GODAVARI INSTITUTE OF ENGINEERING AND TECHNOLOGY

Department of _____

Name:_____ Pin No: | | | | | | | | | | |

CERTIFICATE

Certified that this is the bonafide record of practical work done by

Mr./Ms._____

a Student of_____with Pin No:_____

in the_____Laboratory during the Academic year_____

No. of Experiments Conducted: | | | No. of Experiments attended: | | |

Faculty In-charge Head of the Department

Submitted for Practical Examination Conducted on_____

Examiner – 1 Examiner – 2

INDEX

Exp No	Date	Name of the Experiment	Page No.	Signature
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				

EXPERIMENT-1

AIM: Introduction to R

a) **Installing R console and R studio and showing differences for both.**
b) **Importing and loading packages in R console and R studio.**
c) **Built in Mathematical operations**

The process of installing r depends on type of operating system. R can be installed in following way:

Step 1: Go to website can R project window .
Step 2: Click on the download R.4.1.3 for windows.
Step 3: Click on the tab will download the r installer double click on the Installer to launch it.
Step 4: Select the language of your choice and click ok.
Step 5: clicking on next will leave you to important information of license click on next
Step 6: Then it will provide you with an interface for selecting the Destination for your R installation.
Step 7: After clicking on next you will be directed to select components for installation. It is ideal to keep all components.
Step 8: In this next step you will be ask to select the start menu folder. By default, start menu can't be created just you can change that By uncheck in the checkbox.
Step 9: In the next step you will be asked for additional details, you can Create a desktop shortcut by check in the boxes.
Step 10: Clicking on next will prompt the installation to start, after this the installation is complete and R is installed.

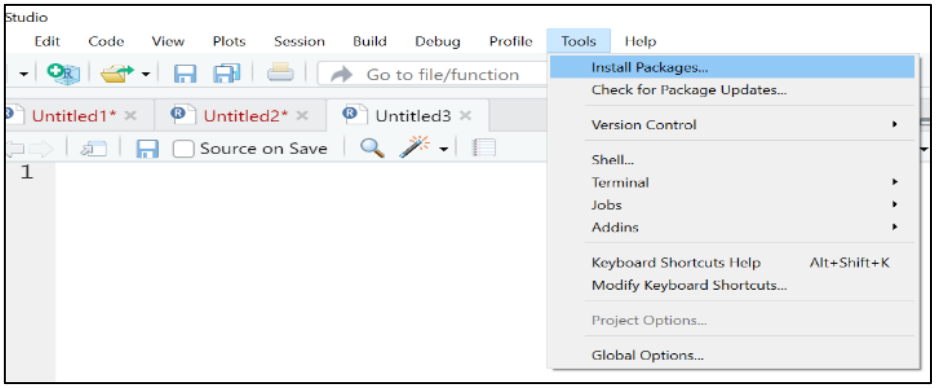
DIFFERENCES BETWEEN R CONSOLE & R STUDIO:

R Console	R Studio
1.Enter is used to directly run the program.	1.Run is used to run the program
2.Print function is not necessary.	2.Print function is necessary.
3.The output is displayed in next line.	3.The output is displayed in console window.
4.R may be used without r studio.	4.R studio may not be used without r console.

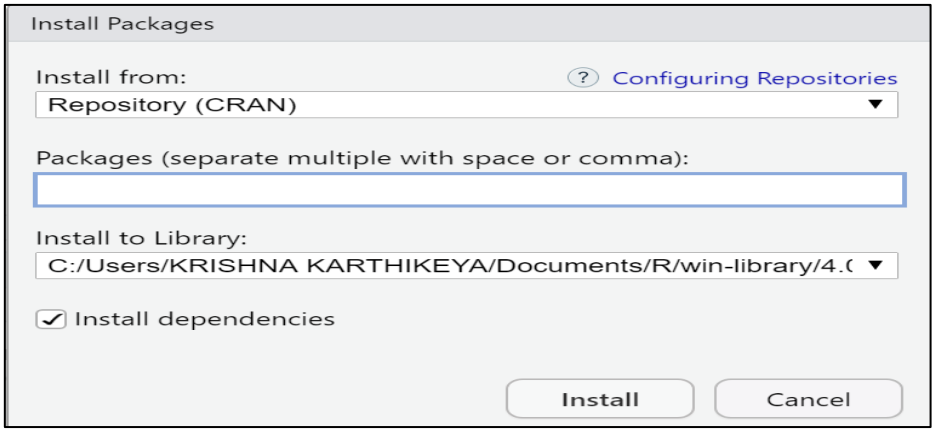
INSTALLATION AND LOADING PACKAGE SIN R CONSOLE AND R STUDIO:

Step 1: Open R Studio

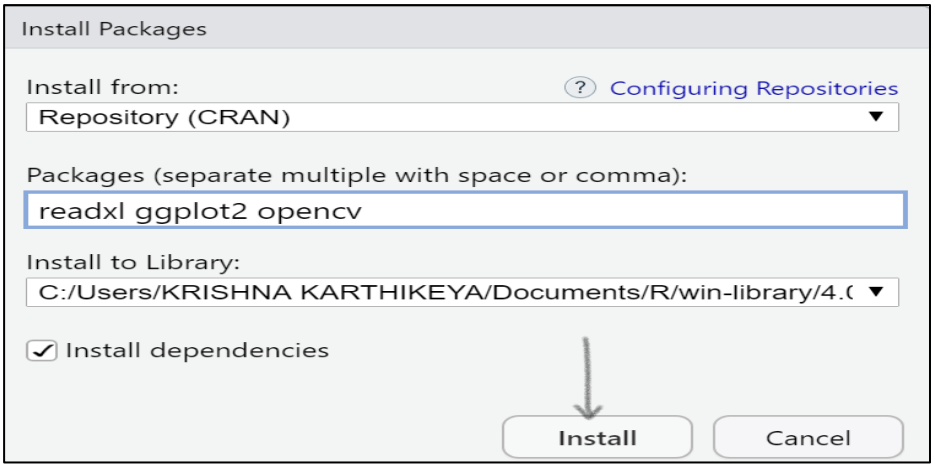
Step 2: Navigate to Tools and select the Install packages option



Step 3: In the Text box packages, enter the package names



Step 4: Click on install button



Step 5: Find the console after clicking on the install button. If all packages installed successfully the below output will be generated. Otherwise, follow the steps again.

```
package 'readxl' successfully unpacked and MD5 sums checked
package 'ggplot2' successfully unpacked and MD5 sums checked
package 'opencv' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\KRISHNA KARTHIKEYA\AppData\Local\Temp\Rtmpwj
loaded_packages
>
```

BUILT IN MATHEMATICAL OPERATIONS:

s.no	Function	Description	Example
1	abs(x)	it returns the absolute value of input x.	x<- -4 print(abs(x)) **Output** [1] 4
2	sqrt(x)	It returns the square root of input x.	x<- 4 print(sqrt(x)) **Output** [1] 2
3	ceiling(x)	It returns the smallest integer which is larger than or equal to x.	x<- 4.5 print(ceiling(x)) **Output** [1] 5
4	floor(x)	It returns the largest integer, which is smaller than or equal to x.	x<- 2.5 print(floor(x)) **Output** [1] 2
5	trunc(x)	It returns the truncate value of input x.	x<- c(1.2,2.5,8.1) print(trunc(x)) **Output** [1] 1 2 8
6	round(x, digits=n)	It returns round value of input x.	x<- -4 print(abs(x)) **Output** 4
7	cos(x), sin(x), tan(x)	It returns cos(x), sin(x) value of input x.	x<- 4 print(cos(x)) print(sin(x)) print(tan(x)) **Output** [1] -06536436 [2] -0.7568025 [3] 1.157821
8	log(x)	It returns natural logarithm of input x.	x<- 4 print(log(x)) **Output** [1] 1.386294
9	log10(x)	It returns common logarithm of input x.	x<- 4 print(log10(x)) **Output** [1] 0.60206
10	exp(x)	It returns exponent.	x<- 4 print(exp(x)) **Output** [1] 54.59815

