LAB MANUAL

STATISTICAL VISUALIZATION USING R

List of Experiments

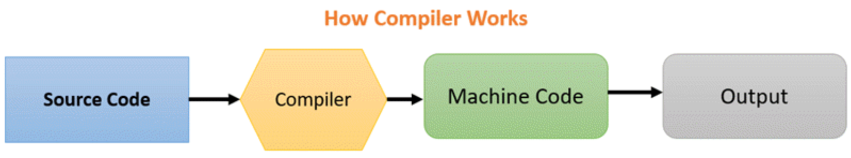
1. Demonstrate the basic math functions in R
2. Demonstrate Vector operations in R
3. Demonstrate Matrix operations in R
4. Demonstrate Array operations in R
5. Demonstrate Dataframes in R
6. Demonstrate Lists in R
7. Illustrate the following controls statements in R
    i. if and else
    ii. ifelse
    iii. switch
8. Demonstrate for and while loops in R
9. Demonstrate importing and exporting data using R
10. Illustrate the descriptive statistics using summary() in R
11. Demonstrate the following statistical distribution functions in R:
    i. Normal Distribution
    ii. Binomial Distribution
    iii. Poisson Distribution
    iv. Chi Square Distribution
12. Illustrate the following basic graphs in R:
    i. Bar plots
    ii. Pie Charts
    iii. Histograms
    iv. Kernel density plots
    v. Boxplots
    vi. Dotplots
13. Illustrate the Correlation and Covariance analysis using R
14. Illustrate the different types of t-tests using R
15. Illustrate the ANOVA test using R

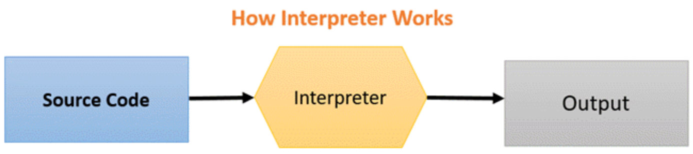
Introduction to R

R is an open source programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analytics tools. It is modelled after S & S-plus, developed at AT&T labs in late 1980s. R project was started by Robert Gentleman and Ross Ihaka Department of Statistics, University of Auckland (1995). Currently maintained by R core development team – an international team of volunteer developers (since 1997).

Features of R:

- R is an interpreted language; users typically access it through a command-line interpreter.
- 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.





R & R Studio Installation:

A comprehensive R Archive Network is present at https://cran.r-project.org/, where once can download R for any platform.

Installing R on windows PC :

- Use internet browser to point to: http://mirror.aarnet.edu.au/pub/CRAN
- Under the heading Precompiled Binary Distributions, choose the link Windows.
- Next heading is R for Windows; choose the link base.
- Click on download option(R 3.4.1 for windows).
- Save this to the folder C:\R on your PC.
- When downloading is complete, close or minimize the Internet browser.
- Double click on R 3.4.1-win32.exe in C:\R to install.

Installing R on Linux:

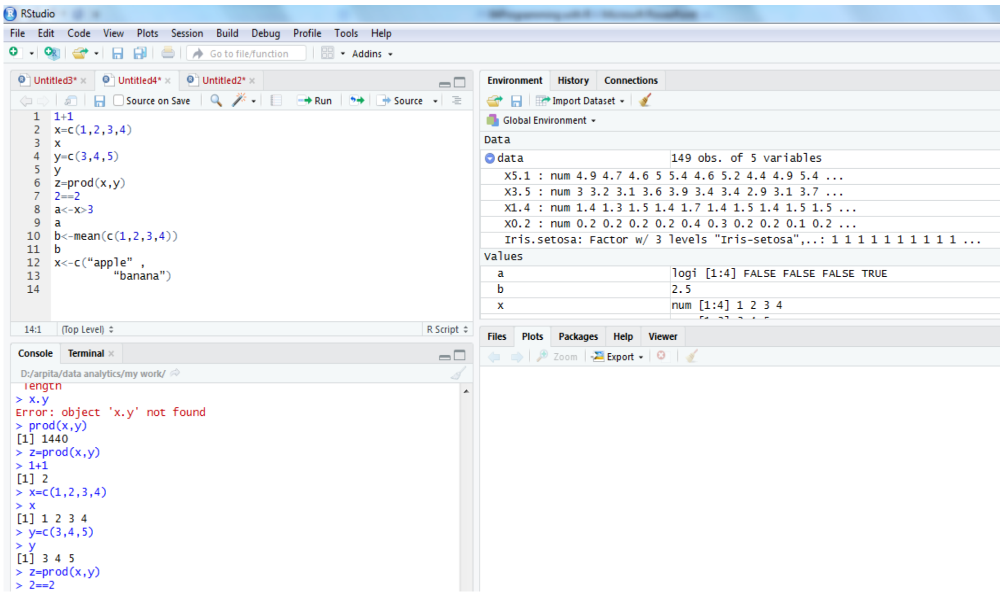
- Use the command: sudo apt-get install r-base-core

R-Studio's IDE is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace.

Installing R-Studio:

- Go to www.rstudio.com and click on the "Download RStudio" button.

- Click on "Download RStudio Desktop. "
- Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.



```
1+1
x=c(1,2,3,4)
x
y=c(3,4,5)
y
z=prod(x,y)
2==2
a<-x>3
a
b<-mean(c(1,2,3,4))
b
x<-c("apple" ,
         "banana")
```

D:/arpita/data analytics/my work/

```
 length
> x.y
Error: object 'x.y' not found
> prod(x,y)
[1] 1440
> z=prod(x,y)
> 1+1
[1] 2
> x=c(1,2,3,4)
> x
[1] 1 2 3 4
> y=c(3,4,5)
> y
[1] 3 4 5
> z=prod(x,y)
> 2==2
```

```
Data
data                        149 obs. of 5 variables
  X5.1 : num 4.9 4.7 4.6 5 5.4 4.6 5.2 4.4 4.9 5.4 ...
  X3.5 : num 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 ...
  X1.4 : num 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 ...
  X0.2 : num 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 ...
  Iris.setosa: Factor w/ 3 levels "Iris-setosa",..: 1 1 1 1 1 1 1 1 1 1 ...
Values
  a                         logi [1:4] FALSE FALSE FALSE TRUE
  b                         2.5
  x                         num [1:4] 1 2 3 4
```

Figure 1: R-Studio IDE

Exp. 1: Demonstrate the basic math functions in R

Aim: To understand about operators, variable, built-in constants, built-in functions in R and how to execute basic mathematical operations in R.

Operators: An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Operator Type	**Operators**
Arithmetic Operators	+, -, *, /, %%, %/%, ^
Relational Operators	<, >, ==, <=, >=, !=
Logical Operators	&, |, !, &&, ||
Assignment Operators	= or <- or <<- & ->, ->>
Miscellaneous Operators	:, %in%, %*%

Arithmetic Operators:

```
> 3+5
[1] 8
> 12 + 3 / 4 – 5 + 3*8
[1] 31.75
> (12 + 3 )/ 4 – 5 + 3*8
[1] 22.75
> ((12+3)/(4 – 5) +3)*8
[1] -96
> 256%%13
[1] 9
> 256%/%13
[1] 19
>6^3
[1] 216
```

Relational Operators:

```
> 3<5
[1] TRUE
> 6>9
[1] FALSE
> 5==6
[1] FALSE
> 12!=52
```

```
[1] TRUE
>5<=6
[1] TRUE
>125>=50
[1] TRUE
```

Logical Operators:

```
> TRUE&TRUE
[1] TRUE
> TRUE&FALSE
[1] FALSE
> FALSE&TRUE
[1] FALSE
> FALSE&FALSE
[1] FALSE
> TRUE|TRUE
[1] TRUE
> TRUE|FALSE
[1]  TRUE
> FALSE|TRUE
[1]  TRUE
> FALSE|FALSE
[1] FALSE
>!TRUE
[1] FALSE
```

Variables: A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Assignment Operators:

```
> x=10
>y<-15
>18->z
```

Built-in Constants

```
>LETTERS                ##the 26 upper-case letters of the Roman alphabet;
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
[18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
>letters                        ##the 26 lower-case letters of the Roman alphabet;
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
>month.abb                      ##the three-letter abbreviations for the English month names;
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov""Dec"
>month.name                     ##the English names for the months of the year;
[1] "January"   "February"  "March"     "April"     "May"
[6] "June"      "July"      "August"    "September" "October"
[11] "November"  "December"
>pi                             ##the ratio of the circumference of a circle to its diameter.
[1] 3.141593
```

Built-in Functions (Math)

```
>abs(-4)
[1] 4
>sqrt(25)
[1] 5
>ceiling(5.7)
[1] 6
>ceiling(4.2)
[1] 5
>floor(3.3)
[1] 3
>floor(10.75)
[1] 10
>trunc(2.5)
[1] 2
>round(3.475, digits=2)
[1] 3.48
>log(2,10)
[1] 0.30103
>log(2, base=10)
[1] 0.30103
>log10(2)
[1] 0.30103
>log(2)
```

```
[1] 0.6931472
>exp(4)
[1] 54.59815
>cos(pi)
[1] -1
>sin(pi)
[1] 1.224606e-16~0
>tan(pi/4)
[1] 1
```

Exp. 2: Demonstrate Vector operations in R

Aim: To experiment with vectors, the most basic data type/objects of R.

First of all, What is meant by Data Type?A variable can be used for storing various kinds of data like character, wide character, integer, floating point, double floating point, Boolean etc.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 basic data types are:

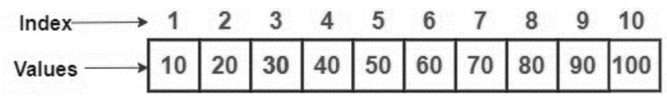
- Boolean/Logical : TRUE, FALSE
- Numeric : 2, 5, 10.6
- Integer : 2L, 5L, 10L
- Complex : 2+5i
- Character : "a", "u", "all", "SVEC", "TRUE", '10.6', 'VASAVI'

The advanced data types or data structures in R Language are:

- Vectors
- Lists
- Matrices
- Arrays
- Data Frames

Vectors:

The most basic R data objects and there are six types of atomic vectors. Even when you write just one value in R, it becomes a vector of length 1.Vectors in R are used to hold multiple data values of the same type.



Creating a vector:There are different ways of creating vectors. Generally, we use 'c' to combine different elements together.We can also create a numerical vector using colon operator. A colon operator always generate a sequence with step size 1 or -1. To create a numerical sequence vector with a specific step size we can useseq().

