

# SYLLABUS

## **List of Experiments:**

1. Implement all basic R commands.
2. Interact data through .csv files (Import from and export to .csv files).
3. Get and Clean data using swirl exercises. (Use 'swirl' package, library and install that topic from swirl).
4. Visualize all Statistical measures (Mean, Mode, Median, Range, Inter Quartile Range etc., using Histograms, Boxplots and Scatter Plots).
5. Create a data frame with the following structure.

EMP ID	EMP NAME	SALARY	START DATE
1	Satish	5000	01-11-2013
2	Vani	7500	05-06-2011
3	Ramesh	10000	21-09-1999
4	Praveen	9500	13-09-2005
5	Pallavi	4500	23-10-2000

- a) Extract two column names using column name.
  - b) Extract the first two rows and then all columns.
  - c) Extract 3 rd and 5th row with 2nd and 4th column.
6. Write R Program using 'apply' group of functions to create and apply normalization function on each of the numeric variables/columns of iris dataset to transform them into
    - i) 0 to 1 range with min-max normalization.
    - ii) a value around 0 with z-score normalization.
  7. Create a data frame with 10 observations and 3 variables and add new rows and columns to it using 'rbind' and 'cbind' function.
  8. Write R program to implement linear and multiple regression on 'mtcars' dataset to estimate the value of 'mpg' variable, with best R<sup>2</sup> and plot the original values in 'green' and predicted values in 'red'.
  9. Implement k-means clustering using R.
  10. Implement k-medoids clustering using R.
  11. Implement density based clustering on iris dataset.

12. Implement decision trees using 'readingSkills' dataset.
13. Implement decision trees using 'iris' dataset using package party and 'rpart'.
14. Use a Corpus() function to create a data corpus then Build a term Matrix and Reveal word frequencies.

# **DATA MINING TECHNIQUES WITH R LAB (IT3104)**

B.Tech 3/4, V-SEMESTER


## **COURSE OUTCOMES**

<b>S.No</b>	<b>Course Code - CO</b>	<b>Course Outcomes</b>	<b>Blooms Taxonomy</b>
1	CO1	Extend the functionality of R by using add-on packages	Application
2	CO2	Extract data from files and other sources and perform various data manipulation tasks on them	Application
3	CO3	Code statistical functions in R	Application
4	CO4	Use R Graphics and Tables to visualize results of various statistical operations on data	Application
5	CO5	Apply the knowledge of R gained to data Analytics for real life applications	Application
6	CO6	Apply clustering techniques for data analysis using R software	Application

## **GENERAL INSTRUCTIONS**

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
  - Laboratory observation notes.
  - Laboratory Record updated up to the last session experiments.
  - Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
8. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

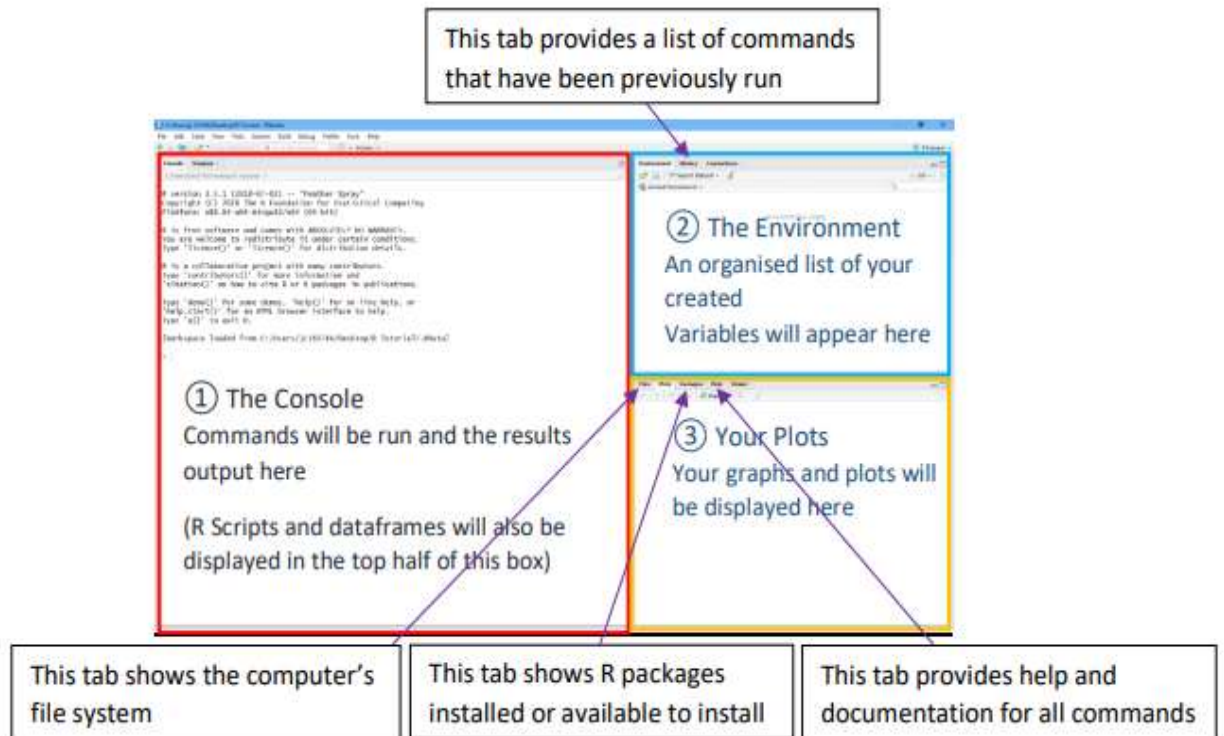
## **TOOLS USED IN THE DATA MINING TECHNIQUES WITH R LAB**

<b>IDE</b>	
<b>LANGUAGE</b>	

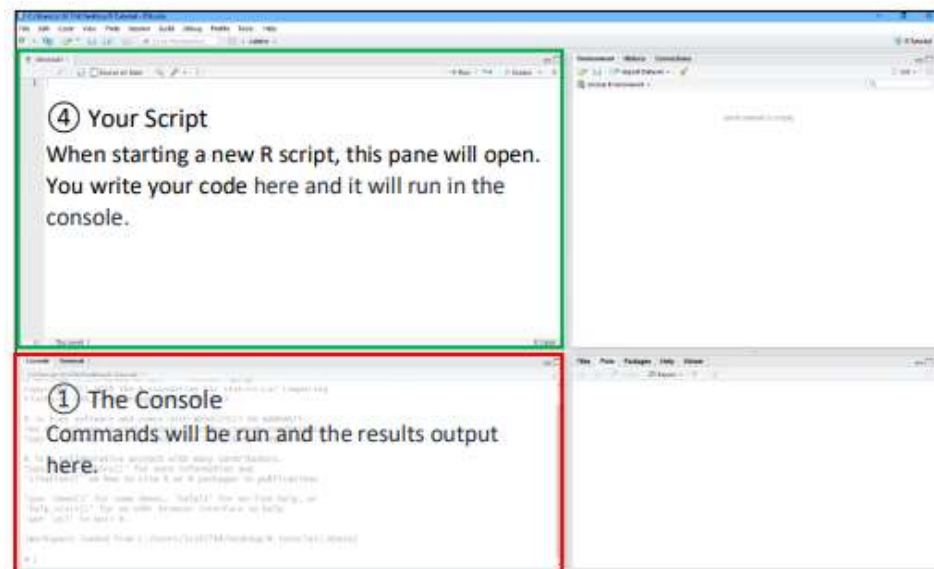
# 1. Setting Up R-Studio

## The R Studio Interface

When opening R Studio, you will be presented with the following screen. The colors and fonts may vary depending on your Operating System but the layout will be the same.

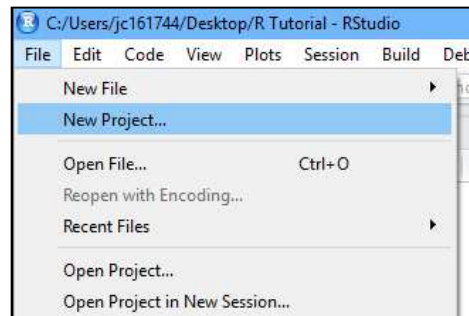


On creating a new R script (see next section), the script panel will open. You will write your Rcode/script here and it will be run in the console.

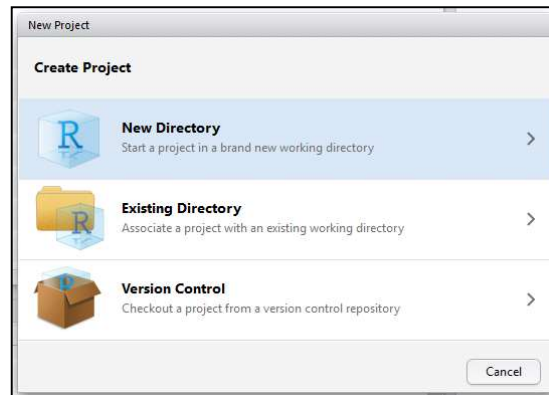


## Start a New Project in R

When starting to work with a new dataset, a New Project should be created.



Creating a New Directory makes a default working directory and a logical place to store all associated files such as raw data spreadsheets.



Any associated excel documents or text files can be saved into this new folder and easily accessed from within R. You can then perform data analysis or produce visualizations with your imported data.

## Experiment 1

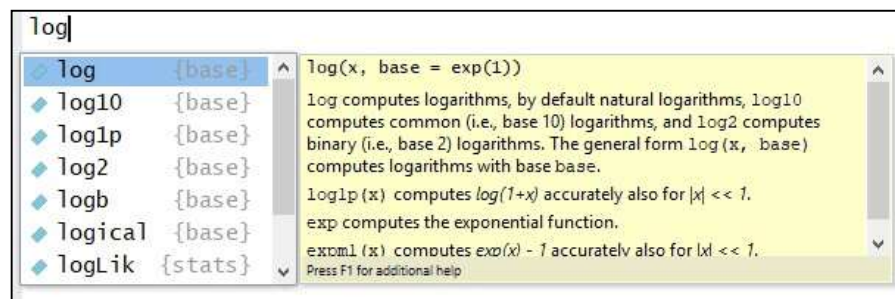
Implement all basic R commands

### Solution:

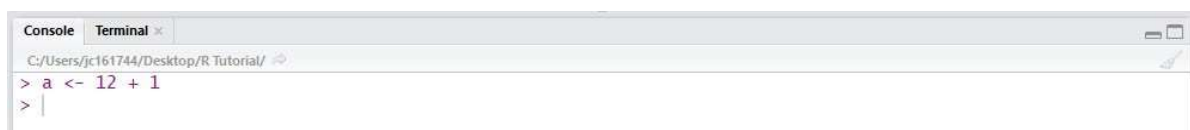
#### Basic Commands in R Studio

##### Run a Command

Commands in R can be entered directly into the console at the bottom left of the screen. Entering 3 or more characters of a command into the console or a script will open the suggested command menu. This menu suggests commands or the names of variables you have intended to type, alongside a description and suggested use. Entering ? followed by any R command will open a help page in the help tab found in the bottom right hand corner of the screen (eg. ?log10 opens the log10 help page). This help page will offer settings and formatting for each command, as well as an example.



On completing a command and pressing enter, R will immediately run the code, print the output and move to a new line. Using the ↑ key will repeat the last command entered into the console.