EXPERIMENT 2

AIM: Data Types in R –Vectors and Matrices.

a) Numerical vectors: Find the Length, Mode, Max and Min, Sum of the vector, Sort the vector and locate an element in vector for the following:

a. Create numerical vectors A and B with 5 elements each
b. Create vector C with 5 elements starts with 2 and with an increment of 3.
c. Create a vector D which is equal to A+B
d. Create a vector E which is equal to A+C
e. Find the length of vector D
f. Find the mode of vector E
g. Create a vector F expression which is 2times vector D + 3 times vector E – 1
h. Find the Min & Max of vector F
i. Sort the vector F in descending order and save it in vector G.
j. Find the sum of elements in vector G.

b) Create a vector of the values e^xcos(x) at x=3 ,3.1 ,3.2 ,……6

VECTORS:

Vector is a collection of elements of same type or set of elements of same type.
→It is a homogeneous element.
→It has dynamic memory . The memory used by the vector can be increased or decreased as elements are added to the vector or removed from the vector.
→In r programming language the very basic data types of the objects called vectors which hold elements of different classes.
→vectors can be created by giving values to variable.
→vectors can be created by single element vector or by multiple element vector.

Single Element Vector:

If you assign only one element to the variable is called single element vector.

Syntax:

Variable name ← value
X ←10

Multiple Element Vector:

If you assign more than one element to a variable is called multiple element vector.
→we can create multiple element vector in three ways they are
1):operator
2)seq() function
3)c() function

Syntax:

Variable name←c(set of values)
X←c(1:10)

Accessing Vector Elements:

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 or 0 and 1 can also be used for indexing.

Example:

```
#accessing vector position using indexing
X←c("sun","mon","tue")
U←t[c(2)]
Print(u)
```

Output:
[1] mon

Vector Manipulation:
Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)
# Vector addition.
add.result <- v1+v2
print(add.result)
# Vector subtraction.
sub.result <- v1-v2
print(sub.result)
# Vector multiplication.
multi.result <- v1*v2
print(multi.result)
# Vector division.
divi.result <- v1/v2
print(divi.result)
```

output:
[1] 7 19 4 13 1 13
[1] -1 -3 4 -3 -1 9
[1] 12 88 0 40 0 22
[1] 0.7500000 0.7272727 Inf 0.6250000 0.0000000 5.5000000

PROGRAM:

```
(a)#Creating Numerical Vectors A and B with 5 elements
A<-c(1:5)
print(A)
B<-c(6:10)
print(B)
#Creating a vector C starting from 2 with an increment of 3
C<-seq(from=2,to=16,by=3)
print(C)
#Creating a Vector D Which is equal to A+B
D<-A+B
print(D)
#Creating a vector E Which is equal to A+C
E<-A+C
print(E)
#Creating a vector F which is 2D+3E-1
F<-(2*D)+(3*E)-1
print(F)
#Finding Min and Max of Vector F
print(min(F))
print(max(F))
#Sorting Vector F in desc order and storing in G
G<-sort(F,TRUE)
print(G)
```

#Finding the sum of elements in Vector G

```
print(sum(G))
```

(b)**#Create a vector of values (e^n)*x*cos(x)**

```
x<-c(3,3.1,3.2)
x<-seq(from=3,to=6,by=0.1)
y<-exp(x)*cos(x)
print(x)
```

Output:

```
> print(A)
[1] 1 2 3 4 5
> print(B)
[1]  6  7  8  9 10
> print(C)
[1]  2  5  8 11 14
> print(D)
[1]  7  9 11 13 15
> print(E)
[1]  3  7 11 15 19
> print(F)
[1] 22 38 54 70 86
> print(min(F))
[1] 22
> print(max(F))
[1] 86
> print(G)
[1] 86 70 54 38 22
> print(sum(G))
[1] 270
> print(x)
 [1] 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4
5.5 5.6 5.7 5.8 5.9 6.0
```

EXPERIMENT 3

AIM:

a. **Create a matrix A with elements starting from 1.Number of rows '3' and columns '4' and the elements should be in column.**
b. **Create a matrix 'B' and the elements should be added in row wise.**
c. **Find the element in 2nd row and 4th column of A**
d. **Find the element in 3rd row and 1st column of A**
e. **In matrix 'B', display all the elements in 1st row.**
f. **In matrix 'B', display all the elements in 2nd column**
g. **Create a matrix 'C' with four rows and three columns starting from 10 with an inc of 8.**
h. **Create a matrix 'D', with 3 rows and three columns starting from 100, with an inc of 4.**
i. **Create a matrix 'E' in which matrix 'D' is appended to below matrix 'C'.**

MATRIX:

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. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

Syntax:

The basic syntax for creating a matrix in R is −

matrix(data, nrow, ncol, byrow, dimnames)

Following is the description of the parameters used −

- **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.

Accessing Elements of a Matrix:

Elements of a matrix can be accessed by using the column and row index of the element.

Syntax:

Matrix name[row name][column name]

Matrix Computations:

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.

PROGRAM:

```
#Creating a matrix, A with elements starting from 1 and columns 4
#Elements Should be ordered in column
A<-matrix(data=c(1:12),nrow=3,ncol=4,byrow=FALSE)
print(A)
#Creating a matrix B with elements starting from 1 and columns 4
#Elements Should be ordered in row
B<-matrix(data=c(1:12),nrow=3,ncol=4,byrow=TRUE)
print(B)
#Finding the element in 2nd row and 4th column of A
print(A[2,4])
#Finding the element in 3rd row and 1st column of A
print(A[3,1])
#In Matrix B display all elements in the first row
print(B[1,])
#In Matrix B display all elements in the second column
```

```
print(B[,2])
#Creating a matrix C with four rows and three columns
#Starting with 10 with an increment of 8
C<-matrix(seq(10,105,8),4,3)
print(C)
#Creating a matrix D with 3 rows and 3 cols starting from 100
#With inc of 4
D<-matrix(seq(100,135,4),3,3)
print(D)
#Creating a matrix E with in which D is Appended with C
E<-rbind(D,C)
print(E)
```

OUTPUT:

```
> print(A)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> print(B)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> print(A[2,4])
[1] 11
> print(A[3,1])
[1] 3
> print(B[1,])
[1] 1 2 3 4
> print(B[,2])
[1]  2  6 10
> print(C)
     [,1] [,2] [,3]
[1,]   10   42   74
[2,]   18   50   82
[3,]   26   58   90
[4,]   34   66   98
> print(D)
     [,1] [,2] [,3]
[1,]  100  112  124
[2,]  104  116  128
[3,]  108  120  132
> print(E)
     [,1] [,2] [,3]
[1,]  100  112  124
[2,]  104  116  128
[3,]  108  120  132
[4,]   10   42   74
[5,]   18   50   82
[6,]   26   58   90
[7,]   34   66   98
```

EXPERIMENT 4

Aim: Data Types in R –Arrays, Lists and Data Frames.

a) Create an array with number of rows 2 and number of columns 3, number of matrices 2 Starting from 1 with an increment of 2. You can save it as Array A. Array B starting from 10, number of rows 3, number of columns 2, and number of matrices in Sequence.

b) Find the dim (A) and dim (B)

Arrays:

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

→An array is created using the **array()** function. It takes vectors as input and uses the values in the dim parameter to create an array.

Syntax:

Array_name←array(data,dim=(rows,cols,matrix),dimnames))

Where

Data=input,

Rows=number of rows

Cols=number of columns

Matrix=number of matrices to be printed

Naming Columns and Rows:

We can give names to the rows, columns and matrices in the array by using the dimnames parameter

Accessing Array Elements:

We can access the element of an array by using row,column,name of the matrix

Syntax:

Array name[row name ,column name ,name of matrix]

Operations on array:

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.

PROGRAM:

```
(a)#Create an array with no of rows 2 and no of cols 3,
#No of matrices 2 Starting from 1 with an inc of 2.
#Save it as Array A.
A<-array(data=seq(1,23,2),dim=c(2,3,2))
print(A)
#Array B starting from 10, no of rows 3, no of cols 2,
B<-array(data=seq(10,by=2,35),dim=c(3,2,2))
print(B)
(b)#Finding the Dimensions of A&B
print(dim(A))
```

```
print(dim(B))
```

OUTPUT:

```
> print(A)
, , 1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    3    7   11
, , 2
     [,1] [,2] [,3]
[1,]13  17   21
[2,]15  19   23

> print(B)
, , 1
     [,1] [,2]
[1,]   10   16
[2,]   12   18
[3,]   14   20
, , 2
     [,1] [,2]
[1,]   22   28
[2,]   24   30
[3,]   26   32

> print(dim(A))
[1] 2 3 2

> print(dim(B))
[1] 3 2 2
```

EXPERIMENT 5

AIM: Create a list containing a vector, a matrix and a list.

a) Give names to the list **b) Add element at the end of the list**
c) Remove the last element **d) Update the 3rd element** **e) Merge the two lists**

LIST:

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

CREATING A LIST:

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

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

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

```
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
[1] 119.1
```

Naming List Elements:

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

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))
# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
# Show the list.
print(list_data)
```

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"
```

$A_Inner_list[[2]]
[1] 12.3

Accessing List Elements:

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

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

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

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

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

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

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

$`1st_Quarter`

[1] "Jan" "Feb" "Mar"

$A_Inner_list

$A_Inner_list[[1]]

[1] "green"

$A_Inner_list[[2]]

[1] 12.3

	[,1]	[,2]	[,3]
[1,]	3	5	-2
[2,]	9	1	8

Manipulating List Elements:

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

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
  list("green",12.3))
# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
# Add element at the end of the list.
list_data[4] <- "New element"
print(list_data[4])
# Remove the last element.
list_data[4] <- NULL
# Print the 4th Element.
print(list_data[4])
# Update the 3rd Element.
list_data[3] <- "updated element"
print(list_data[3])
```

When we execute the above code, it produces the following result −
[[1]]
[1] "New element"
$<NA>
NULL
$`A Inner list`
[1] "updated element"

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

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

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

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

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

```
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

[[6]]
[1] "Tue"
```

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

```
# Create lists.
list1 <- list(1:5)
print(list1)
list2 <-list(10:14)
print(list2)

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

 # Now add the vectors
result <- v1+v2
print(result)
When we execute the above code, it produces the following result −

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

[[1]]
[1] 10 11 12 13 14

[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 1

PROGRAM:
#Creating a list with a vector,matrix and list
lis<-list(c(1:10),matrix((1:12),3,4,TRUE),list((10:20),matrix(1:6),2,3))
print(lis)
#Giving names for list
names(lis)<-c("VECT","MAT","LISS")
print(lis$VECT)
#Adding an element at the end of the list
lis[4]<-"NEW ELEMENT"
print(lis[4])
#Removing the last element
lis[4]=NULL
print(lis)
#Updating the 3rd element
lis[3]<-"UPDATED ELEMENT"
print(lis)
#Creating a second list
lis2<-list(c(20:30))
print(lis2)
#Merging 2 lists
merg_lis<-list(lis,lis2)
print(merg_lis)

OUTPUT:
> print(lis)
[[1]]
[1] 1 2 3 4 5 6 7 8 9 10

[[2]]
[,1] [,2] [,3] [,4]
[1,] 1 2 3 4
[2,] 5 6 7 8
[3,] 9 10 11 12

[[3]]
[[3]][[1]]
[1] 10 11 12 13 14 15 16 17 18 19 20

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

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

[[3]][[4]]
[1] 3
```

```
> print(lis$VECT)
 [1]  1  2  3  4  5  6  7  8  9 10
> print(lis[4])
[[1]]
[1] "NEW ELEMENT"
> print(lis)
$VECT
 [1]  1  2  3  4  5  6  7  8  9 10
$MAT
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

$LISS
$LISS[[1]]
 [1] 10 11 12 13 14 15 16 17 18 19 20
$LISS[[2]]
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6

$LISS[[3]]
[1] 2

$LISS[[4]]
[1] 3
> print(lis)
$VECT
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
$MAT
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

$LISS
[1] "UPDATED ELEMENT"
> print(lis2)
[[1]]
 [1] 20 21 22 23 24 25 26 27 28 29 30
> print(merg_lis)
[[1]]
[[1]]$VECT
 [1]  1  2  3  4  5  6  7  8  9 10
[[1]]$MAT
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[[1]]$LISS
[1] "UPDATED ELEMENT"
[[2]]
[[2]][[1]]
 [1] 20 21 22 23 24 25 26 27 28 29 30
```

EXPERIMENT 6

AIM: Import an EXCEL/CSV/XML file in R studio and find the following:

a) Mean b) Median c) Mode d) Variance e) Standard Deviation

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.

Getting and 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.

Getting and 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**.**

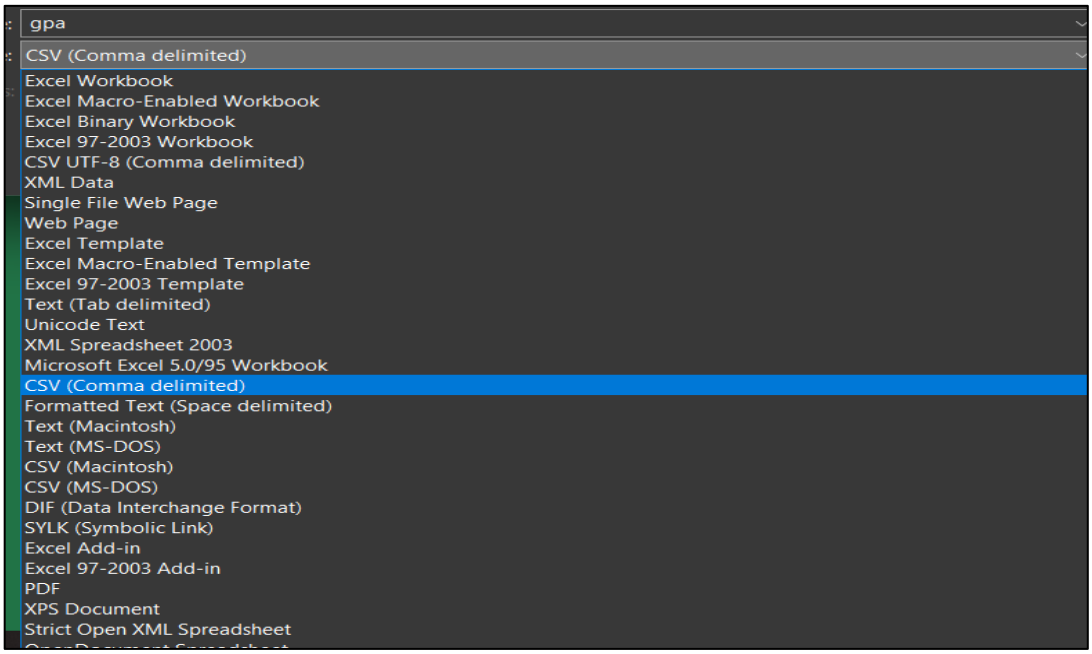
Reading a CSV File

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

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

Creating a CSV File in Excel:

Name	gpa	pin
Sampath	8.26	60
Shariff	8.19	31
Aditya	8.3	2
Mahesh	9.99	29
Nithin	7.69	16
Rohith	7.9	9



Click on Save

Program:

```
#Getting and Setting Working Directory
print(getwd())
setwd("C:/Users/sampath/Documents")
#importing csv file
data<-read.csv("gpa.csv",header=TRUE,sep=",")
print(data)
#Printing Mean of data
m<-mean(data$gpa)
print(m)
#Printing Median of GPA
md<-median(data$gpa)
print(md)
#printing max of gpa
med<-max(data$gpa)
print(med)
#MODE FUNCTION
gmode<-function()
{
  return(sort(table(data$gpa)))
}
#PRINTING MODE
print(gmode())
```

Output :

```
> print(getwd())
[1] "C:/Users/sampath/Desktop"
> data<-read.csv("gpa.csv",header=TRUE,sep=",")
> print(data)
```

	Name	gpa	pin
1	**Sampath**	**8.26**	**60**
2	**Shariff**	**8.19**	**31**
3	**Aditya**	**8.30**	**2**
4	**Mahesh**	**9.99**	**29**
5	**Nithin**	**7.69**	**16**
6	**Rohith**	**7.90**	**9**

```
> print(m)
[1] 8.388333
> print(md)
[1] 8.225
> print(med)
[1] 9.99
> print(gmode())
7.69  7.9 8.19 8.26  8.3 9.99
   1    1    1    1    1    1
```

EXPERIMENT 7

AIM: Write a R script to calculate the correlation between two variables. How to make scatter plots. Use the scatter plot to investigate the relationship between two variables.

Scatter plot:

A scatter plot is a set off dotted points to represent individual pieces of data in the horizontal and vertical axes. A graph in which values of 2 variables are plotted along x axis and y axis, the pattern of resulting points reveals a correlation between them.

→we can create a scatter plot in r programming language using plot() function.

Syntax:

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

Where,

X : this parameter sets the horizontal coordinates.
Y : this parameter sets the vertical coordinates.
Xlab : this parameter is labelled for horizontal axes.
Ylab : this parameter is labelled for vertical axes.
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 Plot.
Main: this parameter main is the little of chart.

Creating a CSV File called height.csv:

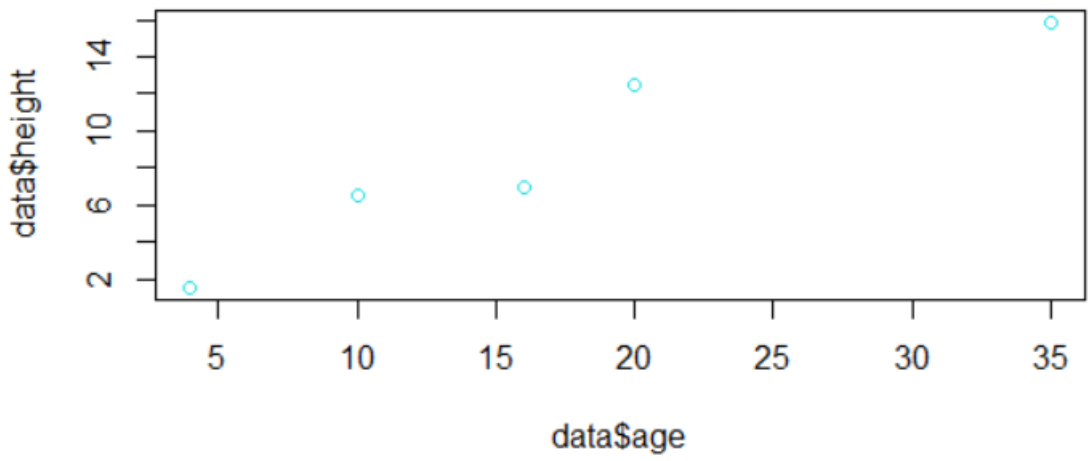
age	height
4	1.5
10	6.5
16	7
20	12.5
35	15.9

CODE:

```
#Importing Required Libraries
library("graphics")
library("base")
#Setting the working directory
setwd("C:/Users/sampath/Documents")
#importing CSV
data<-read.csv("height.csv")
print(data)
#Finding Pearson Correlation coefficient
x<-cor(data$age,data$height,method="pearson")
print(x)
#PLOTTING A SCATTERED GRAPH
with(x,plot(data$age,data$height,col=5,pch=1))
```

OUTPUT:

```
> print(data)
  Age height
1   4    1.5
2  10    6.5
3  16    7.0
4  20   12.5
5  35   15.9
> print(x)
[1] 0.9549717
> with(x,plot(data$age,data$height,col=5,pch=1))
```



EXPERIMENT 8

AIM: Import an excel/CSV file and find the following statistics

a) correlation coefficient b) Covariance

PROGRAM:

```
#Setting the working directory
setwd("C:/Users/sampa/Documents")
#IMPORTING CSV FILE
dat<-read.csv("height.csv")
#Finding Correlation Coefficient
x<-cor(dat$age,dat$height,method="pearson")
print(x)
#Finding Covariance
y<-cov(dat$age,dat$height,method="pearson")
print(y)
```

Output :

```
> print(x)
[1] 0.9549717
> print(y)
[1] 62.925
```

EXPERIMENT 9

AIM: Study and implement the functions of R-Binomial Distribution: dbinom(), pbinom(), qbinom(), rbinom().

Binomial distribution:

Binomial distribution in r programming language is a probability distribution used in statistics. The binomial distribution is a discrete distribution and it has only two outcomes i.e., success or failure. The outcomes from different trails are independent. Binomial distribution helps us to find individual probabilities as well as cumulative probabilities over a certain range.

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

→we have four functions for handling binomial distributions in r namely:

Dbinom() function:

This function is used to find probability at a particular value for a data that follows binomial distribution

Syntax:

dbinom(x, size, p)

where

x is vector of number size is total number of trails, p is probability of success.

Pbinom() function:

The function pbinom() is used to find the cumulative probability of a data following binomial distribution till a given value i.e it finds

Syntax:

Pbinom(x, size,p)

Where

X is vector of numbers,

size is total number of trails,

p is probability of success

qbinom() function:

This function is used to find the nth quatile

Syntax:

qbinom(p, size, p)

where

p is the probability,

size is the total number of trails

p is the probability of success

rbinom() function:

This function generates n random variables of a particular probability

Syntax:

rbinom(n, size, prob)

where,

n is number of observations,

size is the total number of trails,

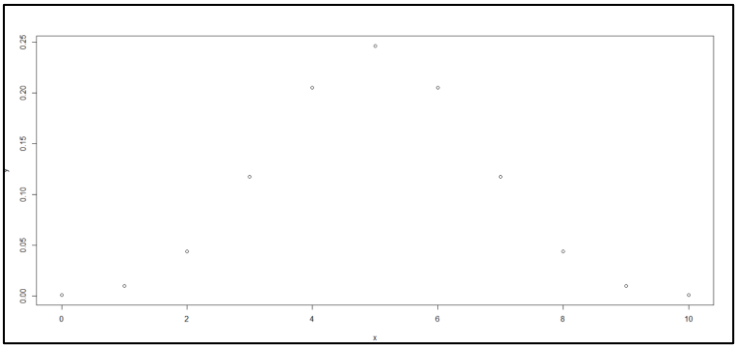
p is the probability of success

PROGRAM:

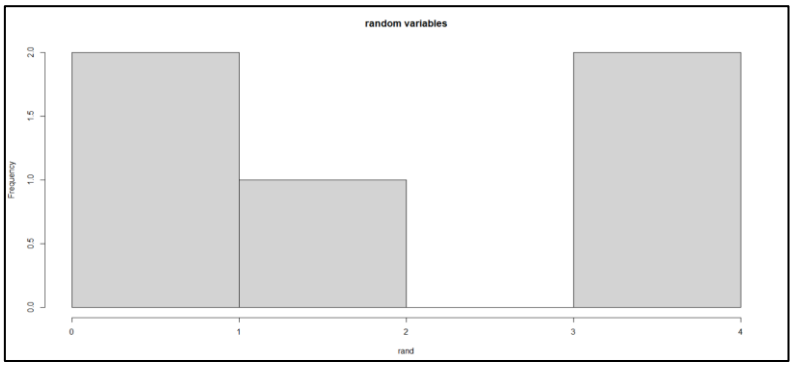
```
#Creating a sequence of numbers 10 and increment value is 0.1 and probability of success is 0.5
x<-seq(0,10,by=1)
print(x)
y<-dbinom(x,10,1/2)
print(y)
plot(x,y)
# Probability of getting 20 heads from 51 tosses of coin and probabilty of success is 1/5
x<-pbinom(20,51,1/5)
print(x)
# How many heads will have probability is 0.25 will come out from 51 tosses of coin and probability of success is 1/6
x<-qbinom(0.25,51,1/2)
print(x)
# Find 5 random values from a sample of 20 and probability of succes is 0.2
rand<-rbinom(5,20,0.2)
print(rand)
hist(rand, main="random variables")
```

OUTPUT:

```
> print(x)
 [1]  0  1  2  3  4  5  6  7  8  9 10
> print(y)
 [1] 0.0009765625 0.0097656250 0.0439453125 0.1171875000 0.2050781250 0.2460937500
0.2050781250 0.1171875000 0.0439453125 0.0097656250 0.0009765625
> plot(x,y)
```



```
> print(x)
[1] 0.999557
> print(x)
[1] 23
> print(rand)
[1] 2 3 5 4 2
> hist(rand, main="random variables")
```



EXPERIMENT 10

AIM: Study and Implement Normal Distribution in R Studio.

NORMAL DISTRIBUTION:

Normal Distribution is a probability function used in statistics that tells about how the data values are distributed.

For example, the height of the population, rolling a dice, and many more.

The graph produced after plotting the value of the variable on x-axis and count of the value on y-axis is 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.

In R, there are 4 built-in functions to generate normal distribution: dnorm(),pnorm(),qnorm(),rnom() functions

Dnorm()function:

dnorm() function in R programming measures density function of distribution.

Syntax :

dnorm(x, mean, sd)

where

x represents the vector of numbers.

mean represents the mean value.

sd represents the sd value

pnorm()function:

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

Syntax:

pnorm(x, mean, sd)

qnorm()function:

qnorm() function is the inverse of pnorm() function. It takes the probability value and gives output which corresponds to the probability value. It is useful in finding the percentiles of a normal distribution.

Syntax:

qnorm(P, mean, sd)

P represens the number probabilitity values

Rnorm()function:

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

Syntax:

rnorm(n, mean, sd)

n is the number of observations

PROGRAM:

```
#Creating a sequence of numbers -10 to 10 increment value is 1 and mean=2.5,sd=2
x<-seq(-10,10,by=0.1)
print(x)
y<-dnorm(x,mean=2.5,sd=0.8)
print(y)
plot(x,y)
#Creating a sequence of numbers -15 to 15 increment value is 0.2 and mean=1.5,sd=2
x<-seq(-15,15,by=0.2)
```

```
print(x)
y<-pnorm(x,mean=1.5,sd=2)
print(y)
plot(x,y)
#Creating a sequence of probability values 0,1 and increment value is 0.02,mean=2,sd=1
x<-seq(0,1,by=0.02)
print(x)
y<-qnorm(x,mean=2,sd=1)
print(y)
plot(x,y)
#Create a sample of 8 numbers which are normally distributed
x<-rnorm(8)
print(x)
hist(x)
```

OUTPUT:

```
> print(x)
 [1] -10.0 -9.9 -9.8 -9.7 -9.6 -9.5 -9.4 -9.3 -9.2 -9.1 -9.0 -8.9 -8.8 -8.7 -8.6 -8.5 -8.4 -8.3 -
8.2 -8.1 -8.0 -7.9 -7.8 -7.7 -7.6 -7.5 -7.4 -7.3 -7.2 -7.1

 [31] -7.0 -6.9 -6.8 -6.7 -6.6 -6.5 -6.4 -6.3 -6.2 -6.1 -6.0 -5.9 -5.8 -5.7 -5.6 -5.5 -5.4 -5.3 -
5.2 -5.1 -5.0 -4.9 -4.8 -4.7 -4.6 -4.5 -4.4 -4.3 -4.2 -4.1

 [61] -4.0 -3.9 -3.8 -3.7 -3.6 -3.5 -3.4 -3.3 -3.2 -3.1 -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -
2.2 -2.1 -2.0 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1

 [91] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7
 0.8  0.9  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9

 [121]  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8
 3.9  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9

 [151]  5.0  5.1  5.2  5.3  5.4  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8
 6.9  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9

 [181]  8.0  8.1  8.2  8.3  8.4  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8
 9.9 10.0

> print(y)
 [1] 4.823441e-54 3.374391e-53 2.324064e-52 1.575850e-51 1.051952e-50 6.913387e-50
 4.473012e-49 2.849204e-48 1.786738e-47 1.103094e-46 6.704687e-46 4.011977e-45 2.363484e-44
 1.370758e-43

 [15] 7.826783e-43 4.399667e-42 2.434841e-41 1.326586e-40 7.115648e-40 3.757574e-39
 1.953507e-38 9.998535e-38 5.038160e-37 2.499319e-36 1.220634e-35 5.868994e-35 2.778151e-34
 1.294679e-33

 [29] 5.939949e-33 2.682980e-32 1.193071e-31 5.223109e-31 2.251159e-30 9.552069e-30
 3.990275e-29 1.641052e-28 6.644405e-28 2.648524e-27 1.039360e-26 4.015522e-26 1.527328e-25
 5.719219e-25

 [43] 2.108411e-24 7.652231e-24 2.734229e-23 9.618248e-23 3.330973e-22 1.135692e-21
 3.812097e-21 1.259742e-20 4.098391e-20 1.312681e-19 4.139228e-19 1.284972e-18 3.927190e-18
 1.181638e-17

 [57] 3.500266e-17 1.020779e-16 2.930737e-16 8.283922e-16 2.305203e-15 6.315339e-15
 1.703328e-14 4.522868e-14 1.182344e-13 3.042901e-13 7.709850e-13 1.923172e-12 4.722856e-12
```

1.141840e-11

[71] 2.717816e-11 6.368672e-11 1.469237e-10 3.336946e-10 7.461403e-10 1.642502e-09 3.559636e-09 7.594854e-09 1.595318e-08 3.299054e-08 6.716541e-08 1.346220e-07 2.656443e-07 5.160589e-07

[85] 9.869885e-07 1.858399e-06 3.444928e-06 6.286884e-06 1.129548e-05 1.997968e-05 3.479254e-05 5.964830e-05 1.006756e-04 1.672878e-04 2.736645e-04 4.407446e-04 6.988269e-04 1.090853e-03