Ex:

```
>X <- c(61, 4, 21, 67, 89, 2)

>X

[1] 61  4 21 67 89  2

>Y=c("S", "V", "E", 'C')

> Y

[1] "S" "V" "E" "C"

>Z=c("SVEC", "CSE")

> Z

[1] "SVEC" "CSE"

>v1 <- 5:13
```

```
>v1
[1]  5  6  7  8  9 10 11 12 13
>v2 <- 6.6:12.6
>v2
[1]  6.6  7.6  8.6  9.6 10.6 11.6 12.6
>v3 <- 11.4:3.8
>V3
[1] 11.4 10.4  9.4  8.4  7.4  6.4  5.4  4.4
>v3 <- seq(5, 9)
>v3
[1] 5 6 7 8 9
>v4 <- seq(5, 9, 0.4)
>v4
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
>v5<- seq(5, 9, by=0.4)
>v5
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Accessing a Vector: Elements of a Vector are accessed using indexing. The [ ] brackets are used for indexing. Indexing starts with position 1.Giving a negative value in the index drops that element from result.TRUE, FALSE can also be used for indexing.

Ex:

```
>X <- c(61, 4, 21, 67, 89, 2)
>X[1]
[1] 61
>X[4]
[1] 67
>X[c(1,3)]
[1] 61  21
>X[-2]
[1] 61 21 67 89  2
>X[c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE)]
[1] 61  4  8
```

Manipulating a Vector:An element of a vector can be altered using the indexing.Two vectors of same length can be added, subtracted, multiplied or divided element by element giving the result as a vector output. If the length of the vectors are not same the short vector will be replicated. There are various inbuilt functions that can be performed on a vector.

Ex:

```
>V1 <- c(61, 4, 21, 67, 89, 2)
>V1[2]=10
>V1[c(1,3)]=c(25,36)
>V1
[1] 25 10 36 67 89  2
>V2 <- c(6, 24, 43, 2, 8, 2)
>V3<-c(10,25)
>V1+V2
[1] 31 34 79 69 97  4
>V1*V3
[1] 250  250  756 2144  890   50
Warning message:
In V1 * V3 :
longer object length is not a multiple of shorter object length
>length(V1)                    #Length of the Vector
[1] 6
>sort(V2)                      #Shorts elements of a Vector
[1]  2  2  6  8 24 43
>rep(V1,2)                     #Creates a vector by Repeating given Vector.
[1] 25 10 36 67 89  2 25 10 36 67 89  2
>is.vector(V1)                 #Tells whether the object is vector or not
[1] TRUE
>any(V1>50)          #Checks whether any element satisfy the given condition
[1] TRUE
>all(V1>10)                    #Checks whether all elements satisfy thegiven condition
[1] FALSE
```

Exp. 3: Demonstrate Matrix operations in R

Aim: To experiment with matrices, a two-dimensional data type/objects of R.

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout.They contain elements of the same atomic types.

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Creating and Accessing a Matrix: A Matrix is created using the matrix() function. We can use the indexes (starting with 1) to access a row or a column or an element. One can also createan empty Matrix without data and specify the elements individually later. We can add names to rows and columns and use them to access a row or a column or an element.

Syntax*: matrix(data, nrow, ncol, byrow, dimnames)*

Where

data is the input vector which becomes the data elements of the matrix.

nrow is the number of rows to be created.

ncol is the number of columns to be created.

byrow is a logical clue. If TRUE then the input vector elements are arranged by row.

dimname is the names assigned to the rows and columns.

Ex:

```
>P <- matrix(c(1,2,3,4,5,6,7,8,9), nrow = 3, byrow = TRUE)
>P
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
>P[1,]                          #First Row
[1] 1 2 3
>P[,3]                          #Third Col
[1] 3 6 9
>P[2,3]                         #2nd Row, 3rd Col Element
[1] 6
>rownames=c("r1", "r2", "r3")
>colnames=c("c1", "c2", "c3")
>P <- matrix(c(1,2,3,4,5,6,7,8,9), nrow = 3, byrow = TRUE,
                                dimname=list(rownames,colnames))
```

```
>P
  c1 c2 c3
r1  1  2  3
r2  4  5  6
r3  7  8  9
>P[“r1”,]                                   #Row with “r1” as name, i.e., 1st Row
c1 c2 c3
1  2  3
```

One can join multiple vectors or matrices row wise or column wise using rbind() or cbind() to create a matrix.

```
>X=c(1,2,3,4,5)
>Y=11:15
>Z=c(24,10,23,22,43)
>cbind(X, Y, Z)
    X  Y  Z
[1,] 1 11 24
[2,] 2 12 10
[3,] 3 13 23
[4,] 4 14 22
[5,] 5 15 43
>rbind(X, Y, Z)
[,1] [,2] [,3] [,4] [,5]
X    1    2    3    4    5
Y   11   12   13   14   15
Z   24   10   23   22   43
```

Matrix Operations: We can use any operators to perform element by element operations on matrices of equal dimensions also. There are also some matrix specific operations like dim(), t(), det() etc.

```
>M <- matrix(c(1,2,3,4), nrow = 2, byrow = TRUE)
>P <- matrix(c(1,2,3,4,5,6,7,8,9), nrow = 3, byrow = TRUE)
>Q<- matrix(c(11,12,13,14,15,16,17,18,19), nrow = 3, byrow = TRUE)
>P+Q
[,1] [,2] [,3]
[1,]   12   14   16
[2,]   18   20   22
[3,]   24   26   28
```

```
>M
    [,1] [,2]
[1,]    1    2
[2,]    3    4
>t(M)                          #Transpose
    [,1] [,2]
[1,]    1    3
[2,]    2    4
>diag(M)                       #diagnol of M
[1] 1 4
>I=diag(3)                     #identity matrix of size 3x3
>I
    [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
>dim(M)                        #dimensions of M
[1] 2 2
>det(M)                        #deteriment of M
[1] -2
>solve(M)                      #Inverse of M
[,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

Exp. 4: Demonstrate Array operations in R

Aim: To create an Array, access and manipulate the elements of an array

An Array is a multidimensional vector. They contain elements of the same atomic types.An array in R can have one, two or more dimensions.A two-dimensional array is similar to a matrix.

Creating and Accessing an Array: They are created with an array function. Use the indeces (starting with 1) to access anydimension.

Syntax: *myarray<- array(data, dim, dimnames)*

Where

data is the input vector which becomes the data elements of the matrix.

dim is a vector of dimensions. Ex: c(2,3,4) means 2 is row index, 3 is column index, 4 is outer dimension.

dimname is the names assigned to the rows and columns.

Ex:

```
>M <- array(c(1,2,3,4,5,6,7,8), dim=c(2,2,2))

>M

, , 1

     [,1] [,2]

[1,]    1    3

[2,]    2    4

, , 2

     [,1] [,2]

[1,]    5    7

[2,]    6    8

> v1 <- 1:9

> v2 <- 11:19

>col.names<- c("Item","Serial","Size")

>row.names<-c("Server","Network","Firewall")

>matrix.names<- c("DataCenter EU","DataCenter US")

>result <- array(c(v1,v2),dim=c(3,3,2), dimnames= list(row.names, col.names,
matrix.names))

>result

, ,DataCenter EU

        Item Serial Size

Server     1      4    7

Network    2      5    8
```

```
Firewall    3      6    9

, ,DataCenter US

        Item Serial Size

Server      11     14   17

Network    12     15   18

Firewall   13     16   19

>A <- array(1:12, dim=c(2,2,3))

>A[1,,]

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

[1,]    1    5    9

[2,]    3    7   11

>A[,2,]

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

[1,]    3    7   11

[2,]    4    8   12

>A[,,3]

     [,1] [,2]

[1,]    9   11

[2,]   10   12

>A[1,2,]

[1]  3  7 11

>A[1,2,2]

[1] 7

>result["Server", ,"DataCenter EU"]

  Item Serial   Size

     1      4      7
```

Calculations Across Array Elements:We can do calculations across the elements in an array using the apply() function.

Syntax: *apply(x, margin, fun)*

Where

x is an array.

margin is the name of the data set used.

fun is the function to be applied across the elements of the array.

Ex:

```
>A=array(1:6,dim=c(2,3))

>apply(A, c(2), sum)

[1]  3  7 11

>apply(A, c(2), function(x) x+3)

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

[1,]    4    6    8

[2,]    5    7    9
```

Exp. 5: Demonstrate Data frames in R

Aim: To create a Data frame, access and manipulate the elements of a Data frame

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.
- For Example:

```
emp_id      emp_name     salary

     11           Rick          623.30

12          Dan           515.20

13          Michelle     611.00

     14           Ryan          729.00

15          Gary          843.25
```

Creating and Exploring a Dataframe: A data frame is created with the data.frame() function. Once created or imported, the data frame's structure and statistical summary can be seen using str() and summary() functions.

Syntax: **mydata<- data.frame(col1, col2, col3,…)**

where col1, col2, col3, … are column vectors of any type (such as character,numeric,or logical).

Ex:

```
>emp_id = 11:15

>emp_name = c("Rick","Dan","Michelle","Ryan","Gary")

>salary = c(623.3,515.2,611.0,729.0,843.25)

>emp.data=data.frame(emp_id, emp_name, salary)

>emp.data

emp_id emp_name        salary

1    11            Rick              623.30

2    12            Dan               515.20

3    13  Michelle          611.00

4    14            Ryan              729.00

5    15            Gary              843.25

>str(emp.data)

'data.frame':       5 obs. of  3 variables:

 $ emp_id  :int  11 12 13 14 15
 $ emp_name: chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary  :num  623 515 611 729 843
>summary(emp.data)
emp_id  emp_name                  salary
Min.:11 Length:5                  Min.   :515.2
 1st Qu.:12        Class :character 1st Qu.:611.0
Median :13        Mode  :character           Median :623.3
 Mean   :13                   Mean   :664.4
 3rd Qu.:14                                  3rd Qu.:729.0
 Max.   :15                         Max.   :843.2
>names(emp.data)                                           # Column Names
[1] "emp_id"   "emp_name" "salary"
>rownames(emp.data)                                        #Row Names if Present
[1] "1" "2" "3" "4" "5"
>dim(emp.data)                                             #Dimensions
[1] 5 3
>nrow(emp.data)                                            #No of Rows in DF
[1] 5
>ncol(emp.data)                                            #No of Columns in DF
[1] 3
>head(emp.data,3)                                          # Starting Rows
emp_id emp_name salary
1     11     Rick  623.3
2     12      Dan  515.2
3     13 Michelle  611.0
>tail(emp.data)                                            #Ending Rows
emp_id emp_name salary
1     11     Rick 623.30
2     12      Dan 515.20
3     13 Michelle 611.00
4     14     Ryan 729.00
5     15     Gary 843.25
```

Accessing a Dataframe:One can accessa specific column or a group of columns, a specific row or a group of rows, or a single element using the indices or the names.

Ex:

```
>emp.data[1:2]
emp_id emp_name
1     11     Rick
2     12      Dan
3     13 Michelle
4     14     Ryan
5     15     Gary
>emp.data[c("emp_name","salary")]
emp_name salary
1     Rick 623.30
2      Dan 515.20
3 Michelle 611.00
4     Ryan 729.00
5     Gary 843.25
>emp.data[[1]]
[1] 11 12 13 14 15
>emp.data[["emp_name"]]
[1] "Rick"     "Dan"      "Michelle" "Ryan"     "Gary"
>emp.data$emp_name
[1] "Rick"     "Dan"      "Michelle" "Ryan"     "Gary"
>emp.data$1                                              #Wont Work
Error: unexpected numeric constant in "emp.data$1"
>emp.data[,1]
[1] 11 12 13 14 15
>emp.data[1,]
emp_id emp_name salary
1     11     Rick  623.3
>emp.data[1:3,]
emp_id emp_name salary
1     11     Rick  623.3
2     12      Dan  515.2
3     13 Michelle  611.0
>emp.data[1, "emp_name"]
```

```
[1] "Rick"

>emp.data[1, 2]

[1] "Rick"

>emp.data[1:3, 2]

[1] "Rick"     "Dan"      "Michelle"

>emp.data[1, 2:3]

emp_name salary

1     Rick  623.3
```

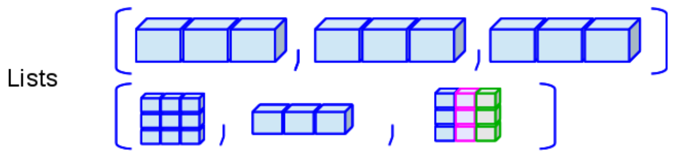
We can expand a dataframe by adding one or more columns to an existing dataframe, but new rows cannot be added directly. But can create a separate dataframe with the new rows and use rbind to join old and new dataframes.

```
>emp.data$dept<-c("IT","Operations","IT","HR","Finance") #Adding a new column.

>emp_id = 16:18                                           #Adding new Rows

>emp_name = c("Rashmi","Pranab","Tushar")

>salary = c(578.0,722.5,632.8)

>dept = c("IT","Operations","Finance")

>emp.newdata=data.frame(emp_id,emp_name, salary, dept)

>emp.finaldata=rbind(emp.data,emp.newdata                 #Updated new dataframe
```

Exp. 6: Demonstrate Lists in R.

