8
Hornet 4 Drive      3.215  21.4
Hornet Sportabout   3.440  18.7□
Valiant             3.460  18.1
```

Creating the Scatterplot

The below script will create a scatterplot graph for the relation between wt(weight) and mpg(miles per gallon).

```
# Get the input values.
input <- mtcars[,c('wt','mpg')]

# Give the chart file a name.
png(file = "scatterplot.png")

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and
30.
plot(x = input$wt,y = input$mpg,    xlab = "Weight",    ylab = "Milage",
   xlim = c(2.5,5),
   ylim = c(15,30),
main = "Weight vs Milage"
)

# Save the file.
dev.off()
```

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



Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots. Syntax

The basic syntax for creating scatterplot matrices in R is − pairs(formula,

data)

Following is the description of the parameters used −

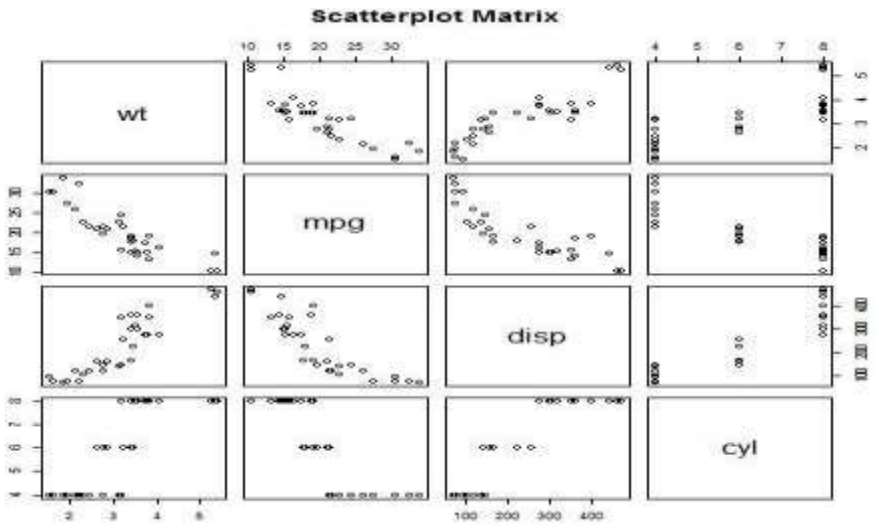
- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

```
# Give the chart file a name.
png(file ="scatterplot_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~wt+mpg+disp+cyl,data = mtcars,
   main = "Scatterplot Matrix")

# Save the file.
dev.off()
```

When the above code is executed we get the following output.



QUESTION:

What is Pi chart, bar graph, line graph and scatter plot?

Ans: Pie Chart: Displays data as a circular chart divided into slices. Shows proportions or percentages.

Example:

```
data <- c(40, 30, 20, 10)
labels <- c("A", "B", "C","D")
pie(data, labels)
```

Bar Graph: Represents data using rectangular bars.

Suitable for comparing categories.

Example:

```
data <- c(10, 20, 30, 40)
barplot(data, names.arg = c("A", "B", "C", "D"))
```

Line Graph: Connects data points with lines. Ideal for showing trends over time.

Example:

```
x <- c(1, 2, 3, 4)
y <- c(10, 15, 25,30)
plot(x, y, type = "l")
```

Scatter Plot: Displays data as points on a graph. Shows the relationship between two variables.

Example:

```
x <- c(1, 2, 3, 4)
y <- c(10, 15, 25,30)
plot(x, y)
```

Sign of Course Teacher

Group B

Government College of Engineering, Jalgaon **(An Autonomous Institute of Government of Maharashtra)**	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.1

AIM:

Write a R program that swaps any two numbers without using any third number.

THEORY:

Algorithm

1. STEP 1: START.
2. STEP 2: ENTER x, y.
3. STEP 3: PRINT x, y.
4. STEP 4: x = x + y.
5. STEP 5: y= x - y.
6. STEP 6: x =x - y.
7. STEP 7: PRINT x, y.
8. STEP 8: END.

Script in R