At its heart, R is a calculator and can accept any mathematical calculation directly into the console. The following table represents just a few of the available mathematical operators:

R Command	Mathematical Operator	Example
*	x (multiply)	12 * 4 = 48
/	÷ (divide)	15 / 3 = 5
^	(exponent/power)	5^2 = 5 <sup>2</sup> = 25



$\log_{10}(x)$	$\log_{10}x$	$\log_{10}(1000) = 3$
$\log(x, \text{base})$	$\log_{\text{base}}x$	$\log(40,3) = 3.357763$
$\text{sum}(a,b,c)$	$a + b + c$	$\text{sum}(31,24,12) = 67$

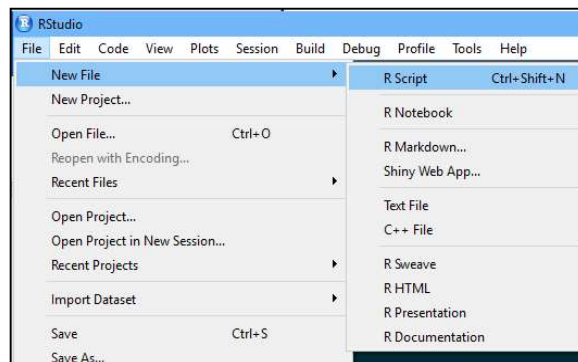
***Directions:***

Enter the following commands into the console, pressing enter after every line:

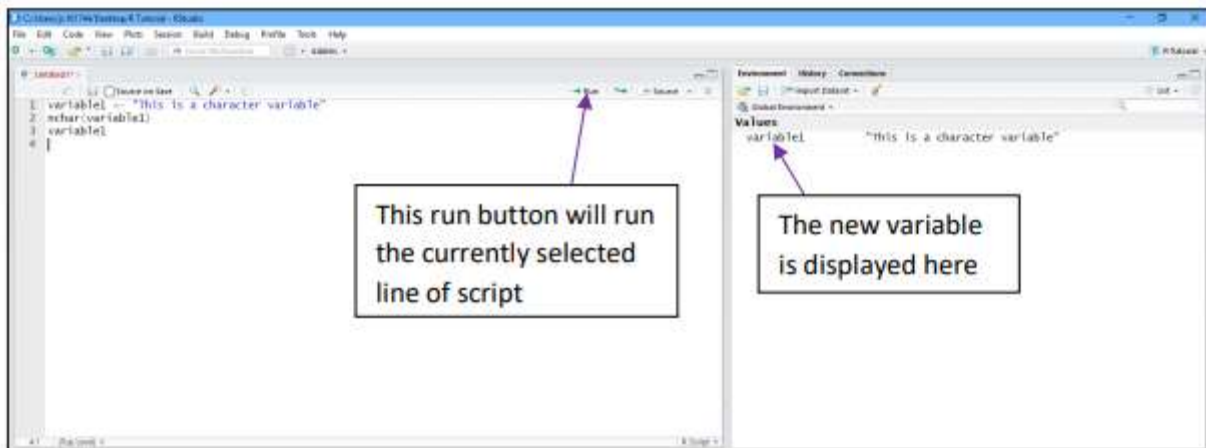
1.  $(4^3+10)/2$
2.  $\log_{10}(75+2^3)$
3.  $\text{sum}(23,45,56,12)$
4.  $\text{sum}(\log(45,10), 2^3, 10-4, 25/5)$

**Using R Scripts**

Commands can be run directly from the console, but creating an R Script allows you to edit and reuse previous commands and to create more complicated lists of commands. A script also allows you to save your commands to be reopened later.

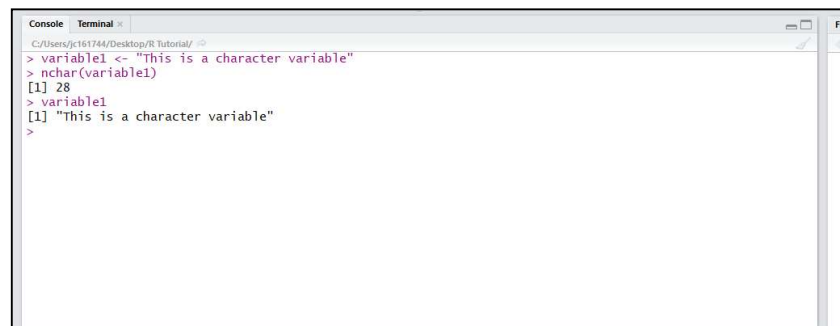


Multiple commands can be entered into a script, one after the other across multiple lines.



The script above creates a new variable called variable1 with the value “This is a character variable”. The next command counts the number of characters in the variable. The final command returns the value of variable1.

To run the script one line at a time, navigate the cursor to the appropriate line and press CTRL + Enter. To run all commands from the start, press CTRL + Shift + Enter.



### Directions:

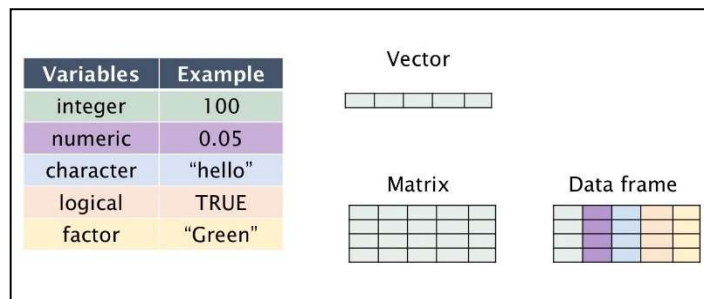
1. Create a new script
2. Enter the following code into your script:
 

```
var1 <- c(45,32,23,78,99,10)
var1
mean(var1)
var1 + 4
```
3. Run your code (Ctrl + Shift + Enter)
4. Save your script

## 2. Getting Data into R-Studio

### Getting Data into R

**Variables** can be thought of as a labelled container used to store information. Variables allow us to recall saved information to later use in calculations. Variables can store many different things in R studio, from single values to tables of information, images and graphs.



### Assigning a Value to a Variable:

Storing a value or “assigning” it to a variable is completed using either `<-` or `=` function. The name given to a variable should describe the information or data being stored. This helps when revisiting old code or when sharing it with others.

A screenshot of the R Studio code editor. A box labeled "Variable Name" points to the first line of code. The code is as follows:

```
1 first_number <- 1
2 a_character <- "a"
3 some_logic <- TRUE
4 a_factor <- "Red"
5 |
```

### Creating a Vector

A variable can hold one value or many values. A vector is used to store more than one value of the same type. The combine or `c()` function (type `?c` in the console for the further information) allows us to combine them into a single list of values.

### *Example:*

A class of 10 students have been surveyed and their heights recorded:

150cm, 150cm, 142cm, 154cm, 168cm, 153cm, 151cm, 153cm, 142cm and 151cm

To begin analyzing this data in R, the values must be stored in a variable. In the following, the variable *height* has been created to contain our values:

```
Console Terminal
C:/Users/jc161744/Desktop/R Tutorial/
> height <- c(150, 150, 142, 154, 168, 153, 151, 153, 142, 151)
> height
[1] 150 150 142 154 168 153 151 153 142 151
>
```

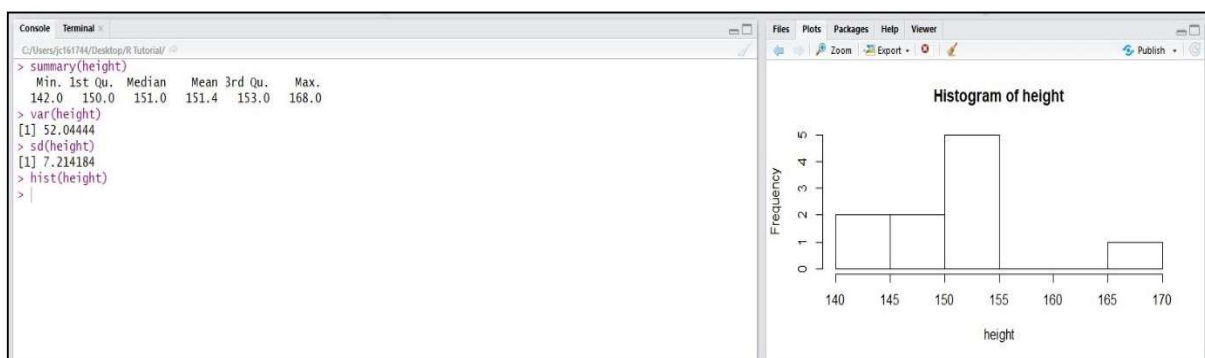
The vector can then be recalled using the variable name *height*. Vectors for colour (textstrings) and survey (logical values) have also created.

```
Console Terminal
C:/Users/jc161744/Desktop/R Tutorial/
> colour
[1] "red" "green" "blue" "blue" "red" "green" "red" "green" "blue" "red"
> survey
[1] TRUE FALSE FALSE TRUE FALSE TRUE TRUE FALSE TRUE TRUE
>
```

```
Console Terminal
C:/Users/jc161744/Desktop/R Tutorial/
> #To request the first value in the vector
> height[1]
[1] 150
>
> #To request the last value
> height[10]
[1] 151
>
> #To request all values between the 3rd and 7th place in the vector
> height[3:7]
[1] 142 154 168 153 151
>
```

To recall a single value from the vector, use the variable name followed by the position of the value in the list and surrounded by [ ] brackets.

Useful statistical analysis can be performed on single set of values using vectors:



## Combining Vectors into a Dataframe

A dataframe will often contain more than 1 variable of interest. To combine vectors of the same length together into a table or “dataframe” (similar to an excel table), we use the `data.frame()` command. Below we have combined 3 vectors (colour, height and survey), containing 10 values each, together into a dataframe. The table has been assigned to the variable *table*.

Columns

Header row

Values

```
> table <- data.frame(colour, height, survey)
> table
  colour height survey
1    red    150   TRUE
2  green    150  FALSE
3   blue    142  FALSE
4   blue    154   TRUE
5    red    168  FALSE
6  green    153   TRUE
7    red    151   TRUE
8  green    153  FALSE
9   blue    142   TRUE
10   red    151   TRUE
```

The top line of the dataframe is called the header row and contains a descriptive name for the values in each column.

Below the header row, values can be referred to individually or in whole rows and columns. To retrieve a single value from a dataframe assigned to a variable, the variable name is used then followed by the coordinates of the value within [row, column] square brackets (eg. `table[2,1]` returns green and `table[3,2]` returns 142). Whole rows can be returned individually using coordinates (eg. `table[3,]` returns all values for row 3). Single columns can be returned by either referring to them using a coordinate with the row blank (eg. `table[,3]` returns all values in 3rd column), or by using the variable name followed by a `$` and the column header (eg. `table$height` returns all “height” values from the dataframe “table”).

```
> table[2,1]
[1] green
Levels: blue green red
> table[3,2]
[1] 142
> table[,3]
[1] TRUE FALSE FALSE TRUE FALSE TRUE TRUE FALSE TRUE TRUE
> table$height
[1] 150 150 142 154 168 153 151 153 142 151
> |
```

**Note:** You can find out what type of data is stored within a variable or a column in a dataframe using the `class()` function.



```
Console Terminal
C:/Users/jc161744/Desktop/R Tutorial/
> class(table)
[1] "data.frame"
> class(table$colour)
[1] "factor"
> class(table$height)
[1] "numeric"
> |
```

## **R Commands**

<code>help()</code>	Obtain documentation for a given R command
<code>example()</code>	View some examples on the use of a command
<code>c()</code> , <code>scan()</code>	Enter data manually to a vector in R
<code>seq()</code>	Make arithmetic progression vector
<code>rep()</code>	Make vector of repeated values
<code>data()</code>	Load (often into a <code>data.frame</code> ) built-in dataset
<code>view()</code>	View dataset in a spreadsheet-type format
<code>str()</code>	Display internal structure of an R object
<code>read.csv()</code> , <code>read.table()</code>	Load into a <code>data.frame</code> an existing data file
<code>library()</code> , <code>require()</code>	Make available an R add-on package
<code>dim()</code>	See dimensions (# of rows/cols) of <code>data.frame</code>
<code>length()</code>	Give length of a vector
<code>ls()</code>	Lists memory contents
<code>rm()</code>	Removes an item from memory

## **Examples:**

```
> x = c(8, 6, 7, 5, 3, 0, 9)
> x
```

```
[1] 8 6 7 5 3 0 9
```

```
> names = c("Owen", "Luke", "Anakin", "Leia", "Jacen", "Jaina")
> names
```

```
> [1] "Owen" "Luke" "Anakin" "Leia" "Jacen" "Jaina"
```

```

> heartDeck = c(rep(1, 13), rep(0, 39))
> heartDeck
>
> [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0
> [49] 0 0 0 0
>
> y = seq(7, 41, 1.5)
> y
>
> [1] 7.0  8.5 10.0 11.5 13.0 14.5 16.0 17.5 19.0 20.5 22.0 23.5 25.0 26.5 28.0 29.5
    31.0 32.5 34.0
> [20] 35.5 37.0 38.5 40.0
>
> data(faithful)
> hist (faithful$eruptions)
> hist (faithful$eruptions, n=15)
> hist (faithful$eruptions, breaks=seq(1.5,5.25,.25), col= "red")
> hist (faithful$eruptions, freq=F, n=15, main="Histogram of Old Faithful Eruption Times",
    xlab= "Duration (mins)")

```

## Experiment 2

Interact data through .csv files (Import from and export to .csv files).