[99] 1.676399e-03 2.536310e-03 3.777823e-03 5.539811e-03 7.997650e-03 1.136695e-02 1.590523e-02 2.191038e-02 2.971488e-02 3.967456e-02 5.215123e-02 6.748871e-02 8.598284e-02 1.078466e-01

[113] 1.331728e-01 1.618970e-01 1.937653e-01 2.283114e-01 2.648458e-01 3.024634e-01 3.400687e-01 3.764218e-01 4.102012e-01 4.400817e-01 4.648189e-01 4.833351e-01 4.947971e-01 4.986779e-01

[127] 4.947971e-01 4.833351e-01 4.648189e-01 4.400817e-01 4.102012e-01 3.764218e-01 3.400687e-01 3.024634e-01 2.648458e-01 2.283114e-01 1.937653e-01 1.618970e-01 1.331728e-01 1.078466e-01

[141] 8.598284e-02 6.748871e-02 5.215123e-02 3.967456e-02 2.971488e-02 2.191038e-02 1.590523e-02 1.136695e-02 7.997650e-03 5.539811e-03 3.777823e-03 2.536310e-03 1.676399e-03 1.090853e-03

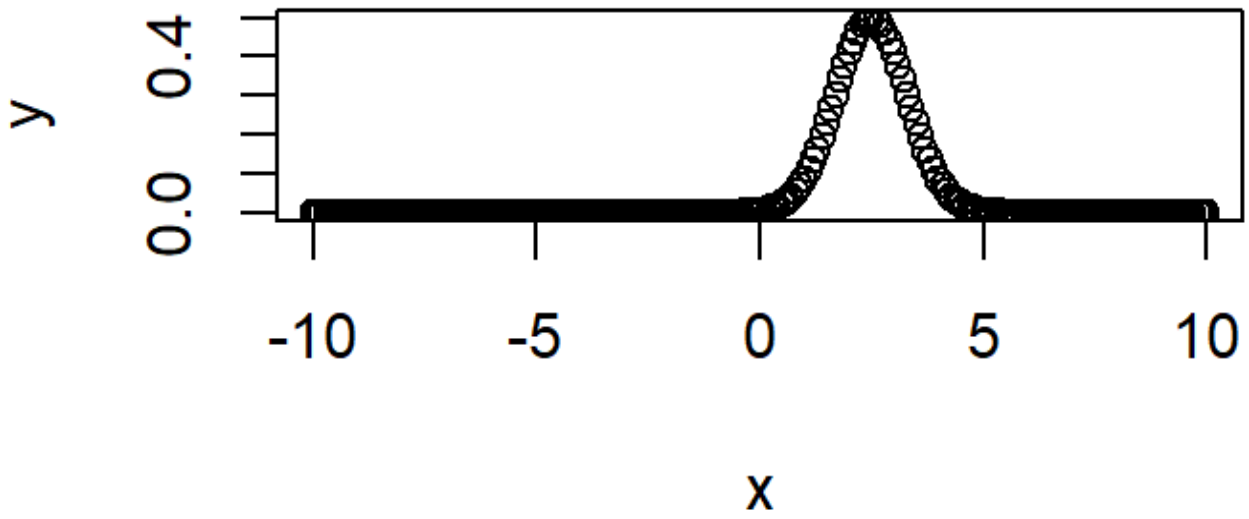
[155] 6.988269e-04 4.407446e-04 2.736645e-04 1.672878e-04 1.006756e-04 5.964830e-05 3.479254e-05 1.997968e-05 1.129548e-05 6.286884e-06 3.444928e-06 1.858399e-06 9.869885e-07 5.160589e-07

[169] 2.656443e-07 1.346220e-07 6.716541e-08 3.299054e-08 1.595318e-08 7.594854e-09 3.559636e-09 1.642502e-09 7.461403e-10 3.336946e-10 1.469237e-10 6.368672e-11 2.717816e-11 1.141840e-11

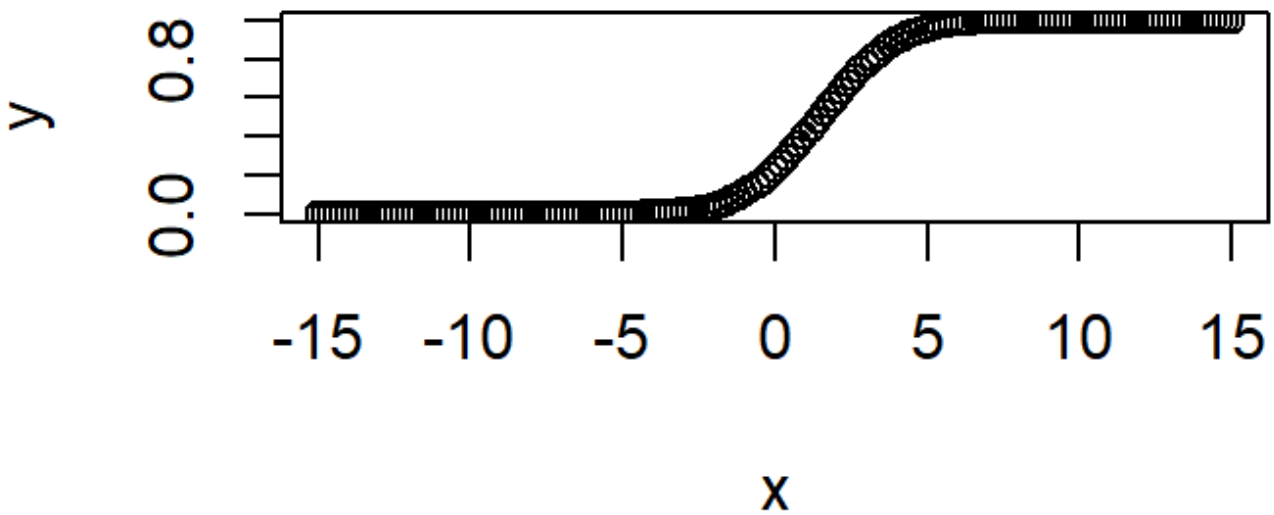
[183] 4.722856e-12 1.923172e-12 7.709850e-13 3.042901e-13 1.182344e-13 4.522868e-14 1.703328e-14 6.315339e-15 2.305203e-15 8.283922e-16 2.930737e-16 1.020779e-16 3.500266e-17 1.181638e-17

[197] 3.927190e-18 1.284972e-18 4.139228e-19 1.312681e-19 4.098391e-20

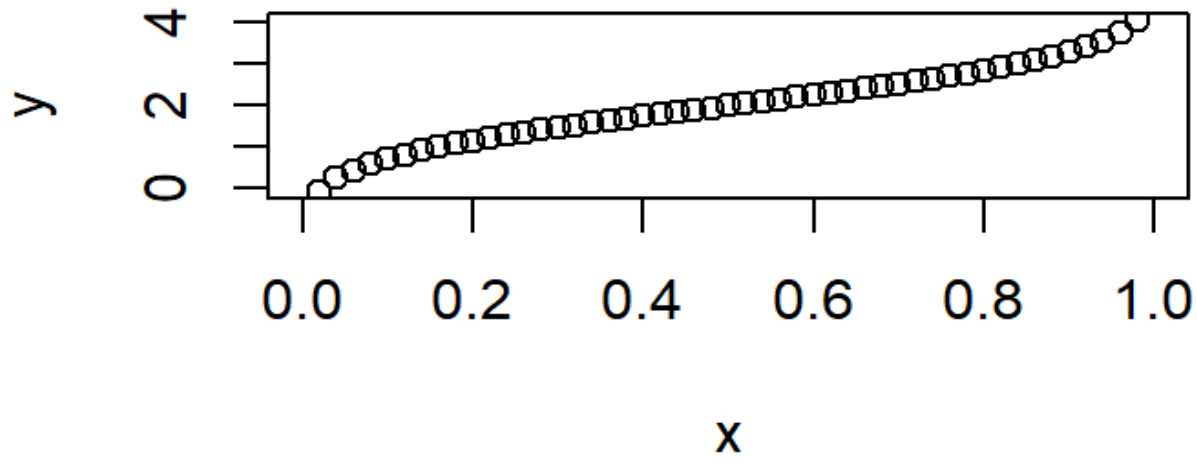
> plot(x,y)