```
# Simple R program to swap two numbers

x=10
y=20

x=x + y
y=x - y
x=x - y
print(x)
print(y)
```

Output

```
> x=10
> y=20
>
> x=x+y
> y=x-y
> x=x-y
>
> print(x)
[1] 20
> print(y)
[1] 10
```

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.2

AIM:

Write a R program script using for, while and repeat loop that prints the value of i from 1 to 10.

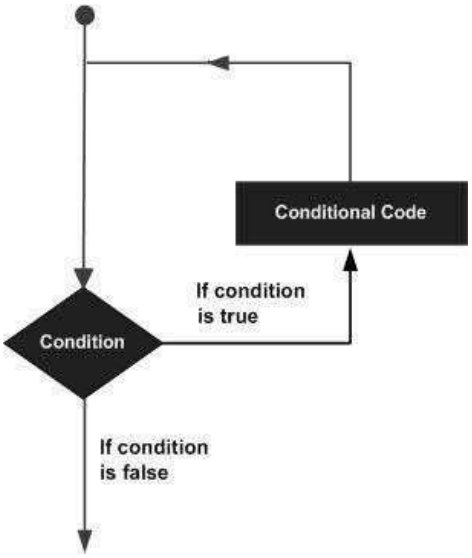
THEORY:

R - Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and the following is the general form of a loop statement in most of the programming languages −



R programming language provides the following kinds of loop to handle looping requirements. Click the following links to check their detail.

Sr.No.	Loop Type & Description
1	repeat loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	for loop Like a while statement, except that it tests the condition at the end of the loop body.

The **Repeat loop** executes the same code again and again until a stop condition is met.

Syntax

The basic syntax for creating a repeat loop in R is −

```
repeat {
commands
if(condition) {
break
  } }
```

The While loop executes the same code again and again until a stop condition is met.

Syntax

The basic syntax for creating a while loop in R is −

```
while (test_expression) {     statement
}
```

A **For loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The basic syntax for creating a **for** loop statement in R is −

```
for (value in vector) {
statements
}
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

R supports the following control statements. Click the following links to check their detail.

Sr.No.	Control Statement & Description
1	break statement Terminates the **loop** statement and transfers execution to the statement immediately following the loop.
2	Next statement The **next** statement simulates the behavior of R switch.

Script in R

```
print("Use of for Loop")
for (i in 1:10)
{
  print(i)
}
print("Use of While Loop")
i<-1
```

```
while(i<11)
{
  print(i)
  i=i+1
}
print("Use of Repeat Loop")
i<-1
repeat
{
  print(i)
 i = i + 1
 if(i > 10)
  {
    break
  }
}
```

Output:

```
> print("Use of for Loop")
[1] "Use of for Loop"
> for (i in 1:10)
+ {
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
> print("Use of While Loop")
[1] "Use of While Loop"
> i<-1
> while(i<11)
+ {
+   print(i)
+   i=i+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
> print("Use of Repeat Loop")
[1] "Use of Repeat Loop"
> i<-1
> # using repeat loop
> repeat
+ {
+   print(i)
+   i = i + 1
+   # checking stop condition
+   if(i > 10)
+   {
+     # using break statement
+     # to terminate the loop
+     break
+   }
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.3

AIM:

Write a R program script to find the factorial of any given number using a recursive Function

THEORY:

R - Functions

Function Definition

An R function is created by using the keyword **function**. The basic syntax of an R function definition is as follows −

```
function_name <- function(arg_1, arg_2, ...) {
  Function body  }
```

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
  for(i in 1:a) {
        b <- i^2
        print(b)
}
}
```

Calling a Function

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
   for(i in 1:a) {
b <- i^2
print(b)
   }
}
```

Call the function new.function supplying 6 as an argument. new.function(6)

```
# Create a function without an argument. new.function
<- function() {
   for(i in 1:5) {
print(i^2)
   }
}

# Call the function without supplying an argument. new.function()
```

```
# Create a function with arguments.
new.function <- function(a,b,c) {
   result <- a * b + c
print(result)
}

# Call the function by position of arguments.
new.function(5,3,11)

# Call the function by names of the arguments.
new.function(a = 11, b = 5, c = 3)
```

```
# Create a function with arguments.
new.function <- function(a = 3, b = 6)
{    result <- a * b
print(result)
}

# Call the function without giving any argument.
new.function()

# Call the function with giving new values of the argument.
new.function(9,5)
```

Script in R

#Find Factorial of a number using recursion

```
recur_factorial <- function(n)
{
  if(n <= 1)
  {
    return(1)
  }
  else
  {
    return(n * recur_factorial(n-1))
  }
}
n = as.integer(readline(prompt = "Enter the Number :"))
a=recur_factorial(n)
print(a)
```

Output:

```
> n = as.integer(readline(prompt = "Enter the Number :"))
Enter the Number :6
> a=recur_factorial(n)
> print(a)
[1] 720
>
```

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.4

AIM:

Write a R program that reads the csv file. Find the maximum and minimum values among all three.

THEORY:

R - CSV Files

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

Input as CSV File

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input.csv**.

Reading a CSV File

Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory −

```
data <- read.csv("input.csv") print(data)
```

Analyzing the CSV File

By default the **read.csv()** function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```
data <- read.csv("input.csv")

print(is.data.frame(data)) print(ncol(data))
print(nrow(data))
```

Get the maximum salary

```
# Create a data frame. data
<- read.csv("input.csv")

# Get the max salary from data
frame. sal <- max(data$salary)
print(sal)
```

Writing into a CSV File

R can create csv file form existing data frame. The **write.csv()** function is used to create the csv file. This file gets created in the working directory.

```
# Create a data frame. data
<- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```

Script R

#4 Load the data and find min, max value of the attribute

m<-read.csv("mtcars.csv", header=TRUE)

View(m)

str(m)

min(m$mpg)

max(m$mpg)

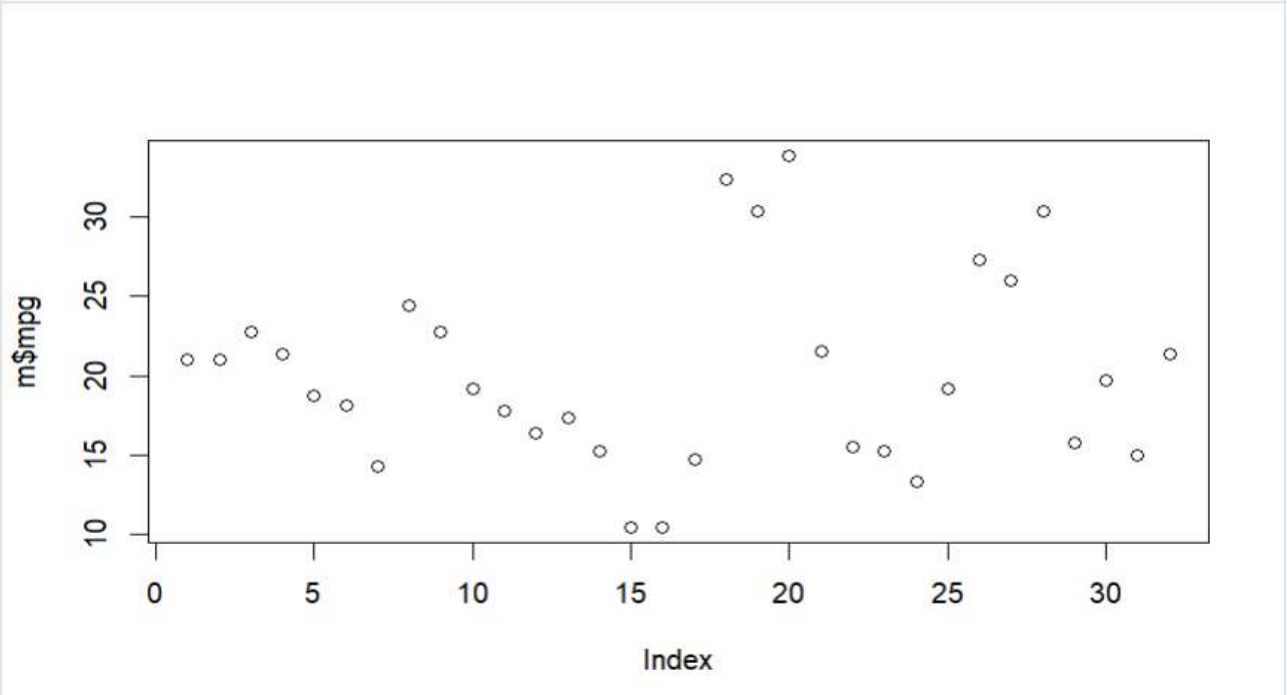
mean(m$mpg)

mode(m$mpg)

plot(m$mpg)

Output:

```
> m<-read.csv("mtcars.csv", header=TRUE)
> View(m)
> str(m)
'data.frame':   32 obs. of  12 variables:
 $ X   : chr  "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : int  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : int  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : int  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : int  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: int  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: int  4 4 1 1 2 1 4 2 2 4 ...
> min(m$mpg)
[1] 10.4
> max(m$mpg)
[1] 33.9
> mean(m$mpg)
[1] 20.09062
> mode(m$mpg)
[1] "numeric"
> plot(m$mpg)
>
```



Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.5

AIM: Using the various built functions plot pie chart, scatter plot, histogram and line charts

THEORY:

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is − pie(x,

labels, radius, main, col, clockwise)

Following is the description of the parameters used −

- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.
x <- c(21, 6□2, 10, 53)
labels <- c("London", "New York",
"Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city.png")

# Plot the chart. pie(x,labels)

# Save the file.
dev.off()
```

Pie Chart Title and Colors

We can expand the features of the chart by adding more parameters to the function. We will use parameter **main** to add a title to the chart and another parameter is **col** which will make use of rainbow colour pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use length(x).