Aim: To create a List, access and manipulate the elements of a List.

Lists are heterogeneous data structure objects of R, which contain elements of different types like − numbers, strings, vectors, etc.



Creating and Accessing of Lists: List is created using list() function.The list elements can be given names and they can be accessed using these names.List elements can be access using the index or using names if assigned.List elements can be manipulated using the index or using names if assigned.Insertion can be done at the end by using the index or a name.

Ex:

```
>L=list("Red", "Green", c(21,32,11), 119.1)
>L
[[1]]
[1] "Red"
[[2]]
[1] "Green"
[[3]]
[1] 21 32 11
[[4]]
[1] 119.1
>names(L) <- c("c1", "c2", "v", "N")
>L
$c1
[1] "Red"
$c2
[1] "Green"
$v
[1] 21 32 11
$N
[1] 119.1
>L[1]
$c1
[1] "Red"
```

```
>L[2]
$c2
[1] "Green"
>L["c1"]
$c1
[1] "Red"
>L[[1]]                         #[] gives a sublistwhere as [[]] gives elements of list.
[1] "Red"
>L[[2]]
[1] "Green"
>L$c1
[1] "Red"
>L[5]=100                       #Replaces the 5th element of list with 100.
>L[1]=NULL                      #Delets the 1st element of list L.
>merged.list<- c(list1,list2)   #Creates a new list by merging elements of both lists.
>unlist(L)
   c1      c2      v1      v2      v3       N
"Red"  "Green"   "21"    "32"    "11" "119.1"
```

Add-on Exp: Some Basic Fundamentals of R

Aim: To understand how to read inputs during runtime from the user and print outputs. Also, we are going to see how to write scripts.

R has readline() and scan() methods to take input from the user during runtime. It also has print(), sprintf(), and cat() methods to print output.

readline(): Takes only one input in string format.Type conversion functions must be used to convert the inputs to the desired data type.

Syntax: *var=readline("Message to User")*

Ex:

```
>s=readline("Enter Your Name: ")

Enter Your Name: VASAVI

>s

[1] "VASAVI"

>a=readline("Enter a Number: ")

Enter a Number: 3

>a

[1] "3"

>class(a)

[1] "character"

>as.integer(a)

[1] 3
```

scan(): This method reads data in the form of a vector or list. The interpreter will take inputs continuously till user presses the Enter key twice. The inputs expected are, by default, numeric.

Syntax: *var=scan(file="stdin",what="numeric") or var=scan()*

Ex:

```
>a=scan()                    #Enter a vector of numeric type

1: 1

2: 2

3: 3

4: 2

5:

Read 4 items

>a

[1] 1 2 3 2
```

print(): Most common method to print output in R.

Syntax: print(<a string>) or print(<var>)

Ex:

```
>print(“CSE”)
[1] "CSE"
>print(a)
[1] 1 2 3 2
>s= "VASAVI"
>print(paste(“My name is”, s))
[1] "My name is VASAVI"
>a=4
>print(sprintf(“The Number is %d”,a))          # To print output as in C language.
[1] "The Number is 4"
```

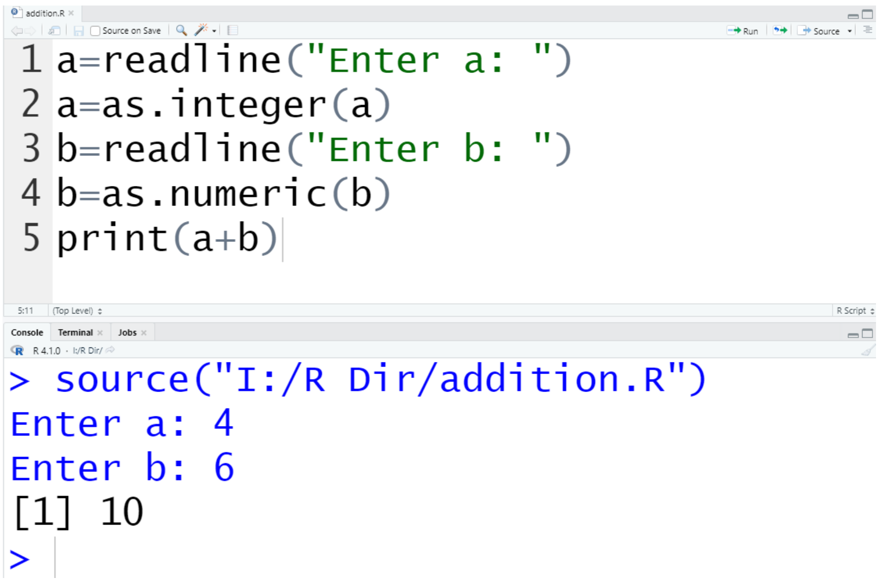
cat(): Another way to print output in R. Converts the arguments in character format.

Syntax: *cat(arg1, arg2, arg 3,…)*

Ex:

```
>Dep= "CSE"
>cat(“Department of ”,D)
Department of  CSE
```

R Scrips: Use any text editor or inbuilt script editor to type in the code.Save the file using ".R" extension.Run the script by clicking on the source.



```
a=readline("Enter a: ")
a=as.integer(a)
b=readline("Enter b: ")
b=as.numeric(b)
print(a+b)
```

```
> source("I:/R Dir/addition.R")
Enter a: 4
Enter b: 6
[1] 10
>
```

Exp. 7: Illustrate the following controls statements in R.

a) if and else b)ifelsec)switch

Aim:To understand the working of different control statements in R.

Control statements allow us to control the flow of our programming and cause different things to happen depending on the values of tests. The test's result is logical TRUE or FALSE.The main control statements are

- if-else
- ifelse
- switch.

IF-ELSE: A control structure with if-else can have an "if" block followed by multiple optional "else if" blocks and followed by an optional "else" block.We can have a control structure with just an if block.

Syntax:

```
if (Boolean Expression){
        <Code Block 1>
} else   if(Boolean Expression){
        <Code Block 2>
        }
   :
   :
}else if (Boolean Expression){
        <Code Block n-1>
        }else{
        <Code Block n>
        }
```

Ex:

Script: age.R

```
age=readline("Eneter Age: ")
age=as.integer(age)
if (age<=12){
print("Child")
}else if(age<=19){
print("Teenager")
}else if (age<=30){
print("Adult")
}else if(age<=45){
print("Middle Aged")
}else {
print("Old")
  }
```

Console:

```
Enter Age: 25
[1] "Adult"
```

ifelse(): A ternary statement that is a one-line equivalent of a simple if-else structure.

Syntax: *ifelse (Boolean Expression, <Exp 1>,<Exp 2>)*

Ex:

```
>a=-8
>ifelse (a>=0, print("+ve"),print("-ve"))
[1] "-ve"
```

switch(): If we have multiple cases to check, writing “else if” repeated, can be cumbersome and inefficient.This is where the switch is most useful. There are basically two ways in which one of the cases is selected. One is using the Index to select a case. If the cases are values like a character vector, and the expression is evaluated to a number than the expression's result is used as an index to select the case.When the cases have both case value and output value like ["case_1"="value1"], then the expression value is matched against case values. If there is a match with the case, the corresponding value is the output.

Syntax: *switch(expression, case1, case2, case3....)*

Ex:

```
>switch(3,"Shubham","Nishka","Gunjan","Sumit")
[1] "Gunjan"
```

Script:

```
a=10
b=2
cat("Enter Your Choice:\n1 for add \n2 for sub \n3 for Div \n4 for mul")
y=readline()
switch(y,
    "1"=cat("Addition=",a+b),
    "2"=cat("Subtraction =",a-b),
    "3"=cat("Division= ",a/b),
    "4"=cat("multiplication =",a*b)
)
```

Console:

```
Enter Your Choice:
1 for add
2 for sub
3 for Div
4 for mul
2
Subtraction = 8
```

Exp. 8: Demonstrate for and while loops in R

Aim: To understand the working of loops in R

Loops: There may be a situation when you need to execute a block of code several times. In general, statements are executed sequentially.A loop statement allows us to execute a statement or group of statements multiple times.R programming language provides for, while, and repeat to handle looping requirements.

For: Unlike traditional languages, in R loops, especially for loops are used to iterate over elements of a vector, list or data.frame.

Syntax: *for (value in vector) {*
statements
}

Ex:

Script:

```
V=c("a", "b", "c", "d")
for (x in V) {
        print(x)
        }
```

Console:

```
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

While: The while loop executes the same code repeatedly until a stop condition is met.

Syntax: *while (Boolean expression) {*
statements
}

Ex:

Script:

```
i=1
while (i<=5) {
print(i)
  i=i+1
}
```

Console:

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Loop control statements change execution from its normal sequence.R supports break & next control statements.When the break statement is encountered inside a loop, the loop is immediately terminated.The next statement in R programming language is useful when we want to skip the current iteration of a loop without terminating it.

Ex:

Script (break):

```
i=0
while (TRUE) {
print(i)
  i=i+2
if (i==10){
break
```

```
  }
}
```

Console:

```
[1] 0
[1] 2
[1] 4
[1] 6
[1] 8
```

Script(next):

```
for(i in 1:10){
if(i%%2){
next
  }
print(i)
}
```

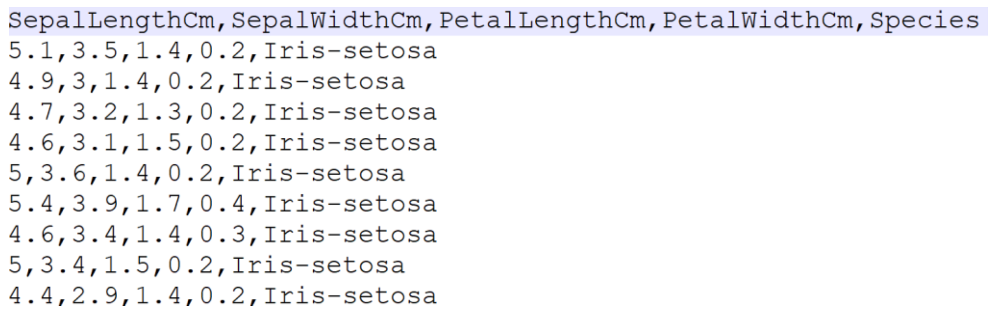
Console:

```
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

Exp. 9: Demonstrate importing and exporting data using R

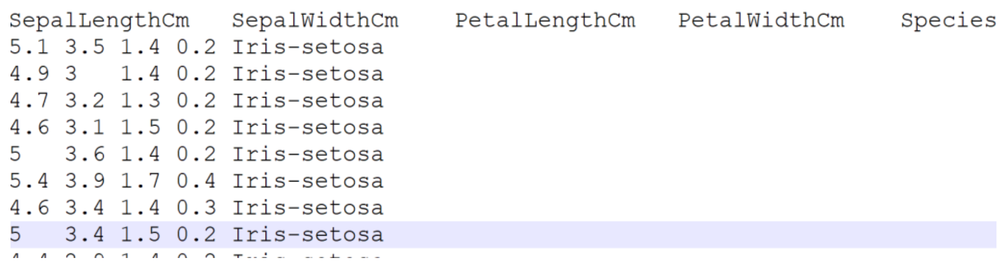
Aim: To understand the process of importing and exporting of data in R.