### Solution:

A CSV file (Comma Separated Values file) is a type of plain text file that uses specific structuring to arrange tabular data. Because it's a plain text file, it can contain only actual text data—in other words, printable ASCII or Unicode characters. Normally, CSV files use a comma to separate each specific data value. Here's what that structure looks like:

```
column 1 name, column 2 name, column 3 name  
first row data 1, first row data 2, first row data 3  
second row data 1, second row data 2, second row data 3
```

Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and is limited only by file size constraints.

### *Create a CSV file*

Create a CSV file with the following data

```
id,name,salary,start_date,dept  
1,Rick,623.3,2012-01-01,IT  
2,Dan,515.2,2013-09-23,Operations  
3,Michelle,611,2014-11-15,IT  
4,Ryan,729,2014-05-11,HR  
5,Gary,843.25,2015-03-27,Finance  
6,Nina,578,2013-05-21,IT  
7,Simon,632.8,2013-07-30,Operations  
8,Guru,722.5,2014-06-17,Finance
```

### *Read a CSV file*

```
read.csv ()    function to read a CSV file available in your current working directory.  
               > data<-read.csv("input.csv")  
               > print(data)
```

### *Analyzing a CSV file*

```
dim()          See dimensions (# of rows/cols) of dataframe  
ncol()         Returns number of columns in the dataframe  
nrow()         Returns number of rows in the dataframe
```



max() Finds the maximum value in the given numeric field

min() Finds the minimum value in the given numeric field

```
> print (is.data.frame(data))
```

```
> print (dim(data))
```

```
> print (ncol(data))
```

```
> print (nrow(data))
```

```
> print (max(data$salary))
```

```
> print (min(data$salary))
```

### ***Querying the dataframe***

subset() Gets the details of the person satisfying the given details

Get the details of the person who got minimum salary

```
> pers <- subset(data, salary == min(salary))
```

```
> print(pers)
```

Get the details of the persons who are working in “Operations” department

```
> opers <- subset(data, dept== “Operations”)
```

```
> print(opers)
```

Get the persons in Finance department whose salary is greater than 800

```
> fins <- subset(data, dept== “Finance” & salary>800)
```

```
> print(fins)
```

read.csv(), read.table() Load into a data.frame an existing data file

```
> data(iris)
```

```
> data(faithful)
```

```
> data(Puromycin)
```

```
> data(LakeHuron)
```

```
> ls()
```

```
[1] “faithful” “heartDeck” “iris” “LakeHuron” “names” “Puromycin” “x” “y”
```

```
> newVector = 1:12
```

```
> ls()
```

```
[1] "faithful" "heartDeck" "iris" "LakeHuron" "names" "newVector" "Puromycin" "x"
[9] "y"
> rm(faithful)
> ls()
[1] "heartDeck" "iris" "LakeHuron" "names" "newVector" "Puromycin" "x" "y"
```

## **Export to CSV file**

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.

## ***Setting the working directory***

You can check which directory the R workspace is pointing to using the `getwd()` function. You can also set a new working directory using `setwd()` function.

```
# Get and print current working directory.
print (getwd())

# Set current working directory.
setwd ("/MyPersonal/RLab")

# Get and print current working directory.
print (getwd())
```

## ***Create a dataframe***

```
#different vectors
Name<-c("Nahida", "Boston", "Hubey", "Justin", "Peter", "Rossow", "Duminy", "Roosevelt")
Gender<-c("Female", "Male", "Female", "Male", "Male", "Female", "male", "Female")
Score<-c(89,76,56,76,78,83,95,61)
Country<-c("Argentina", "USA", "Britain", "India", "Japan", "South Korea", "Israel", "UAE")

#creates a data frame
df<-data.frame(Name, Gender, Score, Country)

#prints the data frame
df
```

	Name	Gender	Score	Country
1	Nahida	Female	89	Argentina
2	Boston	Male	76	USA
3	Hubey	Female	56	Britain
4	Justin	Male	76	India
5	Peter	Male	78	Japan
6	Rossow	Female	83	South Korea
7	Duminy	male	95	Israel
8	Roosevelt	Female	61	UAE

### ***Export the data frame to a CSV file***

Export this data frame to a CSV file using ***write.csv*** function in R. The syntax used to export a DataFrame to CSV in R:

```
write.csv (DataFrame, Filename, Sep= “ “, na= “NA”, row.names = FALSE)
```

DataFrame = input data frame.

Filename = The output file name

Sep = The row values will be separated by this symbol.

na = Identifies the missing values in the data frame.

row.names = Export with / without row names. By default, it will be TRUE.

#writes the data frame to a CSV file with the output filename

```
write.csv (df, “mycsvdata.csv”)
```

### Experiment 3

Get and Clean data using swirl exercises. (Use 'swirl' package, library and install that topic from swirl).

#### Solution:

#### SWIRL

This is an R package for teaching and learning statistics and R simultaneously and interactively. Swirl is great for those who:

- are trying to learn R just for fun
- are just figuring out what R is about
- have no interest in networking on MOOCs
- like interactive learning without an instructor
- don't like watching online videos
- are Google search experts and want to overcome technical hurdles

#### 1. Install SWIRL package

```
> install.packages("swirl")
```

#### 2. Install the lessons

Lessons for FES720 are on my GitHub page

```
> library(swirl)
> install_course_github("saq", "fes720_Basic")
```

This command downloads the first set of lessons to your computer.

#### 3. Start swirl

You will need to do this every time you start R or want to continue an old lesson or start a new lesson.

```
> library(swirl)
> swirl()
```

#### 4. Choose a name

```
> library(swirl)
What shall I call you?
Enter your name.
```

I will need to be able to identify you, so please use:  
firstname lastname, or Yale ID.

This name will also allow you to continue lessons if you stop them in the middle.

#### 5. Choose a course

Please choose a course, or type 0 to exit swirl.

1: fes720 Basic

2: Take me to the swirl course repository!

Selection:

We will be working through the lessons in the 'fes720 Basic' course.

Type: '1'

#### 6. Choose a lesson

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks

2. ...

...

Choose the first lesson: Basic Building Blocks

Type: '1'

#### 7. Do the lesson!

| Attempting to load lesson dependencies...

| Package 'base64enc' loaded correctly!

| In this lesson, we will explore some basic building blocks of the R programming language.

...

Hit 'Enter' to advance when presented with '...'

The screen also shows you how far through the lesson you are (0%).

#### 8. Completing the lesson

**You will need to be connected to the internet to submit your lesson**

When you are done, the last question will ask if you want to submit your answers to me to verify that you completed the lesson.

You should enter the number of your response (usually '1').

This will bring up a new web page, a Google form.

Scroll down, and click 'submit'.

This will send an encrypted response to the Google form so that I can verify you completed the lesson.

## Experiment 4

Visualize all Statistical measures (Mean, Mode, Median, Range, Inter Quartile Range etc., using Histograms, Boxplots and Scatter Plots).

### Solution:

R Programming language provides a variety of simple summary statistics functions that can be applied to a vector of numbers. Two kinds of summary commands used are:

- ***Commands for Single Value Results*** – Produce single value as a result.
- ***Commands for Multiple Value Result*** – Produce multiple results as an output.

#### ***Commands for Single Value Results***

- **`max(x, na.rm = FALSE)`** – It shows the maximum value. By default, NA values are not removed. NA is considered the largest unless *na.rm=true* is used.
- **`min(x, na.rm = FALSE)`** – Shows minimum value in a vector. If there are na values, NA is returned unless *na.rm=true* is used.
- **`length(x)`** – Gives length of the vector and includes na values. *Na.rm=instruction* does not work with this command.
- **`sum(x, na.rm = FALSE)`** – Shows the sum of the vector elements.
- **`mean(x, na.rm = FALSE)`** – We obtain an arithmetic mean with this.
- **`median(x, na.rm = FALSE)`** – Shows the median value of the vector.
- **`sd(x, na.rm = FALSE)`** – Shows the standard deviation.
- **`var(x, na.rm = FALSE)`** – Shows the variance.
- **`mad(x, na.rm = FALSE)`** – Shows the median absolute deviation.

#### ***Commands for Multiple Value Result***

**`log(dataset)`** – Shows log value for each element.

**`summary(dataset)`** – We have seen how it shows a summary of dataset like maximum value, minimum value, mean, etc.

**`quantile()`** – Shows the quantiles by default—the 0%, 25%, 50%, 75%, and 100% quantiles. You can select other quantiles also. The *quantile()* command produces multiple results by default. One can alter the default result to produce quantiles for a single probability or several (in any order).

The names of the quantiles selected are displayed as percentage labels. You can suppress this by using `name = FALSE` instruction. If the data contains NA items, you must remove them using the `na.rm = TRUE` instruction, otherwise, you get an error message.

The command allows other instructions as follows:

```
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE)
```

## Data Visualization

A data visualization is a method of representing data in a graphical format, useful both in communicating results of analyses and in exploring datasets to determine what analyses might be appropriate.

### Histograms

A histogram contains a rectangular area to display the statistical information which is proportional to the frequency of a variable and its width in successive numerical intervals. A graphical representation that manages a group of data points into different specified ranges. It has a special feature which shows no gaps between the bars and is similar to a vertical bar graph.

**Syntax:** `hist(v, main, xlab, xlim, ylim, breaks, col, border)`

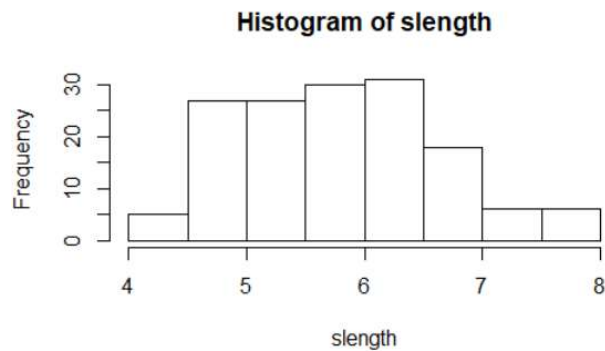
#### Parameters:

- **v:** This parameter contains numerical values used in histogram.
- **main:** This parameter main is the title of the chart.
- **col:** This parameter is used to set color of the bars.
- **xlab:** This parameter is the label for horizontal axis.
- **border:** This parameter is used to set border color of each bar.
- **xlim:** This parameter is used for plotting values of x-axis.
- **ylim:** This parameter is used for plotting values of y-axis.
- **breaks:** This parameter is used as width of each bar.

#### Examples:

##### Simple histogram

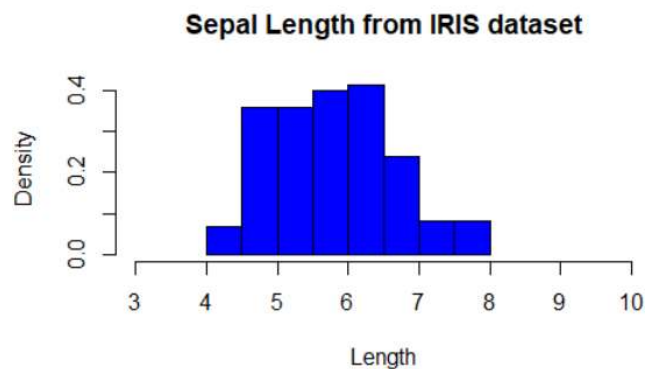
```
> iris  
> view(iris)  
> slength<-iris$Sepal.Length  
> hist (slength)
```



We can see above that there are 8 cells with equally spaced breaks. In this case, the height of a cell is equal to the number of observation falling in that cell.