Example

The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city_title_colours.jpg")

# Plot the chart with title and rainbow color pallet.
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

# Save the file.
dev.off()
```

Slice Percentages and Chart Legend

We can add slice percentage and a chart legend by creating additional chart variables.

```
# Create data for the graph.
x <- c(21, 6□2, 10,53)
labels <- c("London","New York","Singapore","Mumbai")

piepercent<- round(100*x/sum(x), 1)

# Give the chart file a name.
png(file = "city_percentage_legends.jpg")

# Plot the chart.
pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
fill = rainbow(length(x)))

# Save the file.
dev.off()
```

3D Pie Chart

A pie chart with 3 dimensions can be drawn using additional packages. The package **plotrix** has a function called **pie3D()** that is used for this.

```
# Get the library. library(plotrix)

# Create data for the graph.
x <- c(21, 62, 10,53)
lbl <- c("London","New York","Singapore","Mumbai")

# Give the chart file a name.
png(file = "3d_pie_chart.jpg")

# Plot the chart.
pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")

# Save the file.
dev.off()
```

R - Scatterplots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is –

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

Example

We use the data set **"mtcars"** available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

Creating the Scatterplot

The below script will create a scatterplot graph for the relation between wt(weight) and mpg(miles per gallon).

```
# Get the input values.
input <- mtcars[,c('wt','mpg')]

# Give the chart file a name.
png(file = "scatterplot.png")

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
```

```
xlab = "Weight",    ylab
= "Milage",
   xlim = c(2.5,5),
   ylim = c(15,30),          main =
"Weight vs Milage"
)

# Save the file.
dev.off()
```

Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is − pairs(formula,

data)

Following is the description of the parameters used −

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

```
# Give the chart file a name. png(file
= "scatterplot_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~wt+mpg+disp+cyl,data = mtcars,
   main = "Scatterplot Matrix")

# Save the file.
dev.off()
```

R - Histograms

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is − hist(v,main,xlab,xlim,ylim,breaks,col,border)

Following is the description of the parameters used −

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

Example

A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.

```
# Create data for the graph. v <-  c(9□,13,21,8,36,22,12,41,31,33,19□) #

Give the chart file a name. png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")
```

```
# Save the file.
dev.off()
# Create data for the graph.
v <-c(9□,13,21,8,36,22,12,41,31,33,19□)

# Give the chart file a name.
png(file = "histogram_lim_breaks.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
breaks = 5)

# Save the file. dev.off()
```

R - Line Graphs

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is − plot(v,type,col,xlab,ylab)

Following is the description of the parameters used −

- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

Example

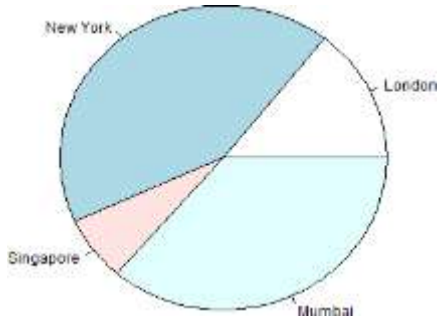
A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory.

```
# Create the data for the chart.
v <- c(7,12,28,3,41)

# Give the chart file a name.
png(file = "line_chart.jpg")

# Plot the bar chart.
plot(v,type = "o")

# Save the file.
dev.off()
```

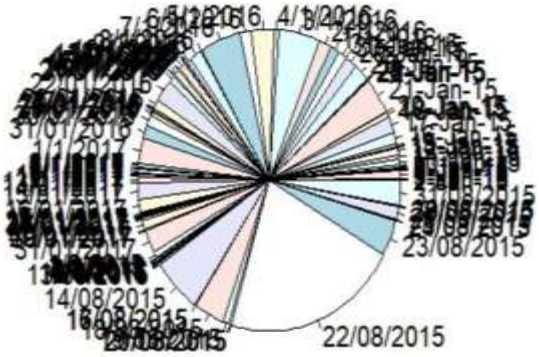
```
# Create the data for the chart.
v <- c(7,12,28,3,41)