Data comes in different formats from different sources.There are numerous ways to get data, the most common is probably reading comma separated values (CSV) files.We have to import data from various types of files.The most important ones apart from CSV are delimited files (.txt), .xlsx. All the imported data was saved as dataframe in R.



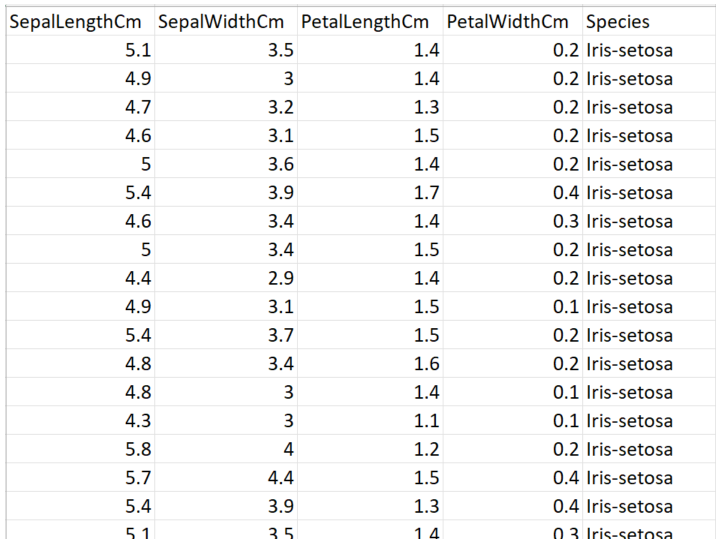
```
SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
```

IRIS CSV FILE



```
SepalLengthCm   SepalWidthCm    PetalLengthCm   PetalWidthCm    Species
5.1 3.5 1.4 0.2 Iris-setosa
4.9 3   1.4 0.2 Iris-setosa
4.7 3.2 1.3 0.2 Iris-setosa
4.6 3.1 1.5 0.2 Iris-setosa
5   3.6 1.4 0.2 Iris-setosa
5.4 3.9 1.7 0.4 Iris-setosa
4.6 3.4 1.4 0.3 Iris-setosa
5   3.4 1.5 0.2 Iris-setosa
```

IRIS TAB DELIMITED FILE



SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa

IRIS XLSX FILE

The best way to read data from a CSV file1 is to use read.table.One can also use read.csv which are wrapper around read.table with the sep argument preset to a comma (,).read.delim() is another wrapper on read.table with the sep argument preset to a tab space (\t)

Syntax: *read.table(file, header = FALSE, sep = "", dec = ".")*

Ex:

```
>iris_csv1=read.table("iris.csv", header = TRUE, sep=",")
```

Syntax: *read.csv(file, header = TRUE, sep = ",", dec = ".", ...)*

Ex:

```
>iris_csv2=read.csv("iris.csv")
```

Syntax: *read.delim(file, header = TRUE, sep = "\t", dec = ".", ...)*

Ex:

```
>iris_tab=read.delim("iris.txt")

>str(iris_tab)                    #str(iris_csv1) or str (iris_csv2)

'data.frame':    150 obs. of  5 variables:
 $ ï..SepalLengthCm: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ SepalWidthCm    : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ PetalLengthCm   : num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ PetalWidthCm    : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species         : chr  "Iris-setosa" "Iris-setosa" "Iris-setosa" "Iris-setosa" ..
```

The readr package provides a family of functions for reading text files.The most commonly used will be read_delim for reading tab delimited files.Note that eventhoughone can use read.table() or read.csv() or read.delim() for the same but readr package reads data at high speed. To use readr, first import the package readr using library(readr) command.If delimiter was not mentioned, the read_delim() function will try to find it out on its own. Note that, one has to install the package if not done erlier using install.packages() command.

Syntax: *read_delim(file, delim = NULL, col_names = TRUE, ...)*

Ex:

```
>read_tab=read_delim("iris.txt")
```

The readxl package provides functions for reading .xls, .xlsx files.

Syntax: *read_excel(file, sheet=NULL, range=NULL, ...)*

Ex:

```
>iris_xlsx=read_excel("iris.xlsx")
```

One can import data from a website using the read.table, read.csv, read.delim.

Ex:

```
>theUrl<- "http://www.jaredlander.com/data/TomatoFirst.csv"

>tomato <-read.table(file=theUrl, header=TRUE, sep=",")

>str(tomato)

'data.frame':    16 obs. of  11 variables:
 $ Round     : int  1 1 1 1 2 2 2 2 3 3 ...
 $ Tomato    : chr  "Simpson SM" "Tuttorosso (blue)" "Tuttorosso (green)" "La Fede SM DOP" ...
 $ Price     : num  3.99 2.99 0.99 3.99 5.49 4.99 3.99 3.99 4.53 NA ...
 $ Source    : chr  "Whole Foods" "Pioneer" "Pioneer" "Shop Rite" ...
 $ Sweet     : num  2.8 3.3 2.8 2.6 3.3 3.2 2.6 2.1 3.4 2.6 ...
 $ Acid      : num  2.8 2.8 2.6 2.8 3.1 2.9 2.8 2.7 3.3 2.9 ...
 $ Color        : num  3.7 3.4 3.3 3 2.9 2.9 3.6 3.1 4.1 3.4 ...
 $ Texture      : num  3.4 3 2.8 2.3 2.8 3.1 3.4 2.4 3.2 3.3 ...
 $ Overall      : num  3.4 2.9 2.9 2.8 3.1 2.9 2.6 2.2 3.7 2.9 ...
 $ Avg.of.Totals: num  16.1 15.3 14.3 13.4 14.4 15.5 14.7 12.6 17.8 15.3 ...
 $ Total.of.Avg :num  16.1 15.3 14.3 13.4 15.2 15.1 14.9 12.5 17.7 15.2 ...
```

We can use any of the following methods to write data into a file.

- write.table()
- write.csv()
- write_delim()
- write_csv()
- write_tsv()
- write.xlsx()

Syntax: *write.table(x, file, append = FALSE, sep = " ", dec = ".", row.names = TRUE, col.names = TRUE)*

Ex:

```
>write.table(iris_csv,"irisv2.csv",sep=",",row.names = FALSE)
```

Syntax: *write.csv(x, file, append = FALSE, sep = " ,", dec = ".", row.names = TRUE, col.names = TRUE, ...)*

Ex:

```
>write.csv(iris_csv2,"irisv3.csv")
```

Syntax: *write_delim(x, file, delim=" ", ...)*

Ex:

```
>write_delim(iris_csv,"irisv2.txt",delim=",")
```

Syntax: *write.xlsx(x, file, sheetname="Sheet1", col.names = TRUE, row.names = TRUE, append = FALSE)*

Ex:

```
>write.xlsx(iris_tab,"irisv2.xls")

>write.xlsx(iris_tab,"irisv2.xlsx")
```

Exp. 10: Illustrate the descriptive statistics using summary() in R

Aim: To generate the statistical measurements for a given dataset using summary() method in R.

A descriptive statistic is a summary statistic that quantitatively describes or summarizes features from a collection of information.They helps describe, show or summarize data in a meaningful way such that, for example, patterns might emerge from the data.Descriptive statistics do not, however, allow us to make conclusions beyond the data we have analyzed or reach conclusions regarding any hypotheses we might have made, they are simply a way to describe our data.

Descriptive statistics are very important because if we simply presented our raw data it would be hard to visualize what the data was showing, especially if there was a lot of it.Descriptive statistics therefore enables us to present the data in a more meaningful way, which allows simpler interpretation of the data.

Typically, there are two general types of statistic that are used to describe data:

1. Measures of central tendency: these are ways of describing the central position of a frequency distribution for a group of data.The most common measures of central tendency are the arithmetic mean, the median, and the mode.
2. Measures of spread: these are ways of summarizing a group of data by describing how spread out the scores are. Measures of spread include the range, quartiles and the interquartile range, variance and standard deviation.

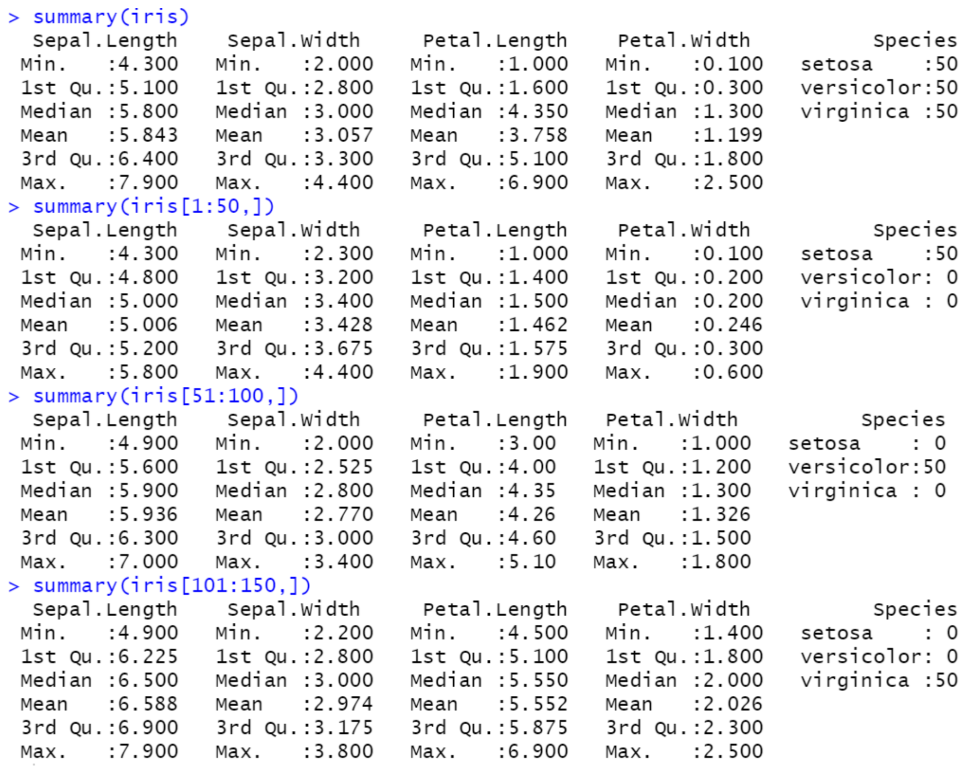
The descriptive statistic measures can be obtained by using the functions like mean(), median(), range(), sd(), quintile(), etc.

Ex:

```
> V

[1] 1 2 3 4

>mean(V)

[1] 2.5

>mean(iris$Sepal.Length)

[1] 5.843333

>sd(iris$Sepal.Width)

[1] 0.4358663

>quantile(iris$Petal.Width)

  0%  25%  50%  75% 100%

0.1  0.3  1.3  1.8  2.5
```

R comes with an inbuilt function summary(), that calculates basic summary statistics for all numerical features in a data.frame. But generating and observing the summery statistics of features with respect to different classes will allow us to find out which features are more important.Fo example, if we want to create a classification model, Petal.Length has more clear boundary between the three classes.