### *Histogram with parameters*

```
> hist (slength, main="Sepal Length from IRIS dataset", xlab="Length",
        xlim=c(3,10), col="blue",freq=FALSE)
```



### *Return value from hist()*

```
> hreturn <- hist (slength)
```

```
> hreturn
```

```
$breaks
```

```
[1] 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
```

```
$counts
```

```
[1] 5 27 27 30 31 18 6 6
```

```
$density
```

```
[1] 0.06666667 0.36000000 0.36000000 0.40000000 0.41333333
```

```
[6] 0.24000000 0.08000000 0.08000000
```

```
$mids
```

```
[1] 4.25 4.75 5.25 5.75 6.25 6.75 7.25 7.75
```



```
$xname  
[1] "slength"  
$equidist  
[1] TRUE  
attr(,"class")  
[1] "histogram"
```

## 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.

**Syntax:** `boxplot(x, data, notch, varwidth, names, main)`

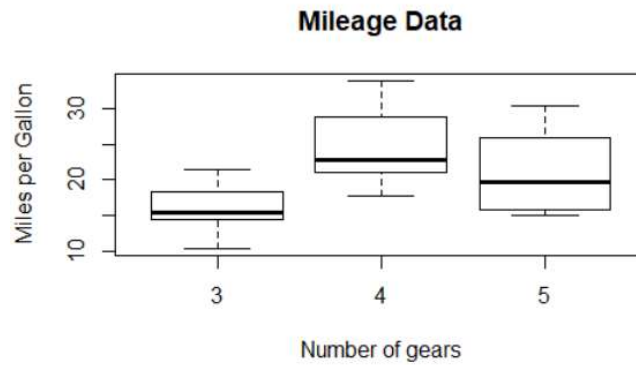
### **Parameters:**

- **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.

### **Examples:**

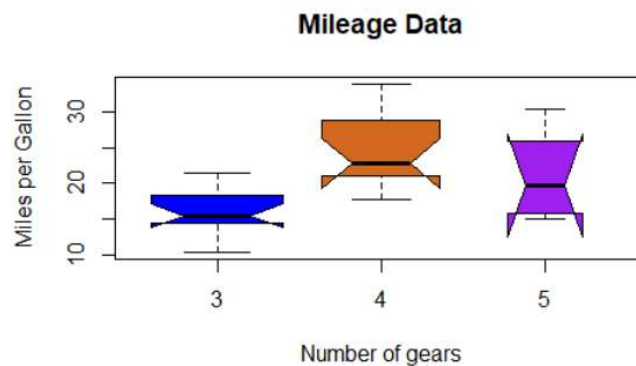
#### **Simple Boxplot**

```
> data <- mtcars  
> view (data)  
> df <- data [mtcars, c("mpg", "gear")]  
> df  
> boxplot (mpg ~ gear, data=df, xlab="Number of gears", ylab="Miles per Gallon",  
main="Mileage Data")
```



### ***Boxplot with notch***

```
> boxplot (mpg ~ gear, data=df, xlab="Number of gears", ylab="Miles per Gallon",
main="Mileage Data", notch=TRUE, varwidth=TRUE, col=c("blue","chocolate","purple"))
```



### **Scatter plots**

Scatterplots show many points plotted in the Cartesian plane. A scatter plot is a set of dotted points to represent individual pieces of data in the horizontal and vertical axis. A graph in which the values of two variables are plotted along X-axis and Y-axis, the pattern of the resulting points reveals a correlation between them.

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

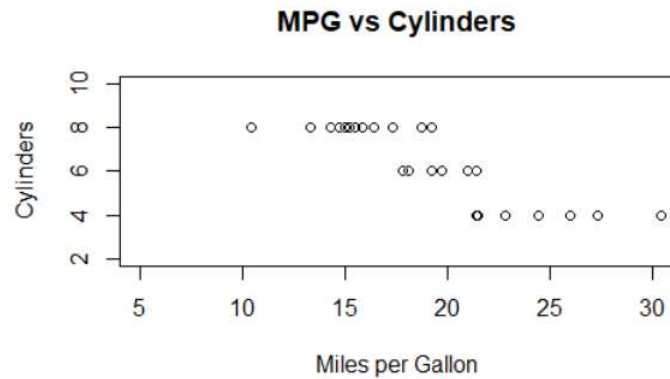
#### ***Parameters:***

- **x:** This parameter sets the horizontal coordinates.
- **y:** This parameter sets the vertical coordinates.
- **xlab:** This parameter is the label for horizontal axis.
- **ylab:** This parameter is the label for vertical axis.
- **main:** This parameter main is the title of the chart.
- **xlim:** This parameter is used for plotting values of x.

- **ylim:** This parameter is used for plotting values of y.
- **axes:** This parameter indicates whether both axes should be drawn on the plot.

**Example:**

```
> plot (x=df$mpg, y=df$cyl, xlab="Miles per Gallon",ylab="Cylinders", xlim = c(5,30), ylim =  
c(2,10), main="MPG vs Cylinders")
```



## Experiment 5

Create a data frame with the following structure.

EMP ID	EMP NAME	SALARY	START DATE
1	Satish	5000	01-11-2013
2	Vani	7500	05-06-2011
3	Ramesh	10000	21-09-1999
4	Praveen	9500	13-09-2005
5	Pallavi	4500	23-10-2000

- a) Extract two columns using column name.
- b) Extract the first two rows and then all columns.
- c) Extract 3rd and 5th row with 2nd and 4th column.

### Solution:

#### *Creating the data frame:*

```
> df <- data.frame (EmpId=c(1,2,3,4,5),  
EmpName=c("Satish","Vani","Ramesh","Praveen","Pallavi"),  
Salary=c(5000,7500,10000,9500,4500),  
StartDate=c("01-11-2013","05-06-2011","21-09-1999","13-09-2005","23-10-2000"))  
> df
```

```
EmpId EmpName Salary StartDate  
1    1  Satish  5000 01-11-2013  
2    2   Vani  7500 05-06-2011  
3    3 Ramesh 10000 21-09-1999  
4    4 Praveen  9500 13-09-2005  
5    5 Pallavi  4500 23-10-2000
```

- a) Extract two columns using column name.

```
> df1 <- df[,c("EmpName", "Salary")]  
> df1
```

```
EmpName Salary  
1 Satish  5000  
2  Vani  7500  
3 Ramesh 10000  
4 Praveen  9500  
5 Pallavi  4500
```

*Extracting columns using column position*

```
> df2<-df[,c(1,4)]
```

```
> df2
```

	EmpId	StartDate
1	1	01-11-2013
2	2	05-06-2011
3	3	21-09-1999
4	4	13-09-2005
5	5	23-10-2000

b) Extract the first two rows and then all columns.

```
> df3<- df[1:2,]
```

```
> df3
```

	EmpId	EmpName	Salary	StartDate
1	1	Satish	5000	01-11-2013
2	2	Vani	7500	05-06-2011

c) Extract 3rd and 5th row with 2nd and 4th column.

```
> df4<-df[c(3,5),c(2,4)]
```

```
> df4
```

	EmpName	StartDate
3	Ramesh	21-09-1999
5	Pallavi	23-10-2000

## Experiment 6

Write R Program using 'apply' group of functions to create and apply normalization function on each of the numeric variables/columns of iris dataset to transform them into

- i) 0 to 1 range with min-max normalization.
- ii) a value around 0 with z-score normalization.

### Solution:

#### Data Normalization:

Data Normalization is a data preprocessing step where we adjust the scales of the features to have a standard scale of measure. This is also known as *Feature Scaling*. This is one of the preliminary steps in data preprocessing. If this is neglected or bypassed, it leads to bias and in turn influences the prediction accuracy.

Distance-Based algorithms, such as SVM, K-Means, and KNN classifies objects by finding similarity between them using distance functions. These algorithms are receptive to the size of the variables. If we don't scale, the feature with a higher magnitude (Salary), will have more influence than the feature with a lower magnitude (years of experience), and this leads to bias in data, and it also affects the accuracy.

#### Min-Max Normalization

Min-Max Normalization transforms  $x$  to  $x'$  by converting each value of features to a range between 0 and 1, and this is also known as (0–1) Normalization. If the data has negative values the range would have been between -1 and 1.

The formula for Min-Max Normalization is:

$$\text{Normalized Value } x' = \frac{\text{Original Value } x - \text{Minimum Value of } x}{\text{Maximum Value of } x - \text{Minimum Value of } x}$$

#### Example:

The above formula is used to create a custom user-defined function, minmax which takes in

one value at a time and computes the scaled value such that it lies between 0 and 1.

```
> minmax <- function (x) { (x - min(x)) / (max(x) - min(x))  
> minmax (df$Salary)  
[1] 0.09090909 0.54545455 1.00000000 0.90909091 0.00000000
```

### ***Z-Score Normalization***

Z-score Normalization transforms  $x$  to  $x'$  by subtracting each value of features by the sample mean and then dividing by the sample standard deviation. The resulting mean and standard deviation of the standardized values are 0 and 1, respectively. The formula for standardization is

$$x' = \frac{x - \mu}{\sigma}$$

### ***Procedure:***

- Declare a vector
- Calculate its mean and standard deviation by the functions **mean()** and **sd()**.
- To create a standardized vector:
  - Subtract mean from the vector
  - Now divide the above result with standard deviation

### ***Example:***

```
> df <- data.frame (EmpId=c(1,2,3,4,5),  
  EmpName=c("Satish","Vani","Ramesh","Praveen","Pallavi"),  
  Salary=c(5000,7500,10000,9500,4500),  
  StartDate = c("01-11-2013", "05-06-2011", "21-09-1999", "13-09-2005", "23-10-2000"))  
> df  
  EmpId EmpName Salary StartDate  
1     1  Satish  5000 01-11-2013
```

```
2 2 Vani 7500 05-06-2011
3 3 Ramesh 10000 21-09-1999
4 4 Praveen 9500 13-09-2005
5 5 Pallavi 4500 23-10-2000
```

```
> m <- mean(df$Salary)
```

```
> s <- sd(df$Salary)
```

```
> zscores <- (df$Salary - m) / s
```

```
> zscores
```

```
[1] -0.91452919 0.07952428 1.07357774 0.87476705 -1.11333988
```



## Experiment 7

Create a data frame with 10 observations and 3 variables and add new rows and columns to it using 'rbind' and 'cbind' function.

### Explanation:

#### Matrices and Dataframes

There are a number of ways to create a matrix and dataframe objects in R. The most common functions are described in the following table. Because matrices and dataframes are just combinations of vectors, each function takes one or more vectors as inputs, and returns a matrix or a dataframe.

<b>cbind(a, b, c)</b>	Combine vectors as columns in a matrix
<b>rbind(a, b, c)</b>	Combine vectors as rows in a matrix
<b>matrix(x, nrow, ncol, byrow)</b>	Create a matrix from a vector x
<b>data.frame()</b>	Create a dataframe from named column

#### cbind and rbind functions:

The name of the *cbind* function in R stands for column-bind and *rbind* stands for row-bind. These functions are used to combine vectors, matrices and/or data frames by columns / rows. The syntax of these functions is

```
cbind(my_data, new_column)
rbind (my_data, new_row)
```

#### Example:

Let us create three vectors of length 4, then we'll combine them into one matrix. The cbind() function will combine the vectors as columns in the final matrix, while the rbind() function will combine them as rows.

```
> cmatrix <- cbind (a,b,c)
> cmatrix
  a b c
[1,] 1 5 9
[2,] 2 6 10
[3,] 3 7 11
```

```

[4,] 4 8 12
> rmatrix <- rbind (a,b,c)
> rmatrix
  [,1] [,2] [,3] [,4]
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
> cmatrix <- cbind (a,b,c,deparse.level=0)
> cmatrix
  [,1] [,2] [,3]
[1,]  1   5   9
[2,]  2   6  10
[3,]  3   7  11
[4,]  4   8  12
> rmatrix <- rbind (a,b,c,deparse.level=0)
> rmatrix
  [,1] [,2] [,3] [,4]
[1,]  1   2   3   4
[2,]  5   6   7   8
[3,]  9  10  11  12

```

### **Solution:**

1. Create a new data frame with 5 rows with emp\_id, emp\_name, salary and start\_date as columns

```

> emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)

```

```
> emp.data
  emp_id emp_name salary start_date
1     1    Rick 623.30 2012-01-01
2     2     Dan 515.20 2013-09-23
3     3 Michelle 611.00 2014-11-15
4     4    Ryan 729.00 2014-05-11
5     5     Gary 843.25 2015-03-27
```

2. Create a new vector as dept with department names of the employees

```
> dept <- c ("IT","Operations","IT","HR","Finance")
```

3. Add this dept vector as a new column to data frame

```
> emp.data <- cbind (emp.data, dept)
> emp.data
  emp_id emp_name salary start_date   dept
1     1    Rick 623.30 2012-01-01    IT
2     2     Dan 515.20 2013-09-23 Operations
3     3 Michelle 611.00 2014-11-15    IT
4     4    Ryan 729.00 2014-05-11    HR
5     5     Gary 843.25 2015-03-27  Finance
```

4. Create a new data frame with 3 more records

```
> emp.newdata <- data.frame(
  emp_id = c (6:8),
  emp_name = c("Rasmi","Pranab","Tusar"),
  salary = c(578.0,722.5,632.8),
  start_date = as.Date (c ("2013-05-21","2013-07-30","2014-06-17")),
  dept = c("IT", "Operations", "Finance"),
  stringsAsFactors = FALSE
)
```

5. Add these new records to the existing data frame emp.data

```
> emp.finaldata <- rbind (emp.data, emp.newdata)
```

```
> print (emp.finaldata)
```

	emp_id	emp_name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	5	Gary	843.25	2015-03-27	Finance
6	6	Rasmi	578.00	2013-05-21	IT
7	7	Pranab	722.50	2013-07-30	Operations
8	8	Tusar	632.80	2014-06-17	Finance

## Experiment 8:

Write R program to implement linear and multiple regression on 'mtcars' dataset to estimate the value of 'mpg' variable, with best  $R^2$  and plot the original values in 'green' and predicted values in 'red'.