# Give the chart file a name.
png(file ="line_chart_label_colored.jpg")

# Plot the bar chart.
plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",
main = "Rain fall chart")

# Save the file.
dev.off()
```

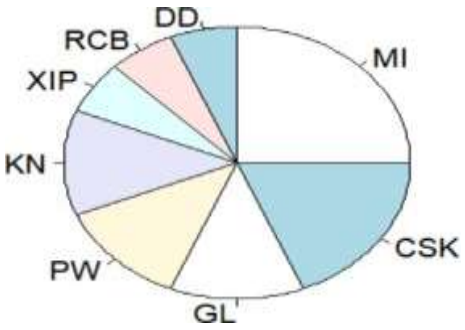
```
# Create the data for the chart.
v <- c(7,12,28,3,41)
t <- c(14,7,6,19,3)

# Give the chart file a name.
png(file = "line_chart_2_lines.jpg")

# Plot the bar chart.
plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",
main = "Rain fall chart")

lines(t, type = "o", col = "blue")

# Save the file.
dev.off()
```

Outputs:

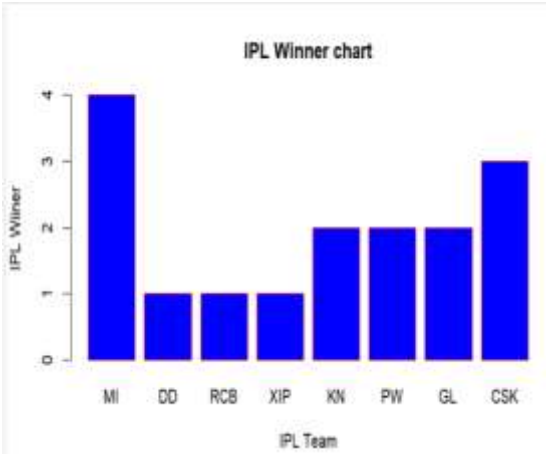

New York
London
Singapore
Mumbai

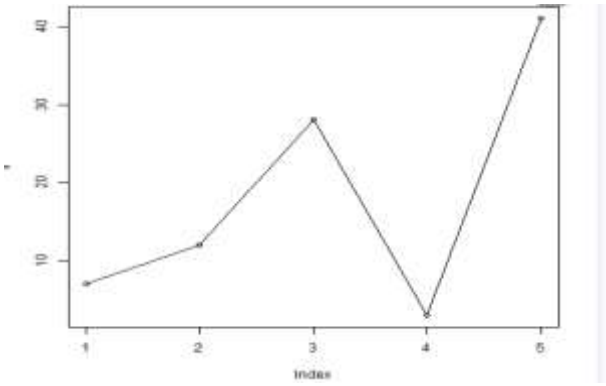

City pie chart
New York
London
Singapore
Mumbai

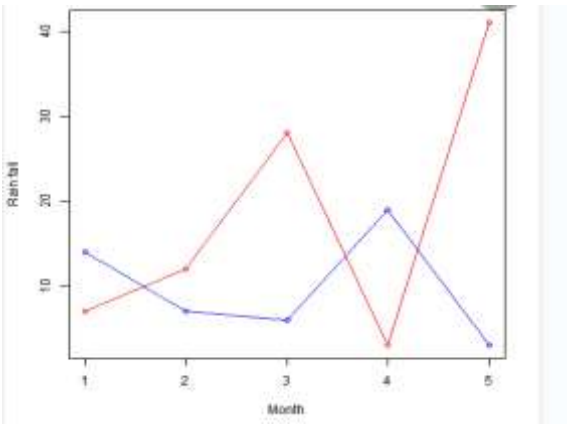

23/08/2015
14/08/2015
22/08/2015


DD
RCB
MI
XIP
KN
CSK
PW
GL


Pie Chart of Countries
New York
London
Singapore
Mumbai


IPL Winner chart
IPL Winner
0
1
2
3
4
MI
DD
RCB
XIP
KN
PW
GL
CSK
IPL Team





Sign of Course Teacher

Group C

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.1

AIM: For Iris dataset visualize data using plot() also perform filter(), select(), mutate(), arrange() functions

THEORY:

filter() allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame. For example, we can select all flights on January 1st with: select() will select only the desirable variables mutate() function **Use :**Creates new variables arrange() orders the rows of a data frame by the values of selected columns.

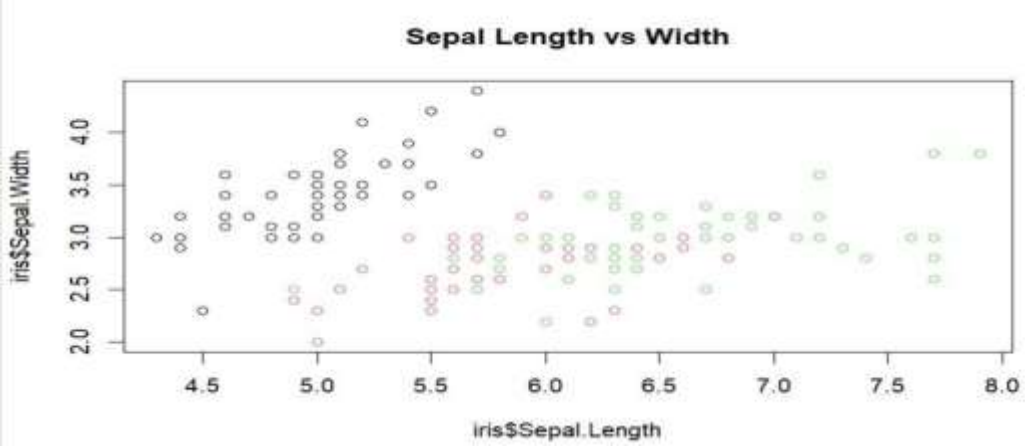
Script R:

```
View(iris)
df <- read.csv("irisdata.csv", header = TRUE)
View(df)
barplot(df$Sepal.Length, main = "Iris Statts",  xlab =
"Sepal Length", ylab = "Value", col="green")
barplot(df$Sepal.Width, main = "Iris Statts",  xlab =
"Sepal Width", ylab = "Value", col="orange")
barplot(df$Petal.Length, main = "Iris Statts",  xlab =
"Petal Length", ylab = "Value", col="red")
barplot(df$Petal.Width, main = "Iris Statts",  xlab =
"Petal Width", ylab = "Value", col="blue")
```

```
names(iris) <- tolower(names(iris))
names(iris)
library(dplyr)
virginica <- filter(iris, species == "virginica")
virginica
head(virginica)
sepalLength6 <- filter(iris, species == "virginica",
sepal.length > 6□)
sepalLength6□
tail(sepalLength6□)
selected <- select(iris, sepal.length, sepal.width,
petal.length)
selected
selected2 <- select(iris, sepal.length:petal.length)
head(selected, 3)
a<-identical(selected, selected2)
a
newCol <- mutate(iris, greater.half = sepal.width > 0.5
* sepal.length)
tail(newCol)
newCol
newCol <- arrange(newCol, petal.width)
head(newCol)
newCol
```

Output:

```
> source("~/.active-rstudio-document")
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
  Sepal.Length Sepal.Width
1          5.1         3.5
2          4.9         3.0
3          4.7         3.2
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Area
1          5.1         3.5          1.4         0.2  setosa      17.85
2          4.9         3.0          1.4         0.2  setosa      14.70
3          4.7         3.2          1.3         0.2  setosa      15.04
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          7.9         3.8          6.4         2.0 virginica
2          7.7         3.8          6.7         2.2 virginica
3          7.7         2.6          6.9         2.3 virginica
```



Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.2

AIM:

Write a R program that will identify and remove the missing values from datasets using frequency mean, median or mode options.

THEORY:

Missing values in data science arise when an observation is missing in a column of a data frame or contains a character value instead of numeric value. Missing values must be dropped or replaced in order to draw correct conclusion from the data.

#identify and remove the missing values from datasets using frequency mean, median or mode options.

Script R:

```
age <- c(23, 16, NA)

mean(age)

mean(age, na.rm = TRUE)

a<-read.csv("123.csv")

View(a)

mean(a$age)

#The complete.cases function detects rows in a

data.frame that do not contain any missing value.

complete.cases(a)

# na.omit is used to remove incompete recores
```

```
b<-na.omit(a)

View(b)

is.special <- function(a){

  if (is.numeric(a)) !is.finite(a) else is.na(a)

}

a

#sapply and is.special check missing values only

sapply(a, is.special)
```

Output:

```
> setwd("C:/Users/prati/OneDrive/Desktop/RPL/R P L")
> age <- c(23, 16, NA)
> mean(age)
[1] NA
> mean(age, na.rm = TRUE)
[1] 19.5
> a<-read.csv("123.csv")
> View(a)
> mean(a$age)
[1] 24.625
> #The complete.cases function detects rows in a data.frame
> complete.cases(a)
[1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
> # na.omit is used to remove incompete recores
> b<-na.omit(a)
> View(b)
> is.special <- function(a){
+   if (is.numeric(a)) !is.finite(a) else is.na(a)
+ }
> a
  age height
1  21    6.0
2  42    5.9
3  18    5.7
4  21     NA
5  22    5.3
6  24    5.0
7  30     NA
8  19    5.0
> #sapply and is.special check missing values only
> sapply(a, is.special)
       age height
[1,] FALSE  FALSE
[2,] FALSE  FALSE
[3,] FALSE  FALSE
[4,] FALSE   TRUE
[5,] FALSE  FALSE
[6,] FALSE  FALSE
[7,] FALSE   TRUE
[8,] FALSE  FALSE
```

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.3

AIM: Write a R program that will identify outliers and remove outliers from dataset

THEORY:

An **outlier** is a value or an **observation that is distant from other observations**, that is to say, a data point that differs significantly from other data points. Enderlein (1987□) goes even further as the author considers outliers as values that deviate so much from other observations one might suppose a different underlying sampling mechanism.

Script R:

```
#9: R program that will identify outliers and remove outliers from dataset
#boxplot.stats list the outlier
x <- c(1:10, 20, 30, 100)
x
y<-boxplot.stats(x)$ou
y

b<-boxplot.stats(x, coef = 2)$out
b

b<-boxplot.stats(x, coef = 1)$out
b
#which() function it is possible to extract the row number corresponding to these outliers:
out_ind <- which(x %in% c(b))
out_ind
```

Output:

```
> #9: R program that will identify outliers and remove outliers from
> #boxplot.stats list the outlier
> x <- c(1:10, 20, 30, 100)
> x
 [1]   1   2   3   4   5   6   7   8   9  10  20  30 100
> y<-boxplot.stats(x)$ou
> y
[1]  20  30 100
>
> b<-boxplot.stats(x, coef = 2)$out
> b
[1]  30 100
>
> b<-boxplot.stats(x, coef = 1)$out
> b
[1]  20  30 100
>
> #which() function it is possible to extract the row number corresp
> out_ind <- which(x %in% c(b))
> out_ind
[1] 11 12 13
>
```

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.4

AIM: Using lm() function, perform linear regression on the dataset.

THEORY: R - Linear Regression

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

The general mathematical equation for a linear regression is − y = ax + b

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is − lm(formula,data)

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152,
131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

print(relation)
```

When we execute the above code, it produces the following result − Call:

lm(formula = y ~ x)

Coefficients:

(Intercept)	x
-38.4551	0.6746

Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

print(summary(relation))
```

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

Call: lm(formula = y ~ x)

Residuals:

Min	1Q	Median	3Q	Max
-6.3002	-1.6629	0.0412	1.8944	3.9775

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509  8.04901  -4.778  0.00139 **
x             0.67461          0.05191   12.997  1.16e-06*** ---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,     Adjusted R-squared:  0.9491
F-statistic: 168.9  on 1 and 8 DF,  p-value: 1.164e-06 predict() Function
```

Syntax

The basic syntax for predict() in linear regression is − predict(object,

newdata)

Following is the description of the parameters used −

- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
x <- c(151, 174, 138, 186,
128, 136, 179, 163, 152,
131)

# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72,
62,8)

# Apply the lm() function.
relation <- lm(y~x)

# Find weight of a person
with height 170.
a <- data.frame(x = 170)
result                    <-
predict(relation,a)
print(result)
```

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

```
       1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131) y <-
c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

# Give the chart file a name.
png(file = "linearregression.png")

# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.
dev.off()
```

Script in R :

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

 y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

print(relation)

print(summary(relation))

#predict() Function

# Find weight of a person with height 170. a

<- data.frame(x = 170)

result <-  predict(relation,a)

print(result)

#Visualize the Regression Graphically
```

Give the chart file a name.

```
png(file = "linearregression.png")

# Plot the chart.

plot(y,x,col = "blue",main = "Height & Weight Regression",

    abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm") #
```

Save the file.

```
dev.off()
```

Output:

```
> source("~/linearregression.R")

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -38.4551       0.6746


Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3002 -1.6629  0.0412  1.8944  3.9775


Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06

       1
76.22869
```

Sign of Course Teacher

Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)	
Name :	**PRN**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO358U R Programming Lab	**Course Teacher** : Mr. Sandip Patil
Date of Performance :	**Date of Completion :**

Practical no.5

AIM:

Write a R script to predict classification of values using decision trees

THEORY:

R - Decision Tree

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions. It is mostly used in Machine Learning and Data Mining applications using R.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

install.packages("party")

The package "party" has the function **ctree()** which is used to create and analyze decison tree.

Syntax

The basic syntax for creating a decision tree in R is − ctree(formula,
data)

Following is the description of the parameters used −

- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.