Ex:



```
> 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                  
> summary(iris[1:50,])
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.300   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400   1st Qu.:0.200   versicolor: 0  
 Median :5.000   Median :3.400   Median :1.500   Median :0.200   virginica : 0  
 Mean   :5.006   Mean   :3.428   Mean   :1.462   Mean   :0.246                  
 3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575   3rd Qu.:0.300                  
 Max.   :5.800   Max.   :4.400   Max.   :1.900   Max.   :0.600                  
> summary(iris[51:100,])
  Sepal.Length    Sepal.Width     Petal.Length   Petal.Width          Species  
 Min.   :4.900   Min.   :2.000   Min.   :3.00   Min.   :1.000   setosa    : 0  
 1st Qu.:5.600   1st Qu.:2.525   1st Qu.:4.00   1st Qu.:1.200   versicolor:50  
 Median :5.900   Median :2.800   Median :4.35   Median :1.300   virginica : 0  
 Mean   :5.936   Mean   :2.770   Mean   :4.26   Mean   :1.326                  
 3rd Qu.:6.300   3rd Qu.:3.000   3rd Qu.:4.60   3rd Qu.:1.500                  
 Max.   :7.000   Max.   :3.400   Max.   :5.10   Max.   :1.800                  
> summary(iris[101:150,])
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.900   Min.   :2.200   Min.   :4.500   Min.   :1.400   setosa    : 0  
 1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100   1st Qu.:1.800   versicolor: 0  
 Median :6.500   Median :3.000   Median :5.550   Median :2.000   virginica :50  
 Mean   :6.588   Mean   :2.974   Mean   :5.552   Mean   :2.026                  
 3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875   3rd Qu.:2.300                  
 Max.   :7.900   Max.   :3.800   Max.   :6.900   Max.   :2.500                  
```

Exp. 11:Demonstrate the following statistical distribution functions in R:

a. Normal Distribution

b. Binomial Distribution

c. Poisson Distribution

d. Chi-Square Distribution

Aim: To understand and use various statistical distributions of data

A probability distribution describes how the values of a random variable are distributed. Forexample, the collection of all possible outcomes of a sequence of coin tossing is known to follow the **Binomial distribution.** The means of sufficiently large samples of a data population are known to resemble the **Normal distribution.**These distributions can be used to make statistical inferences on the entire data population as a whole.

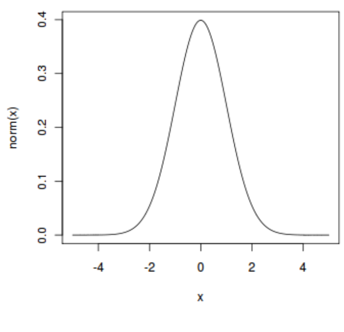
R has functions available for most of the famous statistical distributions.

- Prefix the name as follows:
  - With d for the density or probability mass function (pmf)
  - With p for the cumulative distribution function (cdf)
  - With q for quantiles
  - With r for random number generation

Distribution	Density/pmf	cdf	Quantiles	Random Numbers
Normal	dnorm()	pnorm()	qnorm()	rnorm()
Binomial	dbinom()	pbinom()	qbinom()	rbinom()
Poisson	dpois()	ppois()	qpois()	rpois()
Chi-square	dchisq()	pchisq()	qchisq()	rchisq()

a. Normal Distribution

Normal Distribution is a probability function used in statistics that tells about how the data values are distributed. It is the most important probability distribution function used in statistics because of its advantages in real case scenarios. For example, the height of the population, shoe size, IQ level, rolling a dice, and many more.It is generally observed that data distribution is normal when there is a random collection of data from independent sources. The graph produced after plotting the value of the variable on x-axis and count of the value on y-axis is a bell-shaped curve graph.



The graph signifies that the peak point is the mean of the data set and half of the values of data set lie on the left side of the mean and other half lies on the right part of the mean, telling about the distribution of the values.The graph is symmetric distribution.

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

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

– x,q represents the vector of quantiles

– p is vector of probabilities

– n is number of observations

– mean(x) represents the mean of data set ***x****. (Default value:0)*

– sd represents the standard deviation of data set ***x*** *(Default value:1)*

$$= \sqrt{\frac{\sum_{i=1}^{n}(x_i - mean)^2}{n}}$$

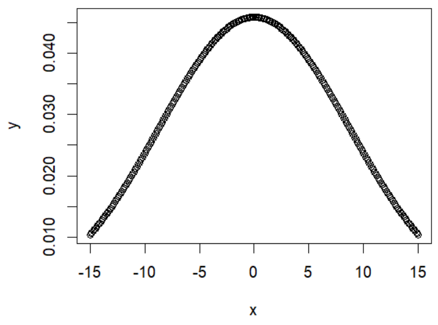
dnorm(x, mean, sd)

dnorm() function in R programming measures density function of distribution. In statistics, it is measured by below formula-

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

where, **μ** is mean and **σ** is standard deviation.

```
> x = seq(-15, 15, by=0.1)

> y = dnorm(x, mean(x), sd(x))

>plot(x,y)
```

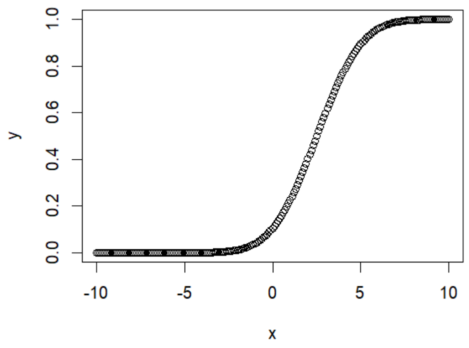


pnorm(q, mean, sd)

pnorm() function is the cumulative distribution function which measures the probability that a random number takes a value less than or equal to x, i.e., in statistics it is given by-

$$F_X(x) = Pr[X \leq x] = \alpha$$

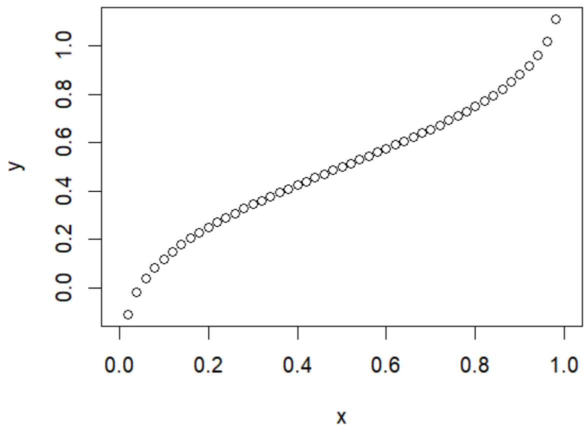
```
> x <- seq(-10, 10, by=0.1)

> y <- pnorm(x, mean = 2.5, sd = 2)

>plot(x,y)
```



qnorm(p, mean, sd)

This function takes the probability value and gives a number whose cumulative value matches the probability value.It is useful in finding the percentiles of a normal distribution

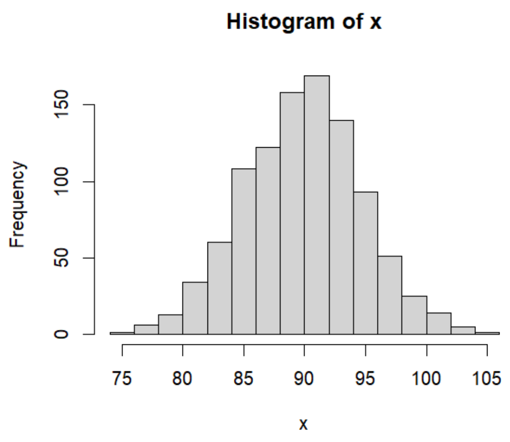
```
> x <- seq(0, 1, by = 0.02)

> y <- qnorm(x, mean(x), sd(x))

>plot(x, y)
```



rnorm(n, mean, sd)

rnorm() function in R programming is used to generate a vector of random numbers which are normally distributed

```
# Create a vector of 1000 random numbers with mean=90 and sd=5

> x <- rnorm(1000, mean=90, sd=5)

>hist(x)
```



b. **Binomial distribution in R**

The binomial distribution is a discrete distribution and has only two outcomes i.e. success or failure.The previous outcome does not affect the next outcome.It is used in many real-life scenarios such as in determining whether a particular lottery ticket has won or not, whether a drug is able to cure a person or not, for analyzing the outcome of a dice, etc.In statistics, it is measured by below formula:

$P(X = k) = nC_r p^r q^{n-r}$, where r = 0, 1, 2, 3,.., n

p is the probability of success

q is the probability of failure

p + q = 1

We have four functions for handling binomial distribution in R:

- dbinom(x, size, prob)
- pbinom(q, size, prob)
- qbinom(p, size, prob)
- rbinom(n, size, prob)

– x, q represents the vector of quantiles

– p is vector of probabilities

– n is number of observations

– size is number of trials (zero or more)

– prob is probability of success on each trial

dbinom(x, size, prob) -This function is used to find probability at a particular value for a data that follows binomial distribution i.e. it finds:

P(X = x)

Eg.1.

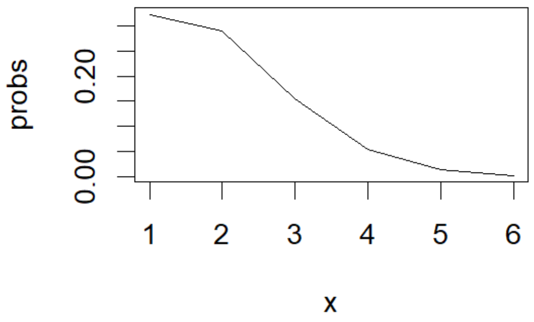
# Probability of getting 3 when a dice is thrown 13 times

>a = dbinom(3, size = 13, prob = 1 / 6)

```
>print(a)

[1] 0.2138454
```

Eg.2.

```
> x=c(1:6)

>probs<- dbinom(x, 10, 1 / 6)

>print(probs)

[1] 0.323011166 0.290710049 0.155045360 0.054265876 0.013023810 0.002170635

>plot(x, probs, type = "l")
```



pbinom(q, size, prob) -It is used to find the cumulative probability of a data following binomial distribution till a given value i.e. it findsP(X <= q)

```
# Probability of getting 26 or less heads from 51 tosses of a coin

> x <- pbinom(26,51,0.5)

>print(x)

[1] 0.610116
```

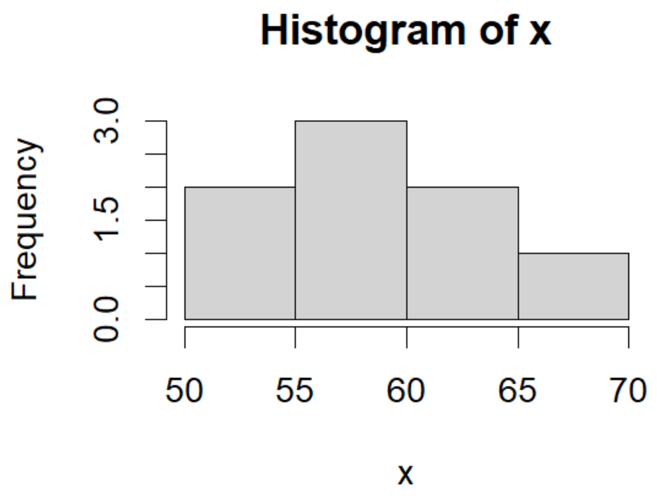
qbinom(p, size, prob) -This function takes the probability value and gives a number whose cumulative value matches the probability value

```
# How many heads will have a probability of 0.25 when a coin is tossed 51 times

> x <- qbinom(0.25,51,1/2)

>print(x)

[1] 23
```

rbinom(n, size, prob)-This function generates required number of random values of given probability from a given sample

```
# Find 8 random values from a sample of 150 with probability of 0.4

> x <- rbinom(8,150,0.4)
```

```
>print(x)

[1] 58 52 50 64 61 38 60 52

>hist(x)
```



c. Poisson distribution in R

Poisson distribution: It is the probability distribution of independent event occurrences in an interval. If λ is the mean occurrence per interval, then the probability of having x occurrences within a given interval is:

f(x)=$\frac{\lambda^{x} e-\lambda}{x!}$where x=0,1,2,3,…

There are four Poisson distribution functions available in R:

- dpois(x, lambda)
- ppois(q, lambda)
- qpois(p, lambda)
- rpois(n, lambda)

dpois(x,lambda)– It calculates the probability of a random variable that is available within a certain range.