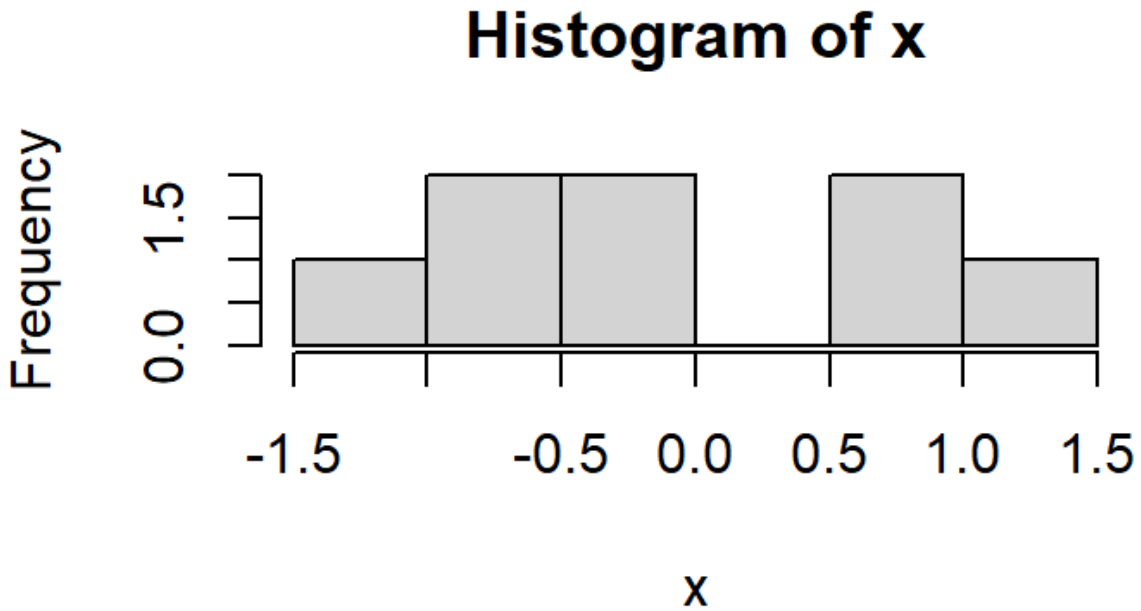
```
> print(y)
[1]   7.919726e-17 1.819621e-16 4.139702e-16 9.325576e-16 2.080186e-15
[6]   4.594627e-15 1.004897e-14 2.176291e-14 4.667012e-14 9.910343e-14
[11] 2.083858e-13 4.338895e-13 8.945890e-13 1.826431e-12 3.692499e-12
[16] 7.392258e-12 1.465465e-11 2.876854e-11 5.592508e-11 1.076575e-10
[21] 2.052263e-10 3.874147e-10 7.242292e-10 1.340712e-09 2.457865e-09
[26] 4.462172e-09 8.022392e-09 1.428348e-08 2.518491e-08 4.397712e-08
[31] 7.604961e-08 1.302432e-07 2.209050e-07 3.710674e-07 6.173074e-07
[36] 1.017083e-06 1.659675e-06 2.682296e-06 4.293514e-06 6.806877e-06
[41] 1.068853e-05 1.662376e-05 2.560882e-05 3.907560e-05 5.905891e-05
[46] 8.841729e-05 1.311202e-04 1.926156e-04 2.802933e-04 4.040578e-04
[51] 5.770250e-04 8.163523e-04 1.144207e-03 1.588870e-03 2.185961e-03
[56] 2.979763e-03 4.024589e-03 5.386146e-03 7.142811e-03 9.386706e-03
[61] 1.222447e-02 1.577761e-02 2.018222e-02 2.558806e-02 3.215677e-02
[66] 4.005916e-02 4.947147e-02 6.057076e-02 7.352926e-02 8.850799e-02
[71] 1.056498e-01 1.250719e-01 1.468591e-01 1.710561e-01 1.976625e-01
[76] 2.266274e-01 2.578461e-01 2.911597e-01 3.263552e-01 3.631693e-01
[81] 4.012937e-01 4.403823e-01 4.800612e-01 5.199388e-01 5.596177e-01
[86] 5.987063e-01 6.368307e-01 6.736448e-01 7.088403e-01 7.421539e-01
[91] 7.733726e-01 8.023375e-01 8.289439e-01 8.531409e-01 8.749281e-01
[96] 8.943502e-01 9.114920e-01 9.264707e-01 9.394292e-01 9.505285e-01
[101] 9.599408e-01 9.678432e-01 9.744119e-01 9.798178e-01 9.842224e-01
[106] 9.877755e-01 9.906133e-01 9.928572e-01 9.946139e-01 9.959754e-01
[111] 9.970202e-01 9.978140e-01 9.984111e-01 9.988558e-01 9.991836e-01
[116] 9.994230e-01 9.995959e-01 9.997197e-01 9.998074e-01 9.998689e-01
[121] 9.999116e-01 9.999409e-01 9.999609e-01 9.999744e-01 9.999834e-01
[126] 9.999893e-01 9.999932e-01 9.999957e-01 9.999973e-01 9.999983e-01
[131] 9.999990e-01 9.999994e-01 9.999996e-01 9.999998e-01 9.999999e-01
[136] 9.999999e-01 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
[141] 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
[146] 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
[151] 1.000000e+00
> plot(x,y)
```



```
> print(x)
 [1]  0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16 0.18 0.20 0.22 0.24 0.26 0.28
[16] 0.30 0.32 0.34 0.36 0.38 0.40 0.42 0.44 0.46 0.48 0.50 0.52 0.54 0.56 0.58
[31] 0.60 0.62 0.64 0.66 0.68 0.70 0.72 0.74 0.76 0.78 0.80 0.82 0.84 0.86 0.88
[46] 0.90 0.92 0.94 0.96 0.98 1.00
> print(y)
 [1]-Inf -0.05374891  0.24931393  0.44522641  0.59492844  0.71844843
 [7]    0.82501321  0.91968066  1.00554212  1.08463491  1.15837877  1.22780679
[13]  1.29369744  1.35665459  1.41715849  1.47559949  1.53230120  1.58753687
[19]  1.64154121  1.69451921  1.74665290  1.79810652  1.84903078  1.89956628
[25]  1.94984642  2.00000000  2.05015358  2.10043372  2.15096922  2.20189348
[31]  2.25334710  2.30548079  2.35845879  2.41246313  2.46769880  2.52440051
[37]  2.58284151  2.64334541  2.70630256  2.77219321  2.84162123  2.91536509
[43]  2.99445788  3.08031934  3.17498679  3.28155157  3.40507156  3.55477359
[49]  3.75068607  4.05374891         Inf
> plot(x,y)
```




```
> print(x)
 [1]  0.02102132 -0.88796845  0.40361616  1.18630630  1.43906084  1.10151421
 [7]  0.63144032 -0.04222271
> hist(x)
```



EXPERIMENT 11

AIM: Functions in R

a. Create a function to print squares of numbers in sequence.

b. Create a function without an argument

Functions:

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

→In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

→ R function is created by using the keyword function.

Syntax:

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

The different parts of a function are

1)Function Name − This is the actual name of the function. It is stored in R environment as an object with this name.