### Solution:

The built-in **mtcars** data frame contains information including their weight, fuel efficiency (in miles-per-gallon), speed, etc. of 32 models of automobile from 1973-74 as reported in Motor Trend Magazine. In analyzing the dataset of different collection of cars, we will explore the relationship between a set of eleven variables, and miles per gallon (MPG). Dataset Motor Trend has been used to find out that,

- Is an automatic or manual transmission better for miles per gallon?
- How different is the MPG between automatic and manual transmission?

### Read MTCARS dataframe

```
> data ("mtcars")
```

```
> 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

We have to explore the relationship between a set of variables and miles per gallon (mpg), so mpg is our **dependent variable**. Plot dependent variable to check its distribution.

```
x <- mtcars$mpg
```

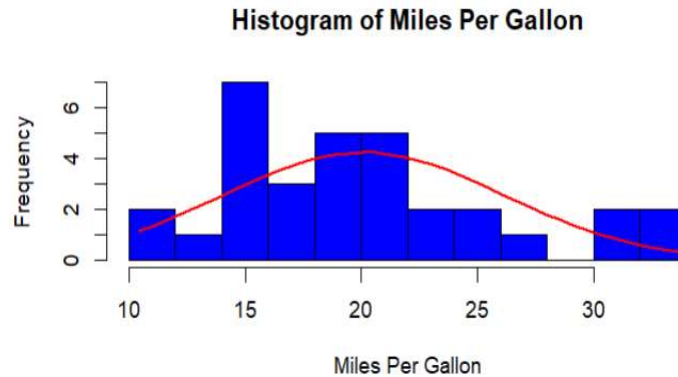
```
h<- hist (x, breaks=10, col="blue", xlab="Miles Per Gallon", main="Histogram of Miles Per Gallon")
```

```
xfit <- seq (min(x),max(x),length=40)
```

```
yfit <- dnorm (xfit, mean=mean(x), sd=sd(x))
```

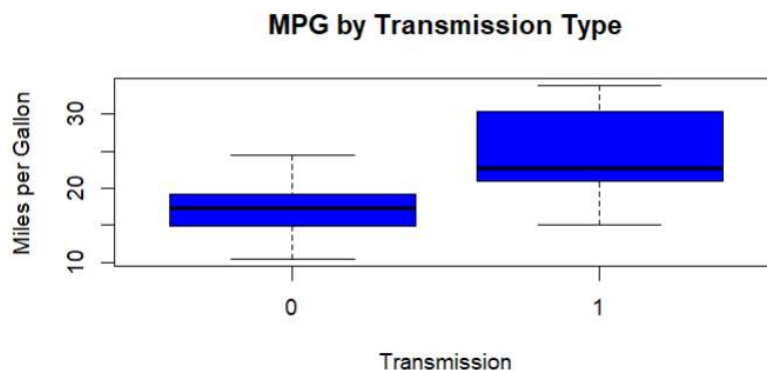
```
yfit <- yfit*diff(h$mids[1:2])*length(x)
```

```
lines (xfit, yfit, type="l", col="red", lwd=2)
```



The distribution of mpg, showing in the graphs is approximately normal and does not contain any outliers. Now we check how mpg can be changed by automatic or manual transmission, by plotting a box plot. From this boxplot, it seems that automatic cars have a lower miles per gallon, and so a lower fuel potency, than manual cars do.

```
> boxplot(mpg~am, data = mtcars, col = c("blue", "blue"), xlab = "Transmission",
  ylab = "Miles per Gallon", main = "MPG by Transmission Type")
```



## Correlation analysis

A correlation test is performed to determine the relationship between the variables, and to find out which variables should be included in our model to answer the questions. The correlation matrix is

```
> cor (mtcars, use="complete.obs", method="pearson")
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
mpg	1.0000000	-0.8521620	-0.8475514	-0.7761684	0.6811719	-0.8676594	0.4186840	0.6640389	0.5998324	0.4802848	-0.5509250
cyl	-0.8521620	1.0000000	0.9020329	0.8324475	-0.6999381	0.7824958	-0.5912420	-0.8108118	-0.5226070	-0.4926866	0.5269882
disp	-0.8475514	0.9020329	1.0000000	0.7909486	-0.7102139	0.8879799	-0.4336978	-0.7104159	-0.5912270	-0.5555692	0.3949768
hp	-0.7761684	0.8324475	0.7909486	1.0000000	-0.4487591	0.6587479	-0.7082233	-0.7230967	-0.2432042	-0.1257043	0.7498124
drat	0.6811719	-0.6999381	-0.7102139	-0.4487591	1.0000000	-0.7124406	0.0912047	0.4402785	0.7127113	0.6996101	-0.0907898
wt	-0.8676594	0.7824958	0.8879799	0.6587479	-0.7124406	1.0000000	-0.1747158	-0.5549157	-0.6924952	-0.5832870	0.4276059
qsec	0.4186840	-0.5912421	-0.4336979	-0.7082234	0.0912047	-0.1747159	1.0000000	0.7445354	-0.2298608	-0.2126822	-0.6562492
vs	0.6640389	-0.8108118	-0.7104159	-0.7230967	0.4402784	-0.5549157	0.7445354	1.0000000	0.1683451	0.2060233	-0.5696071
am	0.5998324	-0.5226070	-0.5912270	-0.2432043	0.7127113	-0.6924953	-0.2298608	0.1683451	1.0000000	0.7940587	0.0575343
gear	0.4802848	-0.4926866	-0.5555692	-0.1257043	0.6996101	-0.5832870	-0.2126822	0.2060233	0.7940587	1.0000000	0.2740728
carb	-0.5509251	0.5269883	0.3949769	0.7498125	-0.0907898	0.4276059	-0.6562492	-0.5696071	0.0575343	0.2740728	1.0000000

The values in correlation matrix shows that variables such as *wt*, *cyl*, *disp*, and *hp* are highly correlated with the dependent variable *mpg*. Hence, they should be included in the regression model. From the correlation matrix, it can be also be observed that *cyl* and *disp* are highly correlated with each other. In order to avoid the problem of collinearity only one variable from these two will be included in the model.

### ***Simple Regression model***

```
> fit <- lm(mpg~am, data=mtcars)
> summary(fit)

Call:
lm(formula = mpg ~ am, data = mtcars)

Residuals:
    Min       1Q   Median       3Q      Max
-9.3923 -3.0923 -0.2974  3.2439  9.5077

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  17.147      1.125   15.247 1.13e-15 ***
am           7.245      1.764    4.106 0.000285 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.902 on 30 degrees of freedom
Multiple R-squared:  0.3598,    Adjusted R-squared:  0.3385
F-statistic: 16.86 on 1 and 30 DF,  p-value: 0.000285
```

From the above summary, there exists a linear relation between the predictor variable MPG and AM. Intercepts and Coefficient can be explained that, on average automatic transmission cars has 17.147 MPG and manual transmission cars has 24.39(17.147 + 7.24).The value of  $R^2$  is 0.3385, which means this model only explain 33.85% of the variance.

### ***Multiple Regression model***

In the correlation analysis, it is observed that variables such as *wt*, *cyl*, *am*, and *hp* are highly correlated with the dependent variable *mpg*. So, we apply a multi variant regression for *mpg* on *am*, *wt*, *cyl*, and *hp*.

```
> mfit <- lm (mpg ~ am + cyl + wt + hp, data = mtcars)
> anova (fit, mfit)
```

#### Analysis of Variance Table

Model 1: mpg ~ am

Model 2: mpg ~ am + cyl + wt + hp

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	30	720.9				
2	27	170.0	3	550.9	29.166	1.274e-08 ***

---

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

The p-value of 1.274e-08, it is clear that the multivariate model of regression is different from that of above simple model.

```
> summary(mfit)
```

Call:

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

Residuals:

	Min	1Q	Median	3Q	Max
	-3.4765	-1.8471	-0.5544	1.2758	5.6608

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	36.14654	3.10478	11.642	4.94e-12 ***
am	1.47805	1.44115	1.026	0.3142
cyl	-0.74516	0.58279	-1.279	0.2119
wt	-2.60648	0.91984	-2.834	0.0086 **
hp	-0.02495	0.01365	-1.828	0.0786 .

---

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

Residual standard error: 2.509 on 27 degrees of freedom

Multiple R-squared: 0.849, Adjusted R-squared: 0.8267

F-statistic: 37.96 on 4 and 27 DF, p-value: 1.025e-10

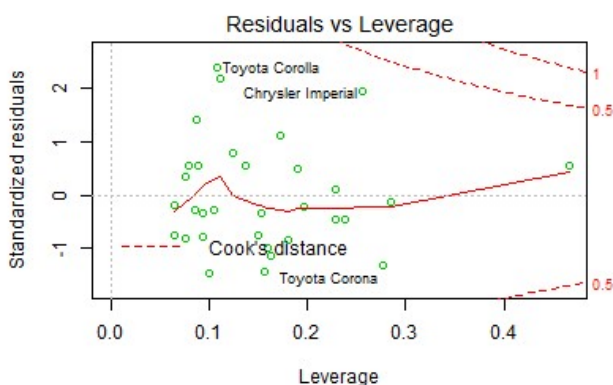
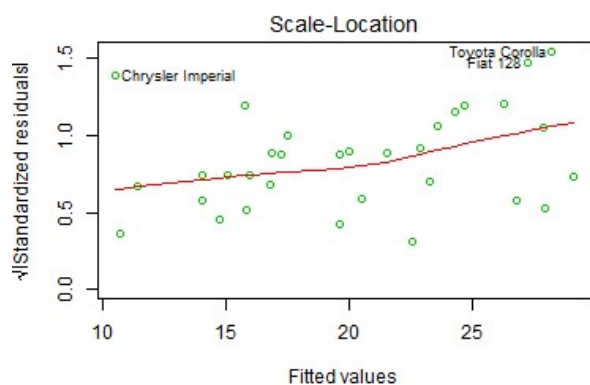
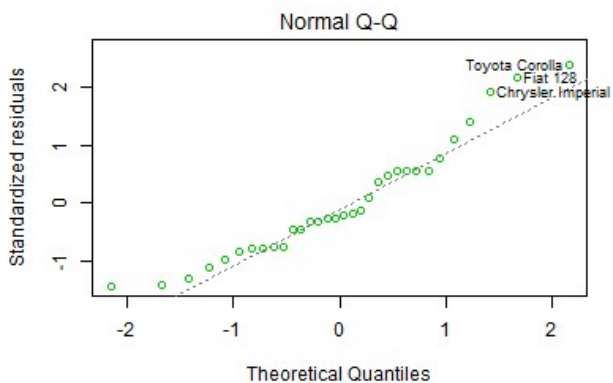
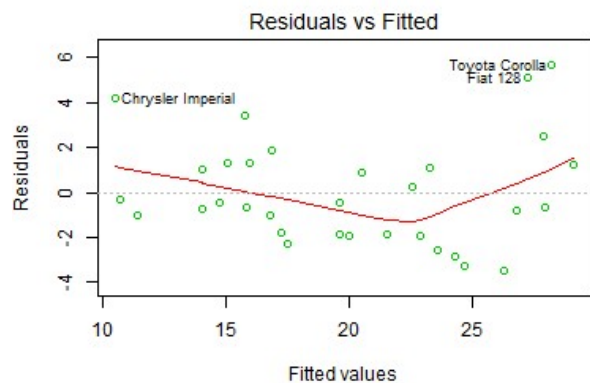
Multivariate regression model explain 84.9% variance. It can be seen that *wt* and up to some extent *hp* confound the relationship between *am* and *mpg*.