Input Data

We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

Here is the sample data.

```
# Load the party package. It will automatically load other
# dependent packages.
library(party)

# Print some records from data set readingSkills.
print(head(readingSkills))
```

When we execute the above code, it produces the following result and chart −

```
nativeSpeaker  age  shoeSize       score 1          yes     5  24.83189□
32.29385
2            yes    6      25.95238     36.63105
3            no    11      30.42170      49.60593
4            yes    7       28.66450    40.28456
5            yes   11     31.88207     55.46085
6            yes   10     30.07843    52.83124
Loading required package: methods
Loading required package: grid
.....
.....
```

Example

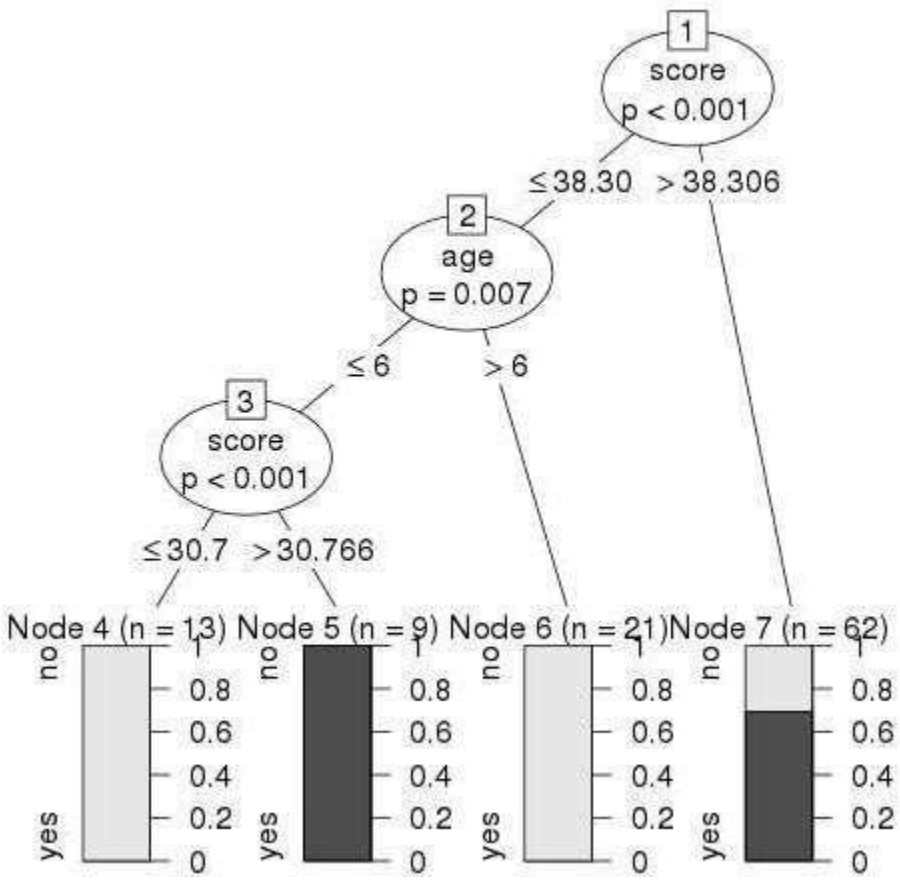
We will use the **ctree()** function to create the decision tree and see its graph.

```
# Load the party package. It will automatically load other #
dependent packages.
library(party)

# Create the input data frame.
input.dat <- readingSkills[c(1:105),]

# Give the chart file a name. png(file
= "decision_tree.png")

# Create the tree.
  output.tree <- ctree(   nativeSpeaker ~
age + shoeSize + score,
data = input.dat)

# Plot the tree. plot(output.tree)

# Save the file. dev.off()
```

When we execute the above code, it produces the following result − null device

```
        1
Loading required package: methods
Loading required package: grid
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
Loading required package: strucchange
Loading required package: zoo
Attaching package: ‘zoo’
The following objects are masked from ‘package:base’:
   as.Date, as.Date.numeric
```

Loading required package: sandwich



Script in R

```
library(party)
print(head(readingSkills))
#We will use the ctree() function to create the decision tree and see its graph.
readingSkills
write.csv(readingSkills, "readingskills.csv")

data1<-read.csv("readingskills.csv")

input.dat <- readingSkills[c(1:105),]
# Give the chart file a name.
```

```
png(file = "decision_tree.png")

#ctree(formula, data)

# Create the tree.
output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.dat)

# Plot the tree.
plot(output.tree)

# Save the file.
dev.off()
```

Output:

```
> source("~/Decsiontree.R")
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
```

Sign of Course Teacher