- If there are five cars crossing a bridge per minute on average, find the probability of having 3 cars crossing the bridge in a particular minute

```
> y=dpois(3, lambda = 5)

> print(y)

[1] 0.1403739
```

ppois(q, lambda) –It calculates the probability of a random variable that will be equal to or less than a number

- If there are twelve cars crossing a bridge per minute on average, find the probability of having upto 16 cars crossing the bridge in a particular minute

```
> c=ppois(16,lambda=12)

> print(c)

  [1] 0.89870
```

qpois(p, lambda)- It allows obtaining the corresponding Poisson quantiles for a set of probabilities

- The quantile 0.75 of a Poisson distribution for *λ =10*:

  ```
  > h=qpois(0.75, lambda = 10)

  >print(h)

  [1] 12
  ```

rpois(n,lambda)-To draw n observations from a Poisson distribution

- To obtain 10 random observations from a Poisson distribution with mean = 4

  ```
  > k=rpois(10, lambda = 4)

  >print(k)

  [1] 2 2 5 3 2 2 4 5 3 6
  ```

d. Chi-Square distribution

It is the distribution of the sum of squared standard normal deviates. A standard normal deviate is a random sample from the standard normal distribution.The degrees of freedom of the distribution is equal to the number of standard normal deviates being summed. The Chi Square distribution is very important because many test statistics are approximately distributed as Chi Square

Formula : $V=X_1^2 + X_2^2 + \dots + X_m^2 \sim \chi_{(m)}^2$ where m is degrees of freedom

- dchisq(x, df)
- pchisq(q, df)
- qchisq(p, df)
- rchisq(n, df)

where

x, q -vector of quantiles

p -vector of probabilities

n -number of observations

df-degrees of freedom (non-negative, but can be non-integer).

dchisq(x, df)

```
>df = 6

> a=dchisq(4,df)

>print(a)

[1] 0.1353353

>vec<- 1:4
```

```
> a=dchisq(vec,df)

>print(a)

[1] 0.03790817 0.09196986 0.12551072 0.13533528

>df = 5

>k=pchisq(5, df)              # Prob. for <=5

>print(k)

[1] 0.5841198
```

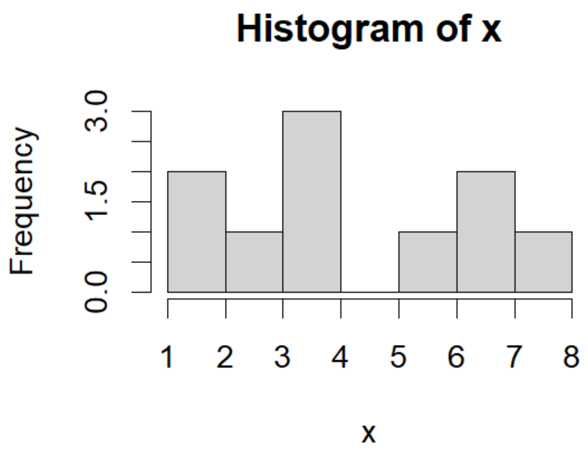
qchisq(p, df)

```
> qchisq(0.95, df=7)        #7 degrees of freedom
[1] 14.067
```

The 95th percentile of the Chi-Squared distribution with 7 degrees of freedom is 14.067

rchisq(n, df)

```
> x=rchisq(10,df=5)

>print(x)

[1] 4.029649 3.493405 3.376066 5.223755 5.727549 5.730031 3.331059 5.122672

 [9] 4.035966 4.965683

>hist(x)
```



Exp. 12:Illustrate the following basic graphics in R:

- **a. Bar plots**
- **b. Pie Charts**
- **c. Histograms**
- **d. Kernel density plots**
- **e. Boxplots**
- **f. Dotplots**

Aim: To understand and use basic graphics in R

a: Bar plots in R

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.

Syntax:

barplot(height,xlab,ylab,main,names.arg,col)

where

heightis a vector or matrix containing numeric values

xlab is the label for x axis

ylab is the label for y axis

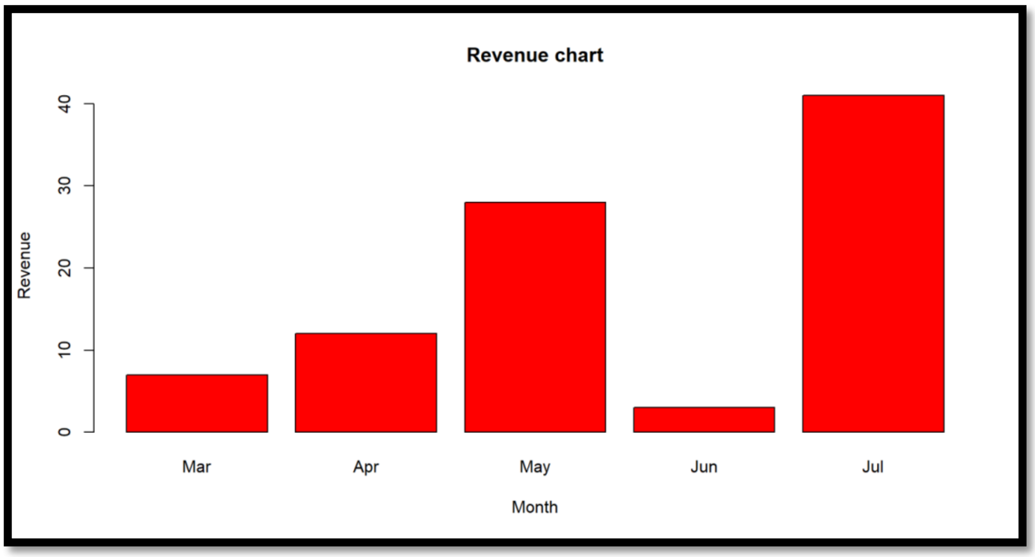
main is the title of the bar chart

names.arg is a vector of names appearing under each bar

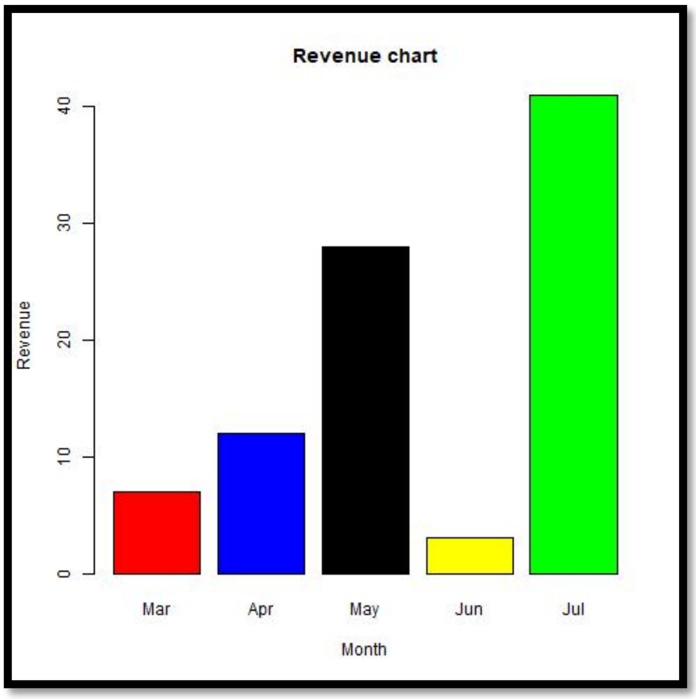
col is used to give colors to the bars in the graph

Example:

```
>H <- c(7,12,28,3,41)
>M <- c("Mar","Apr","May","Jun","Jul")
>barplot(H, names.arg=M, xlab="Month",ylab="Revenue", col="red", main="Revenue chart")
```

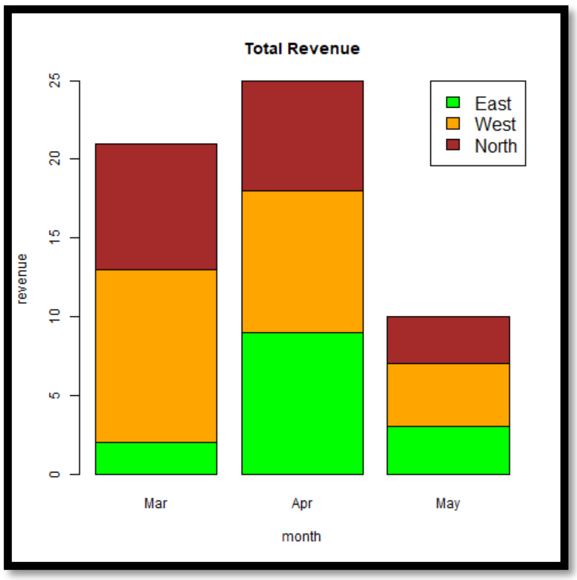


```
>H <- c(7,12,28,3,41)
>M <- c("Mar","Apr","May","Jun","Jul")
>C <-c("red","blue","black","yellow","green")
>jpeg(file="monthly_revenue.jpeg")                                   # pdf / png
>barplot(H, names.arg=M, xlab="Month",ylab="Revenue", col=C, main="Revenue chart")
>dev.off()                              #jpeg image is created in the current working directory
```

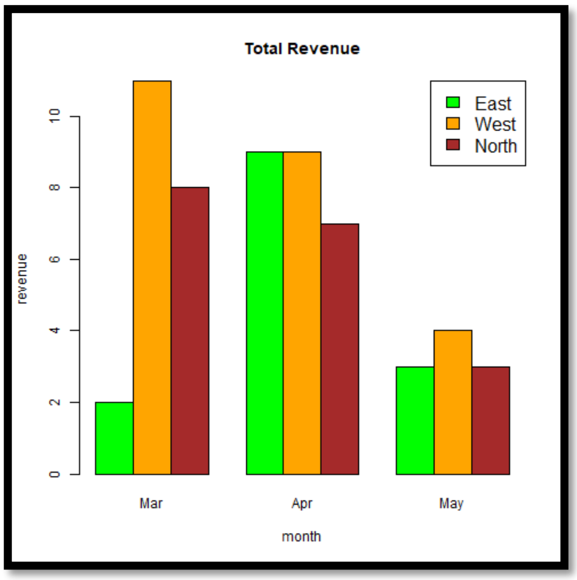


Stacked Barplot / Grouped Barplot

```
colors = c("green","red","blue")

months = c("Mar","Apr","May")

regions = c("East","West","North")

values = matrix(c(2,9,3,11,9,4,8,7,3), nrow=3, byrow=TRUE)

png(file ="barchart_stack.png")

barplot(values, main="Total Revenue",names.arg=months,

xlab="month",ylab="revenue",col=colors)              #beside=TRUE for grouped barplot

legend("topright", regions, cex = 1.3, fill=colors)

dev.off()
```



Stacked Barplot



Grouped Barplot

b: Pie charts in R

A pie-chart is a representation of values as slices of a circle with different colours

Syntax:

pie(x, labels, radius, main, col, clockwise)

where

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.(−1 to +1).