#### Result plots

```
> par(mfrow = c(2,2))
```

```
> plot(mfit, col=3)
```





## **Experiment 9:**

Implement k-means clustering using R.

### **Solution:**

#### **K-Means algorithm**

K-means clustering is one of the most commonly used unsupervised machine learning algorithm for partitioning a given data set into a set of  $k$  groups (i.e.  $k$  clusters), where  $k$  represents the number of groups pre-specified by the analyst. It classifies objects in multiple clusters, such that objects within the same cluster are as similar as possible (i.e., high intra-class similarity), whereas objects from different clusters are as dissimilar as possible (i.e., low inter-class similarity). In k-means clustering, each cluster is represented by its center (i.e, centroid) which corresponds to the mean of points assigned to the cluster.

#### ***Algorithm summary***

K-means algorithm can be summarized as follow:

1. Specify the number of clusters ( $K$ ) to be created (by the analyst)
2. Select randomly  $k$  objects from the dataset as the initial cluster centers or means
3. Assigns each observation to their closest centroid, based on the Euclidean distance between the object and the centroid
4. For each of the  $k$  clusters update the cluster centroid by calculating the new mean values of all the data points in the cluster. The centroid of a  $K_{th}$  cluster is a vector of length  $p$  containing the means of all variables for the observations in the  $k_{th}$  cluster;  $p$  is the number of variables.
5. Iteratively minimize the total within sum of square. That is, iterate steps 3 and 4 until the cluster assignments stop changing or the maximum number of iterations is reached. By default, the R software uses 10 as the default value for the maximum number of iterations.

#### ***Algorithm implementation***

1. Install the relevant packages and call their libraries

```
> install.packages("dplyr")  
> install.packages("ggplot2")  
> install.packages("ggfortify")  
> library("ggplot2")  
> library("dplyr")
```

```
> library("ggfortify")
```

## 2. Loading and analyzing the dataset

```
> summary ("iris")
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min.	:4.300	:2.000	:1.000	:0.100	setosa :50
1st Qu.	:5.100	:2.800	:1.600	:0.300	versicolor:50
Median	:5.800	:3.000	:4.350	:1.300	virginica :50
Mean	:5.843	:3.057	:3.758	:1.199	
3rd Qu.	:6.400	:3.300	:5.100	:1.800	
Max.	:7.900	:4.400	:6.900	:2.500	

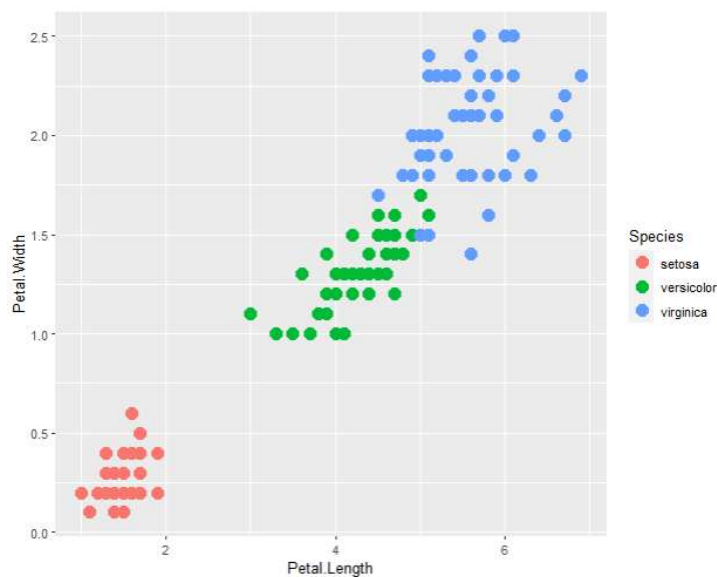
```
> head ("iris")
```

	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
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
> tail ("iris")
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

```
> ggplot(iris)+aes(Petal.Length,Petal.Width)+geom_point(aes(col=Species),size=4)
```



## 3. Eliminating the target variable

```
> data <- select (iris, c(1:4))
```

K-means algorithm requires dataframe and k value and the command is

```
> kmean <- kmeans (data, 2)
```

```
> kmean
```

K-means clustering with 2 clusters of sizes 53, 97

Cluster means:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.005660	3.369811	1.560377	0.290566
2	6.301031	2.886598	4.958763	1.695876

Clustering vector:

[illegible]

Within cluster sum of squares by cluster:

```
[1] 28.55208 123.79588
(between_SS / total_SS = 77.6 %)
```

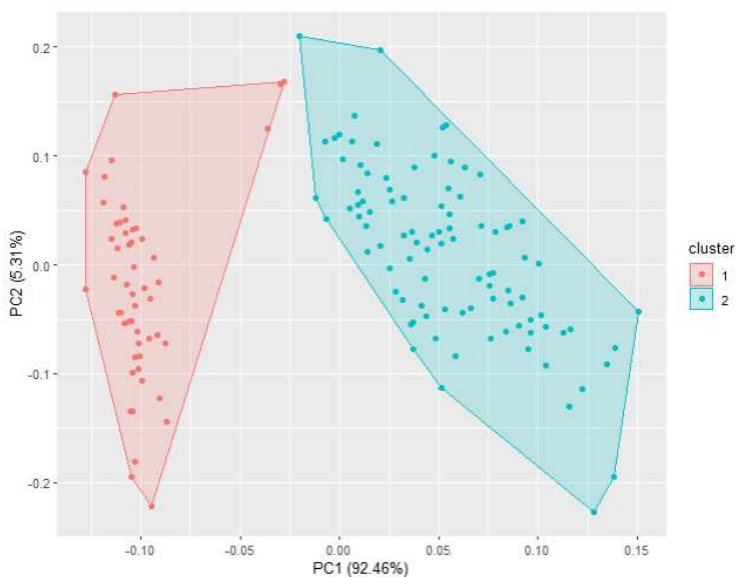
Available components:

```
[1] "cluster"      "centers"      "tottss"       "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

***kmean\$clusters*** return a vector of numbers ranging from 1 to 2, representing which observations belong to cluster 1 and cluster 2. ***kmean\$centers*** return the location of each centroid. For e.g. cluster 1 has a mean value of Sepal.Length = 5.00, Sepal.width = 3.36, Petal.Length = 1.56 and Petal.width = 0.29.

## 5. Plotting our data-points in clusters

```
> autoplot (kmean, data, frame=TRUE)
```



## 6. K-means with $k = 3$

```
> kmean3 <- kmeans (data, 3)
```

```
> kmean3
```

```
K-means clustering with 3 clusters of sizes 33, 21, 96

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1    5.175758    3.624242    1.472727    0.2727273
2    4.738095    2.904762    1.790476    0.3523810
3    6.314583    2.895833    4.973958    1.7031250

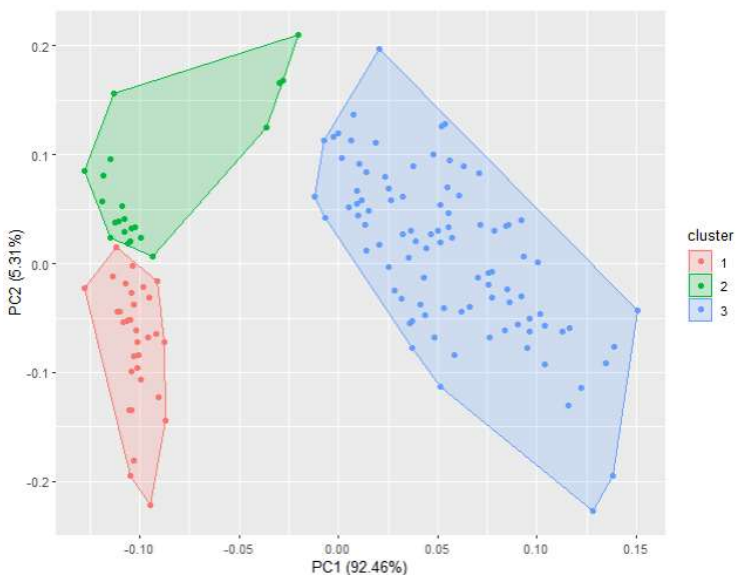
Clustering vector:
 [1] 1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 2 2
[44] 1 1 2 1 2 1 1 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[87] 3 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[130] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:
[1]  6.432121 17.669524 118.651875
(between_SS / total_SS =  79.0 %)

Available components:
[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"       "
```

We see how *kmean\$clusters* has divided the observations into three clusters now and *kmean\$centers* has also updated the centroid values as well. The plot below shows the grouping based on 3 clusters.

```
> autoplot(kmean3, data, frame=TRUE)
```



K-means is an efficient Machine Learning technique that:

- is easy to implement and apply
- has great interpretability
- produces tighter clusters than Hierarchical clustering
- is computationally fast

## Experiment 10:

Implement K-Medoids clustering using R.

### Solution:

#### Partition Around Medoids (PAM)

PAM stands for “Partition Around Medoids.” PAM converts each step of PAM from a deterministic computational to a statistical estimation problem and reduces the complexity of a sample size  $n$  to  $O(n \log n)$ . Medoids are data points chosen as cluster centers. K-Means clustering aims at minimizing the intra-cluster distance (often referred to as the total squared error). In contrast, K-Medoids minimizes dissimilarities between points in a cluster and points considered as centers of that cluster.

#### Algorithm

The fundamental concept of PAM includes:

1. Find a set of  $k$  Medoids ( $k$  refers to the number of clusters, and  $M$  is a collection of medoids) from the data points of size  $n$  ( $n$  being the number of records).
2. Using any distance metric (say  $d(\cdot)$ , could be euclidean, manhattan, etc.), try and locate Medoids that minimize the overall distance of data points to their closest Medoid.
3. Finally, swap Medoid and non-Medoid pairs that reduce the loss function  $L$  among all possible  $k(n-k)$  pairs. The loss function is defined as:

$$L(M) = \sum_{i=1}^n \min_{m \in M} d(m, x_i)$$

*Update centroids:* In the case of K-Means, we were computing the mean of all points present in the cluster. But for the PAM algorithm, the updation of the centroid is different. If there are  $m$ -point in a cluster, swap the previous centroid with all other  $(m-1)$  points and finalize the point as a new centroid with a minimum loss. Minimum loss is computed by the above cost function

#### Algorithm implementation

1. Install the relevant packages and call their libraries

```
> library("ggplot2")
```

```
> library("cluster")
```

2. Loading and analyzing the dataset

```
> summary("iris")
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa :50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor:50
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica :50
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

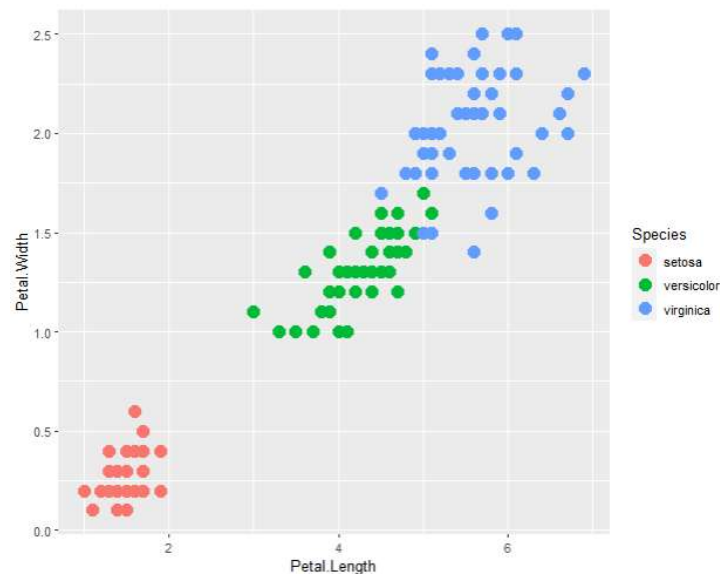
```
> head("iris")
```

	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
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
> tail("iris")
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

```
> ggplot(iris)+aes(Petal.Length,Petal.Width)+geom_point(aes(col=Species),size=4)
```



### 3. Eliminating the target variable

```
> data <- select (iris, c(1:4))
```

### 4. Apply k-medoids algorithm using PAM function

```
> kmediod <- pam(data, k=3, metric="euclidean")
```

```
> kmediod
```





## Experiment 11:

Implement density based clustering on iris dataset.

### Solution:

#### DBSCAN Algorithm

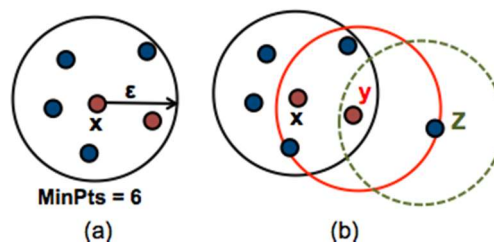
Density Based Spatial Clustering of Applications with Noise (DBSCAN) is the best known density-based clustering algorithm. The central idea behind DBSCAN and its extensions and revisions is the concept that points are assigned to the same cluster if they are density-reachable from each other. Similar to K-Means, DBSCAN is grouping data by their similarities based on distance functions and density. Density here means number of points in region defined by model parameters.

#### Algorithm

Two important parameters are required for DBSCAN: epsilon (“*eps*”) and minimum points (“*MinPts*”). The parameter *eps* defines the radius of neighborhood around a point *x*. It’s called the epsilon-neighborhood of *x*. The parameter *MinPts* is the minimum number of neighbors within “*eps*” radius.

Any point *x* in the dataset, with a neighbor count greater than or equal to *MinPts*, is marked as a **core point**. We say that *x* is **border point**, if the number of its neighbors is less than *MinPts*, but it belongs to the epsilon-neighborhood of some core point *z*. Finally, if a point is neither a core nor a border point, then it is called a **noise point** or an **outlier**.

The figure below shows the different types of points (**core**, **border** and **outlier points**) using *MinPts* = 6. Here *x* is a core point because  $\text{neighbours\_epsilon}(x) = 6$ , *y* is a border point because  $\text{neighbours\_epsilon}(y) < \text{MinPts}$ , but it belongs to the epsilon-neighborhood of the core point *x*. Finally, *z* is a noise point.



The implementation of DBSCAN performs the following steps:

1. Estimate the density around each data point by counting the number of points in a user-specified *eps*-neighborhood and applies a user-specified *minPts* thresholds to identify core,

border and noise points.

2. Core points are joined into a cluster if they are density-reachable (i.e., there is a chain of core points where one falls inside the eps-neighborhood of the next).
3. Border points are assigned to clusters. The algorithm needs parameters *eps* (the radius of the epsilon neighborhood) and *minPts* (the density threshold).

### **Algorithm implementation**

1. Install the relevant packages and call their libraries

```
> library("ggplot2")  
> library("cluster")  
> library("fpc")
```

2. Loading and analyzing the dataset

```
> summary("iris")
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa :50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor:50
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica :50
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

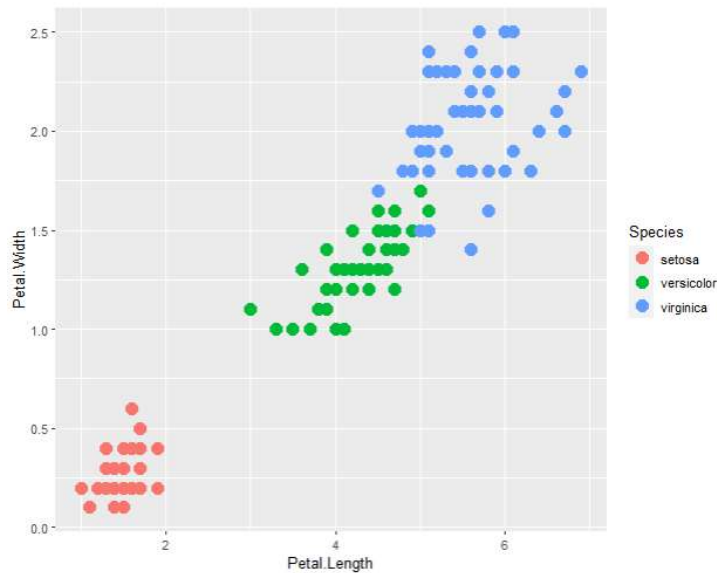
```
> head("iris")
```

	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
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
> tail("iris")
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

```
> ggplot(iris)+aes(Petal.Length,Petal.Width)+geom_point(aes(col=Species),size=4)
```



### 3. Eliminating the target variable

```
> data <- select (iris, c(1:4))
```

```
> head (data)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4

### 4. Apply DBSCAN algorithm

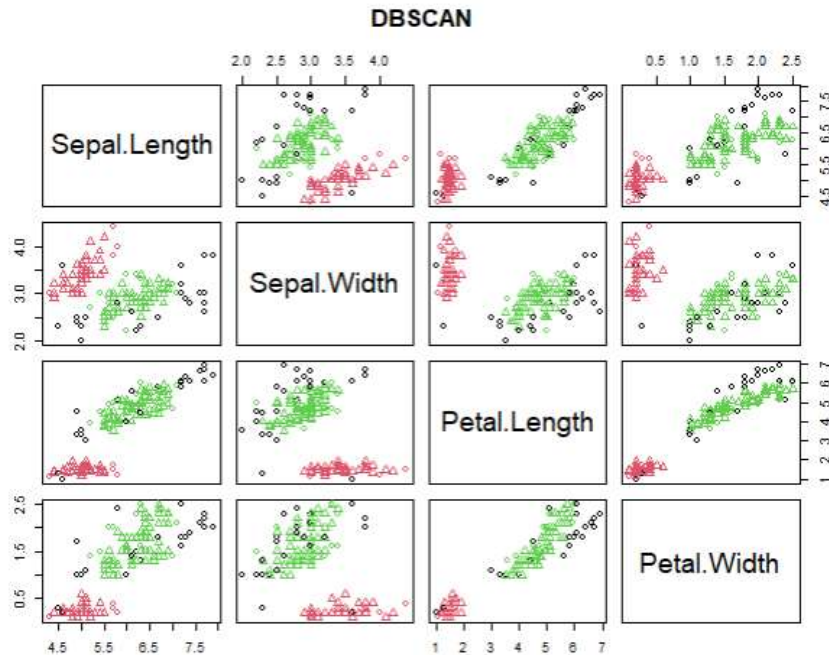
```
> dbiris <- fpc::dbscan (data, eps=0.45, MinPts = 5)
```

```
> print (dbiris)
```

```
dbscan Pts=150 MinPts=5 eps=0.45
      0  1  2
border 24  4 13
seed   0 44 65
total  24 48 78
```

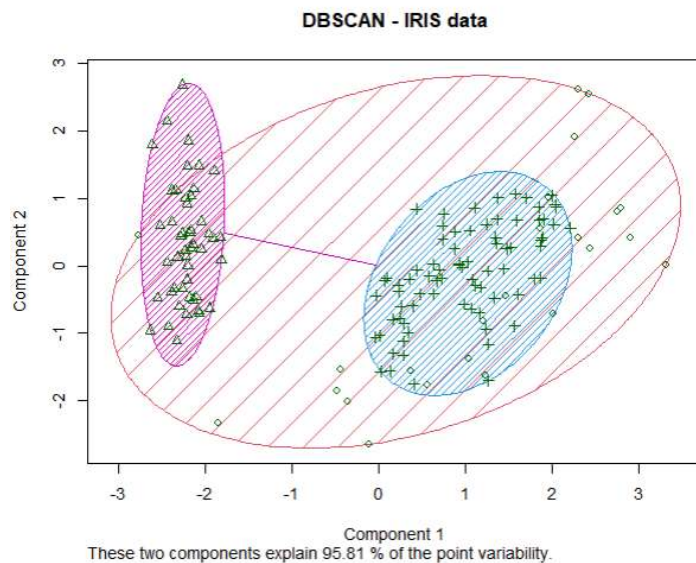
### 5. Plot the results of the algorithm

```
> plot (dbiris, data, main="DBSCAN", frame=FALSE)
```



## 6. Plot clusters

```
> clusplot(data,dbiris$cluster,color=T,shade=T,main="DBSCAN - IRIS data")
```



### *Advantages of DBSCAN algorithm*

1. Unlike to K-means, DBSCAN does not require the user to specify the number of clusters to be generated
2. DBSCAN can find any shape of clusters. The cluster doesn't have to be circular.
3. DBSCAN can identify outliers

## Experiment 12:

Implement decision trees using 'readingSkills' dataset.

### Solution:

#### Decision Tree

Decision Trees are a popular Data Mining technique that makes use of a tree-like structure to deliver consequences based on input decisions. One important property of decision trees is that it is used for both regression and classification. This type of classification method is capable of handling heterogeneous as well as missing data. Decision Trees are further capable of producing understandable rules. Furthermore, classifications can be performed without many computations. Both the classification and regression tasks can be performed with the help of Decision Trees.

#### *Algorithm implementation*

1. Load the relevant packages and call their libraries

```
> library(datasets)
> library(caTools)
> library(party)
> library(dplyr)
> library(magrittr)
```

2. Loading and analyzing the dataset

```
> data ("readingSkills")
> head (readingSkills)
  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

> tail (readingSkills)
  nativeSpeaker age shoeSize  score
195           yes 10  29.48948 50.39916
196            no  8  26.44756 37.54887
197           yes  5  23.98370 32.17017
198           yes  7  27.94532 40.38110
199            no  7  26.89888 33.63669
200            no  8  26.70672 38.68543
```

> summary (readingSkills)

nativeSpeaker	age	shoeSize	score
no :100	Min. : 5.000	Min. :23.17	Min. :25.26
yes:100	1st Qu.: 6.000	1st Qu.:26.23	1st Qu.:33.94
	Median : 8.000	Median :27.85	Median :40.33
	Mean : 7.925	Mean :27.87	Mean :40.66
	3rd Qu.: 9.250	3rd Qu.:29.49	3rd Qu.:47.57
	Max. :11.000	Max. :32.33	Max. :56.71

There 4 columns nativeSpeaker, age, shoeSize, and score. We are going to find out whether a person is a native speaker or not using the other criteria and see the accuracy of the decision tree model developed in doing so.

3. Split the dataset as train and test data in 80 : 20 ratio.

Separating data into training and testing sets is an important part of evaluating data mining models. After a model has been processed by using the training set, we test the model by making predictions against the test set. The testing set contains known values for the attribute that we want to predict, it is easy to determine whether the model's guesses are correct.

> data = sample.split (readingSkills, SplitRatio = 0.8)

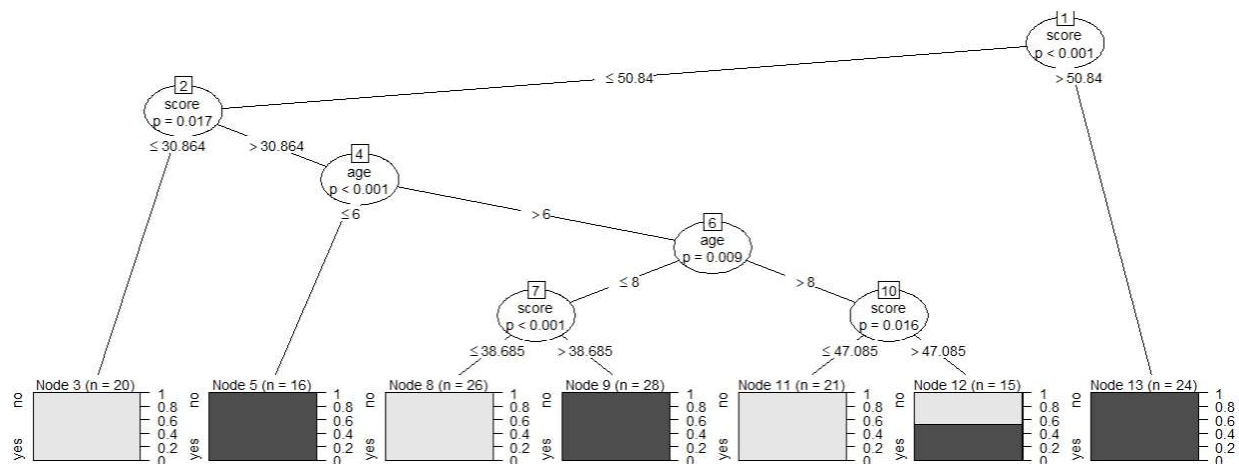
> train\_data <- subset (readingSkills, data==TRUE)

> test\_data <- subset (readingSkills, data=FALSE)

4. Create decision tree and plot the model

> deci\_tree <- ctree (nativeSpeaker~., train\_data)

> plot (deci\_tree)



5. Making the prediction and show the Confusion matrix

> deci\_predict <- predict (deci\_tree, test\_data)

> cmatrix <- table (test\_data\$nativeSpeaker, deci\_predict)

> cmatrix