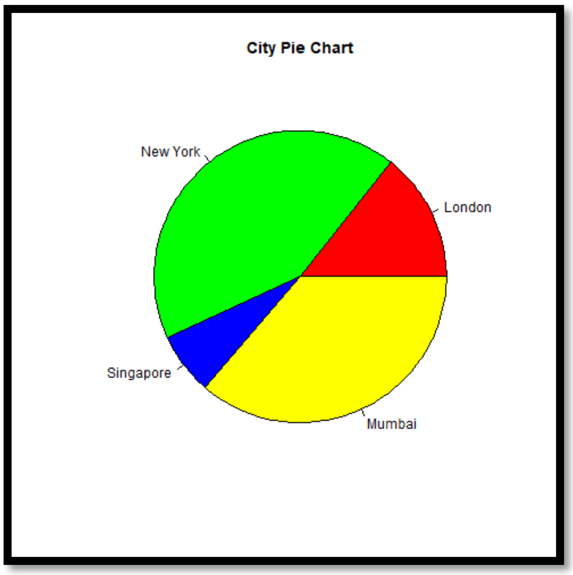
main indicates the title of the chart

col indicates the color palette

clockwise is a logical value indicating if the slices are drawnclockwise or anticlockwise.(default:FALSE)

Example:

```
> x <- c(21, 62, 10, 53)

>labels<- c("London", "New York", "Singapore", "Mumbai")

>colors=c("red", "green", "blue", "yellow")

>png(file = "city.png")

>pie(x, labels, main = "City Pie Chart", col=colors)

>dev.off()
```



c: Histograms in R

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chart but the difference is it groups the values into continuous ranges.

Syntax:

hist(v, main, xlab, xlim, ylim, breaks, col)

where

v is a vector containing numeric values used in histogram

main indicates title of the chart

xlab is used to give description of x-axis

xlim is used to specify the range of values on the x-axis

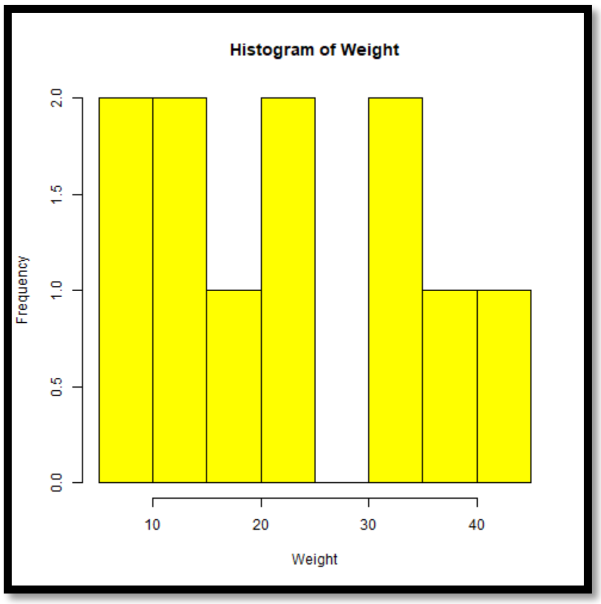
ylim is used to specify the range of values on the y-axis

breaks is used to mention the width of each bar

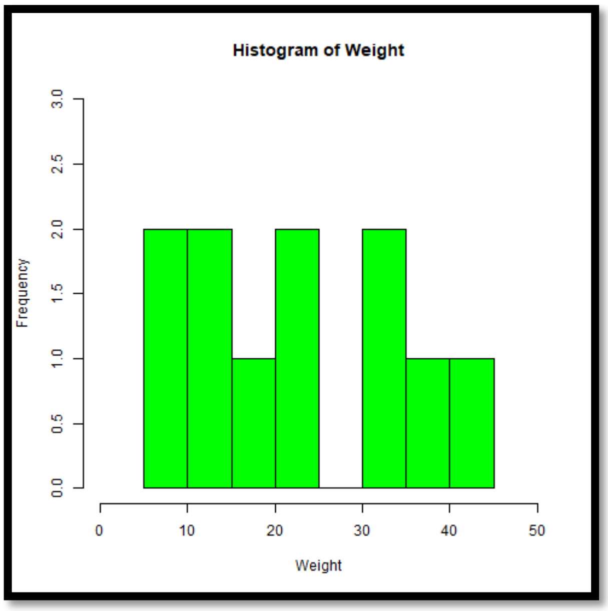
col is used to set color of the bars

Example:

```
> Weight <-  c(9,13,21,8,36,22,12,41,31,33,19)

>png(file = "histogram.png")

>hist(Weight, xlab = "Weight", col = "yellow")

>dev.off()
```



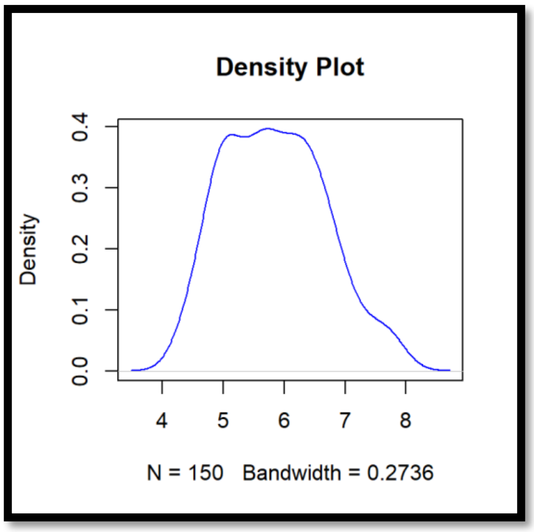
```
>Weight <- c(9,13,21,8,36,22,12,41,31,33,19)

>png(file = "hist_lims.png")

>hist(Weight,xlab="Weight",col="green",xlim =c(0,50),ylim=c(0,3),breaks=5)

>dev.off()
```



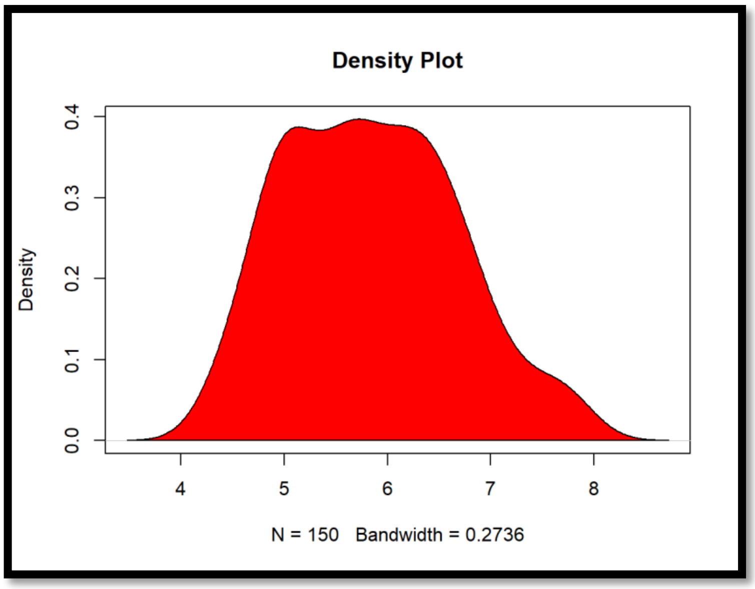
d: Kernel Density plots in R

Kerneldensity plot is a representation of the distribution of a numeric variable that uses a kernel density estimate to show the probability density function of the variable. The density() function is used to compute kernel density estimates

```
> d=density(iris$Sepal.Length)

>plot(d, main="Density Plot",col="blue")
```



```
# Filled density plot
> d=density(iris$Sepal.Length)
>plot(d, main="Density Plot")
>polygon(d, col="red")
```



e: Boxplots in R

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

Syntax:

boxplot(x, data, notch, names, main)

where

x is a vector or a single list

data is the data frame

notch is a logical value, Set as TRUE to draw a notch.

names are the group labels to be printed under each boxplot

main is used to give a title to the graph

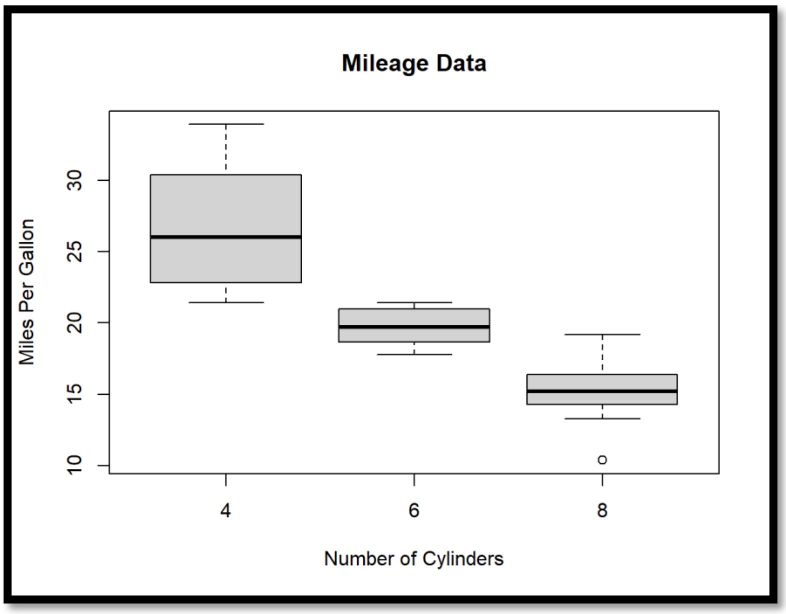
Example:

#png(file = "box_plot.png")

#print(summary(mtcars[1:2]))

>boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders", ylab = "Miles Per Gallon", main="Mileage Data")   #notch=TRUE

#dev.off()



f: Dotplotsin R

Renamed as Dotcharts, it draws a Cleveland dot plot. They are an alternative to bar charts or pie charts.They look somewhat like a horizontal bar chart where the bars are replaced by a dots at the values associated with each category