```

      deci_predict
      no yes
no  24   2
yes   0  24

```

6. Calculate the accuracy of the model

```

> accuracy <- sum(diag (cmatrix)) / sum(cmatrix)
> accuracy
[1] 0.96

```

So, the accuracy-test from the confusion matrix is calculated and is found to be 0.96. Hence this model is found to predict with an accuracy of 96 %.

### ***Advantages of Decision Trees***

- Easy to understand and interpret.
- Does not require Data normalization
- Doesn't facilitate the need for scaling of data
- The pre-processing stage requires lesser effort compared to other major algorithms, hence in a way optimizes the given problem

### ***Disadvantages of Decision Trees***

- Requires higher time to train the model
- It has considerable high complexity and takes more time to process the data
- When the decrease in user input parameter is very small it leads to the termination of the tree
- Calculations can get very complex at times

### Experiment 13:

Implement decision trees using 'iris' dataset using package party and 'rpart'.

### Solution:

#### Decision Tree

Decision Trees are a popular Data Mining technique that makes use of a tree-like structure to deliver consequences based on input decisions. One important property of decision trees is that it is used for both regression and classification. This type of classification method is capable of handling heterogeneous as well as missing data. Decision Trees are further capable of producing understandable rules. Furthermore, classifications can be performed without many computations. Both the classification and regression tasks can be performed with the help of Decision Trees.

#### *Algorithm implementation*

1. Load the relevant packages and call their libraries

```
> library(rpart)
> library(rpart.plot)
```

2. Loading and analyzing the dataset

```
> summary("iris")
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min.	:4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa :50
1st Qu.	:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor:50
Median	:5.800	Median :3.000	Median :4.350	Median :1.300	virginica :50
Mean	:5.843	Mean :3.057	Mean :3.758	Mean :1.199	
3rd Qu.	:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max.	:7.900	Max. :4.400	Max. :6.900	Max. :2.500	

```
> head("iris")
```

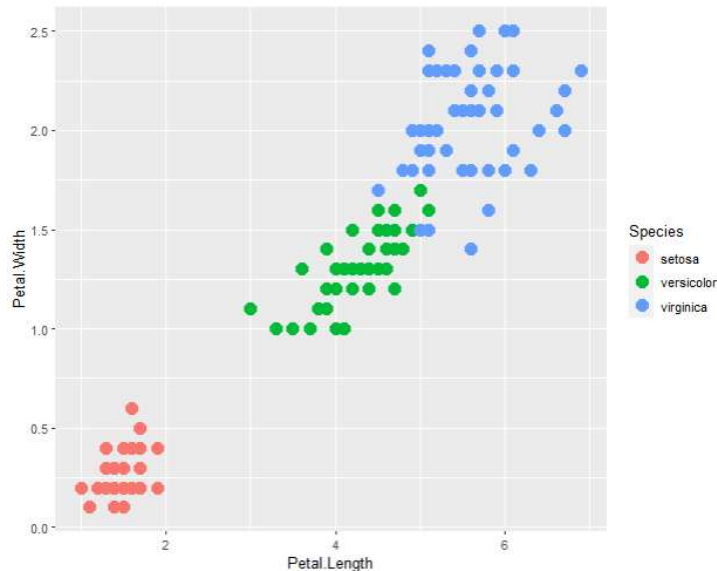
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
> tail("iris")
```



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

```
> ggplot(iris)+aes(Petal.Length,Petal.Width)+geom_point(aes(col=Species),size=4)
```



3. Split the dataset as train and test data in 80 : 20 ratio.

Separating data into training and testing sets is an important part of evaluating data mining models. After a model has been processed by using the training set, we test the model by making predictions against the test set. The testing set contains known values for the attribute that we want to predict, it is easy to determine whether the model's guesses are correct.

```
> data = sample.split (iris, SplitRatio = 0.8)
```

```
> train_data <- subset (iris, data == TRUE)
```

```
> test_data <- subset (iris,data == FALSE)
```

```
> dim (train_data)
```

```
[1] 120  5
```

```
> dim (test_data)
```

```
[1] 30  5
```

4. Create decision tree and plot the model using rpart

```
> deci_tree <- rpart (Species~., train_data, method = "class")
```

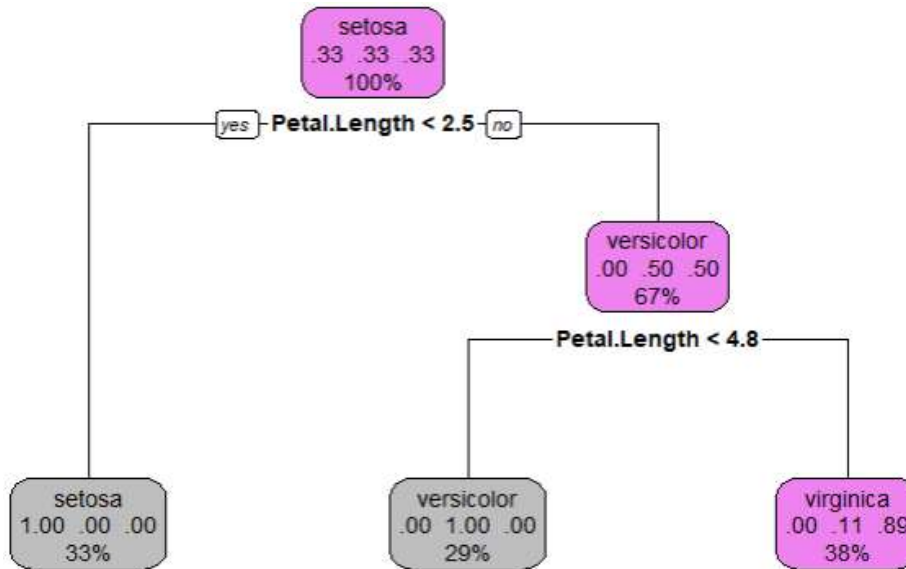
```
> print (deci_tree )
```

```
n= 120
```

```
node), split, n, loss, yval, (yprob)
* denotes terminal node
```

```
1) root 120 80 setosa (0.3333333 0.3333333 0.3333333)
2) Petal.Length< 2.45 40 0 setosa (1.0000000 0.0000000 0.0000000) *
3) Petal.Length>=2.45 80 40 versicolor (0.0000000 0.5000000 0.5000000)
6) Petal.Length< 4.75 35 0 versicolor (0.0000000 1.0000000 0.0000000) *
7) Petal.Length>=4.75 45 5 virginica (0.0000000 0.1111111 0.8888889) *
```

```
> rpart.plot (dec_i_tree, box.col = c ("violet", "grey"))
```



#### 5. Making the prediction and show the Confusion matrix

```
> deci_predict <- predict (dec_i_tree, newdata = test_data[-5], type="class")
```

```
> ctable <- table (dec_i_predict, test_data$Species)
```

```
dec_i_predict setosa versicolor virginica
setosa         10          0          0
versicolor      0          9          1
virginica        0          1          9
```

#### 6. Calculate the accuracy of the model

```
> ctable<-table (dec_i_predict, test_data$Species)
```

```
> accuracy<- sum (diag (ctable)) / sum(ctable)
```

```
> accuracy
```

```
[1] 0.9333333
```

So, the accuracy-test from the confusion matrix is calculated and is found to be 0.933. Hence this model is found to predict with an accuracy of 93.3 %.

#### 7. Confusion matrix using *caret* package

```

> install.packages ("data.table")
> install.packages ("gower")
> install.packages ("caret")
> library (caret)
> ctable <- confusionMatrix (data = deci_predict, test_data$Species)
> ctable

```

#### Confusion Matrix and Statistics

	Reference		
Prediction	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	9	1
virginica	0	1	9

#### Overall Statistics

```

Accuracy : 0.9333
95% CI : (0.7793, 0.9918)
No Information Rate : 0.3333
P-Value [Acc > NIR] : 8.747e-12

```

Kappa : 0.9

Mcnemar's Test P-Value : NA

#### Statistics by Class:

	Class: setosa	Class: versicolor
Sensitivity	1.0000	0.9000
Specificity	1.0000	0.9500
Pos Pred Value	1.0000	0.9000
Neg Pred Value	1.0000	0.9500
Prevalence	0.3333	0.3333
Detection Rate	0.3333	0.3000
Detection Prevalence	0.3333	0.3333
Balanced Accuracy	1.0000	0.9250

	Class: virginica
Sensitivity	0.9000
Specificity	0.9500
Pos Pred Value	0.9000
Neg Pred Value	0.9500
Prevalence	0.3333
Detection Rate	0.3000
Detection Prevalence	0.3333
Balanced Accuracy	0.9250

## Experiment 14:

Use a `Corpus()` function to create a data corpus then Build a term Matrix and Reveal word frequencies.

### Solution:

#### Text Mining

**Text mining** methods allow us to highlight the most frequently used keywords in a paragraph of texts. One can create a **word cloud**, also referred as *text cloud* or *tag cloud*, which is a visual representation of text data. The procedure of creating word clouds is very simple in R if you know the different steps to execute. The text mining package (*tm*) and the word cloud generator package (*wordcloud*) are available in R for helping us to analyze texts and to quickly visualize the keywords as a word cloud.

#### Procedure

1. Install packages for text mining, text stemming, word-cloud generator and color palettes and call their libraries

```
> install.packages("tm")
> install.packages("SnowballC")
> install.packages("wordcloud")
> install.packages("RColorBrewer")
> library("tm")
> library("SnowballC")
> library("wordcloud")
> library("RColorBrewer")
```

2. Load a text file.

```
> text <- readLines (file.choose())
```

3. Load the data as Corpus

```
> docs <- Corpus(VectorSource(text))
```

4. Inspect the content of the document

```
> inspect (docs)
```

5. Text transformation

Transformation is performed using **tm\_map()** function to replace, for example, special characters from the text and replac “/”, “@” and “|” with space:

```

> toSpace <- content_transformer (function (x , pattern ) gsub (pattern, " ", x))
> docs <- tm_map (docs, toSpace, "/")
> docs <- tm_map (docs, toSpace, "@")
> docs <- tm_map (docs, toSpace, "\\")

```

## 6. Clean the text

The `tm_map()` function is used to remove unnecessary white space, to convert the text to lower case, remove numbers, to remove common stop words like “the”, “we”. We also remove numbers and punctuation with `removeNumbers` and `removePunctuation` arguments.

```

> docs <- tm_map (docs, content_transformer (tolower))
> docs <- tm_map (docs, removeNumbers)
> docs <- tm_map (docs, removeWords, stopwords("english"))
> docs <- tm_map (docs, removeWords, c("blabla1", "blabla2"))
> docs <- tm_map (docs, removePunctuation)
> docs <- tm_map (docs, stripWhitespace)

```

## 7. Prepare a term-document matrix

**Document matrix** is a table containing the frequency of the words. Column names are words and row names are documents. The function `TermDocumentMatrix()` from text mining package is used for this.

```

> dtm <- TermDocumentMatrix (docs)
> m <- as.matrix(dtm)
> v <- sort(rowSums(m),decreasing=TRUE)
> d <- data.frame(word = names(v),freq=v)
> head(d, 10)

```

	word	freq
data	data	29
mine	mine	12
analysi	analysi	10
techniqu	techniqu	10
pattern	pattern	8
classif	classif	8
cluster	cluster	8
method	method	7
olap	olap	6
cours	cours	5

## 8. Generate the Word cloud

```

> wordcloud(words=d$word, freq = d$freq, min.freq = 1, max.words = 200,
            random.order = FALSE, rot.per = 0.35, colors = brewer.pal (8, "Dark2"))

```