Syntax:

dotchart(x, labels, cex, main, xlab)

where**x** is either a vector or matrix of numeric values

labelsis a vector of labels for each point

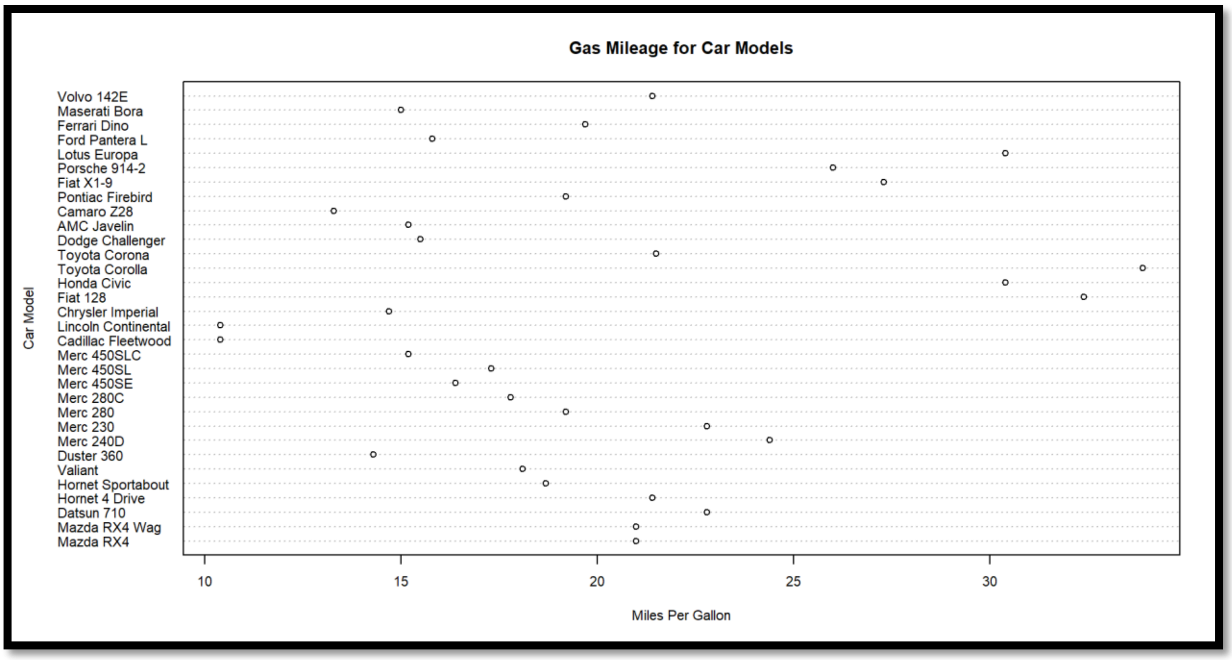
cexis the character size to be used

main is used to give a title to the graph

xlab ,ylab –x and y axis labels

Example:

>dotchart(mtcars$mpg,labels=row.names(mtcars),cex=0.7,main="Gas Mileage for Car Models",xlab="Miles Per Gallon", ylab="Car Model")



Exp. 13:Illustrate the Correlation and Covariance analysis using R

Aim: To analyze data using Correlation and Covariance measures in R

When dealing with more than one variable, we need to test their relationships with each other.Two simple measures are:

- Correlation
- Covariance

Correlation using R

Correlation is used to evaluate the association between two or more variables.It is computed as:

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

r_{xy} =0 => No Correlation

$r_{xy} > 0$ =>Positive Correlation

$r_{xy} < 0$ =>Negative Correlation

x and y are two vectors of length n

$\bar{x}$ and $\bar{y}$ corresponds to the means of x and y, respectively

Syntax:

cor(x, y , method)

where

x is a numeric vector, matrix or data frame

y is NULL (default) or a vector, matrix or data frame with compatible dimensions to x

method ="pearson" or "kendall" or "spearman" #Default:pearson

- Correlation range is [-1,+1]

Example:

```
>pcor=cor(airquality$Wind,airquality$Temp)
>print(pcor)
[1] -0.4579879                              #Negative Correlation

>corw=cor(women$height,women$weight)
>print(corw)
[1] 0.9954948                                 # Positive Correlation

>icor=cor(iris[1:4])
>print(icor)                         #Correlation Matrix


             Sepal.LengthSepal.WidthPetal.LengthPetal.Width
Sepal.Length    1.0000000   -0.1175698   0.8717538   0.8179411
Sepal.Width    -0.1175698    1.0000000  -0.4284401  -0.3661259
Petal.Length    0.8717538   -0.4284401   1.0000000   0.9628654
Petal.Width     0.8179411   -0.3661259   0.9628654   1.0000000
```

Covariance using R

Covariance of two variables x and y in a data set measures how the two are linearly related.A positive covariance would indicate a positive linear relationship between the variables, and a negative covariance would indicate the opposite.Covariance range is [-∞,+∞]. Covariance of two variables *x* and *y* in a data set is computed as:

Population Covariance Formula

$$Cov(x,y) = \frac{\Sigma(x_i-\bar{x})(y_i-\bar{y})}{N}$$

Sample Covariance

$$Cov(x,y) = \frac{\Sigma(x_i-\bar{x})(y_i-y)}{N-1}$$

Syntax:

cov(x, y , method)

where

x is a numeric vector, matrix or data frame

y is NULL (default) or a vector, matrix or data frame with compatible dimensions to x

method ="pearson" or "kendall" or "spearman"   #Default:pearson

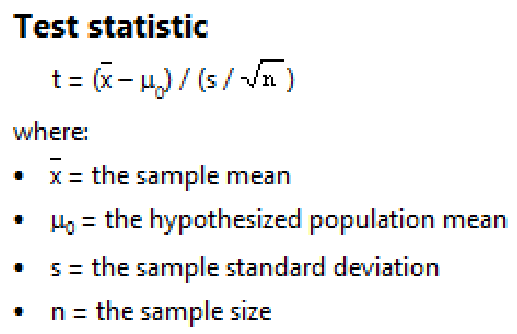
```
>acov=cov(airquality$Wind,airquality$Temp)

>print(acov)

[1] -15.27214

>wcov=cov(women$height,women$weight)

>print(wcov)

[1] 69

>icov=cov(iris[1:4])

>print(icov)                              #Covariance Matrix

              Sepal.LengthSepal.WidthPetal.LengthPetal.Width
Sepal.Length   0.6856935  -0.0424340   1.2743154   0.5162707
Sepal.Width   -0.0424340   0.1899794  -0.3296564  -0.1216394
Petal.Length   1.2743154  -0.3296564   3.1162779   1.2956094
Petal.Width    0.5162707  -0.1216394   1.2956094   0.5810063


#Diagonal values are variance of each attribute
```

Exp.14: Illustrate the different types of t-tests using R

t-test:

It is called Student's t-test. One of the most common tests in statistics is the t-test, used to determine whether the means of two groups are equal to each other.Performs one and two sample t-tests on vectors of data.

t.test() can be used to perform t-tests in R t-statistic is calculated as:



Test statistic

$t = (\bar{x} - \mu_0) / (s / \sqrt{n})$

where:

- $\bar{x}$ = the sample mean
- μ_0 = the hypothesized population mean
- s = the sample standard deviation
- n = the sample size

The assumption for the test is that both groups are sampled from normal distributions with equal variances. The null hypothesis is that the two means are equal, and the alternative is that they are not.
Syntax:
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95,…)

where
- x is a numeric vector of data values
- y is an optional numeric vector of data values
- If y is excluded, the function performs a one-sample t-test on the data contained in x, if it is includedit performs a two-sample t-tests using both x and y
- mu provides a number indicating the true value of the mean (or difference in means if performing a two sample test) under the null hypothesis
- alternative is a character string specifying the alternative hypothesis:
  - "two.sided" (which is the default)
  - "greater" or "less" depending on whether the alternative hypothesis is that the mean is different than, greater than or less than mu, respectively
- paired - a logical value indicating whether a paired t-test is to be done
- var.equal - a logical variable indicating whether to treat the two variances as being equal.
- conf.level- confidence level of the interval

One-Sample t-test:

```
>data(CO2)

>t1= t.test(CO2$uptake)                              #Null Hypothesis, mean=0
```

or

```
> t1=t.test(CO2$uptake, alternative="t")        #two.sided=TRUE, mean=0

>print(t1)
```

```
        One Sample t-test
data:  CO2$uptake
t = 23.063, df = 83, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 24.86622 29.55997
sample estimates:
```

```
mean of x
  27.2131

> t2=t.test(CO2$uptake,alternative="g", mu=10)        # mean <= 10

>print(t2)

            One Sample t-test
data:  CO2$uptake
t = 14.588, df = 83, p-value < 2.2e-16
alternative hypothesis: true mean is greater than 10
95 percent confidence interval:
25.25034      Inf
sample estimates:
mean of x
  27.2131

> t3=t.test(CO2$uptake,alternative="l", mu=10)         # mean >= 10

>print(t3)

One Sample t-test
data:  CO2$uptake
t = 14.588, df = 83, p-value = 1
alternative hypothesis: true mean is less than 10
95 percent confidence interval:
-Inf 29.17585
sample estimates:
mean of x
  27.2131
```

Two-Sample t-test

```
>data(CO2)

>t4=t.test(CO2$uptake,CO2$conc)             # Welch Two Sample t-test,True mean =0

>print(t4)

Welch Two Sample t-test
data:  CO2$uptake and CO2$conc
t = -12.621, df = 83.222, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -472.0467 -343.5271
sample estimates:
mean of x mean of y
27.2131  435.0000

>t5=t.test(CO2$uptake,CO2$conc,alternative="l", mu=12)             #True mean >= 12

>print(t5)

Welch Two Sample t-test
```

```
data:  CO2$uptake and CO2$conc
t = -12.993, df = 83.222, p-value < 2.2e-16
alternative hypothesis: true difference in means is less than 12
95 percent confidence interval:
-Inf -354.0442
sample estimates:
mean of x mean of y
27.2131  435.0000
```

Note: true mean =mean(x) –mean(y)

Paired t-tests

For testing paired data (for example, measurement on twins, before and after treatment effects, father and son comparison), we cannot use two-sample t-tests since the independence assumption is not valid. Instead we need to use a paired t-test. This can be done using the option paired =TRUE.

Example:

```
>fheight=c(165,144,178,189,123)

>sheight=c(167,141,191,200,120)

>ht=t.test(fheight,sheight,paired=TRUE)     # True Mean=0

>print(ht)

        Paired t-test
data:  fheight and sheight
t = -1.1744, df = 4, p-value = 0.3054
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -13.456231   5.456231
sample estimates:
mean of the differences
                  -4

>ht1=t.test(fheight, sheight, alternative="g", paired=TRUE)      # True Mean<=0

>print(ht1)

        Paired t-test
data:  fheight and sheight
t = -1.1744, df = 4, p-value = 0.8473
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
-11.26081       Inf
sample estimates:
mean of the differences
                  -4
```

Exp. 15 :Illustrate the ANOVA test using R

Aim: To analyze data using ANOVA test in R

ANOVA (ANalysis Of VAriance) test : An ANOVA test is a way to find out if survey or experiment results are significant. It is used for testing of various groups to see if there is a difference between them

Examples:

- A group of psychiatric patients are trying three different therapies: counseling, medication and biofeedback. You want to see if one therapy is better than the others.
- A manufacturer has two different processes to make light bulbs. They want to know if one process is better than the other.
- Students from different colleges take the same exam. You want to see if one college outperforms the other

In R, ANOVA test is done using aov()

Syntax:

aov(formula, data = NULL, ...)

where

formula- formula specifying the model

data - data frame in which the variables specified in the formula will be found

Example:

```
>require(reshape2)

>data("tips", package="reshape2")

>tipANOVA=aov(tip ~ day-1, tips)             #-1 to exclude intercept

>print(tipANOVA)

Call:
aov(formula = tip ~ day, data = tips)
Terms:
          day Residuals
Sum of Squares    9.5259  455.6866
Deg. of Freedom        3       240
Residual standard error: 1.377931
Estimated effects may be unbalanced


>print(tipANOVA$coefficients)
```

(Intercept)   daySatdaySundayThur

2.73473684 0.25836661 0.52039474 0.03671477

Alternative way : To use regression

```
>tiplm=lm(tip ~ day-1, tips)                  #No intercept term

>print(tiplm)

Call:

lm(formula = tip ~ day - 1, data = tips)

Coefficients:

dayFridaySatdaySundayThur

  2.735    2.993    3.255    2.771
```