2)Arguments − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

3)Function Body − The function body contains a collection of statements that defines what the function does.

4)Return Value − The return value of a function is the last expression in the function body to be evaluated.

→R has many in-built functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as user defined functions.

PROGRAM:

```
#Create a function to print squares of numbers in sequence
#Create a function to print squares of numbers in sequence
fun_with_para<-function(a)
{
  for(i in 1:a)
  {
    b<-i^2
    print(b)
  }
}
fun_with_para(6)
#Create a function without argument
fun_without_para<-function()
{
  for (i in 1:10)
  {
    print(i^2)
  }
}
fun_without_para()
```

OUTPUT:

```
> fun_with_para(6)
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
> fun_without_para()
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

EXPERIMENT 12

AIM: Write a R- Code to

a. Calculate the mean
b. Calculate the standard error of the mean
c. Find the t-score that corresponds to the confidence level
d. Calculate the margin of error and construct the confidence interval

Mean:

Mean is calculated by taking the sum of the values and dividing with the number of values in a data series. The function mean() is used to calculate this in R

Standard error:

The formula for standard error of mean is the standard deviation divided by the square root of the length of the data. It is relatively simple in R to calculate the standard error of the mean. We can either use the std. error() function provided by the plotrix package, or we can easily create a function for the same

T-test:

A t score is one form of a standard test statistics the other shall come across elementary statistics is the Z-score.

→The t score formula enables you to take an individual score and transform it into a standardized form. One which helps you to compare scores

Margin error:

The margin of error is a statistic expressing the amount of random sampling error in the results of a survey. The larger the margin of error, the less confidence one should have that a poll result would reflect the result of a survey of the entire population

PROGRAM:

```
# Write a R code to calculate a mean
x<-c(1:5)
print(x)
mean(x)
#Calculate the standard error of mean
data<-c(1:6)
print(data)
std_err<-sd(data)/sqrt(length(data))
print(std_err)
# Find the t-score that correspons to confidence level
mydata<-data.frame(
  age=c(10,20,15,25,30),
  height=c(4.5,5,6,6.5,7),
  stringsAsFactors = FALSE)
print(mydata)
#H0=mean<=10
#One sided 95% confidence level
t.test(mydata$age,mean=10,alternate="less",conf.level =0.95)
```

OUTPUT:

```
> print(x)
[1] 1 2 3 4 5
> mean(x)
[1] 3
> print(data)
```

```
[1] 1 2 3 4 5 6
> print(std_err)
[1] 0.7637626
> print(mydata)
  age height
1  10    4.5
2  20    5.0
3  15    6.0
4  25    6.5
5  30    7.0
> t.test(mydata$age,mean=10,alternate="less",conf.level =0.95)
        One Sample t-test
data:  mydata$age
t = 5.6569, df = 4, p-value = 0.004813
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 10.18378 29.81622
sample estimates:
mean of x
       20
```

EXPERIMENT 13

AIM: Study and Implement Z-Test in R studio.

Z-test:
Z-test is a statistical method to determine whether the distribution of the test statistics can be approximated by a normal distribution. It is the method to determine whether two sample means are approximately the same or different when their variance is known and the sample size is large (should be >= 30).

When to Use Z-test:
The sample size should be greater than 30. Otherwise, we should use the t-test.
Samples should be drawn at random from the population.
The standard deviation of the population should be known.
Samples that are drawn from the population should be independent of each other.
The data should be normally distributed, however for large sample size, it is assumed to have a normal distribution.

Hypothesis Testing:
A hypothesis is an educated guess/claim about a particular property of an object. Hypothesis testing is a way to validate the claim of an experiment.

Null Hypothesis: The null hypothesis is a statement that the value of a population parameter (such as proportion, mean, or standard deviation) is equal to some claimed value. We either reject or fail to reject the null hypothesis. Null Hypothesis is denoted by H0.

Alternate Hypothesis: The alternative hypothesis is the statement that the parameter has a value that is different from the claimed value. It is denoted by HA.

Level of significance: It means the degree of significance in which we accept or reject the null-hypothesis. Since in most of the experiments 100% accuracy is not possible for accepting or rejecting a hypothesis, so we, therefore, select a level of significance. It is denoted by alpha ($\propto$).

Steps to perform Z-test:
First, identify the null and alternate hypotheses.
Determine the level of significance ($\propto$).
Find the critical value of z in the z-test using
Calculate the z-test statistics. Below is the formula for calculating the z-test statistics.
Formula:
Z=Xbar-mean/sd/sqrt(n)
where,

X: mean of the sample.
Mu: mean of the population.
Sd: Standard deviation of the population.
n: sample size.
Now compare with the hypothesis and decide whether to reject or not to reject the null hypothesis

PROGRAM:

```
print(getwd())
setwd(“set the path”)
data<-read.csv("population.csv")
print(data)
#Null Hypothesis: mean<=1000
#Alternate Hypothesis: mean<1000
# Z Test Formula: x bar-mean
                  Sd/sqrt(n)
```

OUTPUT:

	population
1	1000
2	2000
3	3500
4	4500
5	5000

```
zstat<-(mean(data$population)-1000)/(sd(data$population)/sqrt(nrow(data)))
print(zstat)
```

OUTPUT:

[1] 2.926836

EXPERIMENT 14

AIM: Write R scripts to plot data in Pie Chart, Histograms and graphs.

R graphs:
R language is mostly used for statistics and data analytics purposes to represent the data graphically in the software. To represent those data graphically, charts and graphs are used in R.

R – graphs:
There are hundreds of charts and graphs present in R. For example, bar plot, box plot, mosaic plot, dot chart, coplot, histogram, pie chart, scatter graph, etc.

Types of R – Charts
1)Bar Plot or Bar Chart
2)Pie Diagram or Pie Chart
3)Histogram
4)Scatter Plot
5)Box Plot

Bar Plot or Bar Chart
Bar plot or Bar Chart in R is used to represent the values in data vector as height of the bars. The data vector passed to the function is represented over y-axis of the graph. Bar chart can behave like histogram by using table() function instead of data vector.
Syntax: barplot(data, xlab, ylab)
where:
data is the data vector to be represented on y-axis
xlab is the label given to x-axis
ylab is the label given to y-axis

Pie Diagram or Pie Chart:
Pie chart is a circular chart divided into different segments according to the ratio of data provided. The total value of the pie is 100 and the segments tell the fraction of the whole pie. It is another method to represent statistical data in graphical form and pie() function is used to perform the same.
Syntax: pie(x, labels, col, main, radius)
where,
x is data vector
labels shows names given to slices
col fills the color in the slices as given parameter
main shows title name of the pie chart
radius indicates radius of the pie chart. It can be between -1 to +1

Histogram:
Histogram is a graphical representation used to create a graph with bars representing the frequency of grouped data in vector. Histogram is same as bar chart but only difference between them is histogram represents frequency of grouped data rather than data itself.
Syntax: hist(x, col, border, main, xlab, ylab)
where:
x is data vector
col specifies the color of the bars to be filled
border specifies the color of border of bars
main specifies the title name of histogram
xlab specifies the x-axis label
ylab specifies the y-axis label

Scatter Plot

A Scatter plot is another type of graphical representation used to plot the points to show relationship between two data vectors. One of the data vectors is represented on x-axis and another on y-axis.

Syntax: plot(x, y, type, xlab, ylab, main)

Where,

x is the data vector represented on x-axis

y is the data vector represented on y-axis

type specifies the type of plot to be drawn. For example, “l” for lines, “p” for points, “s” for stair steps, etc.

xlab specifies the label for x-axis

ylab specifies the label for y-axis

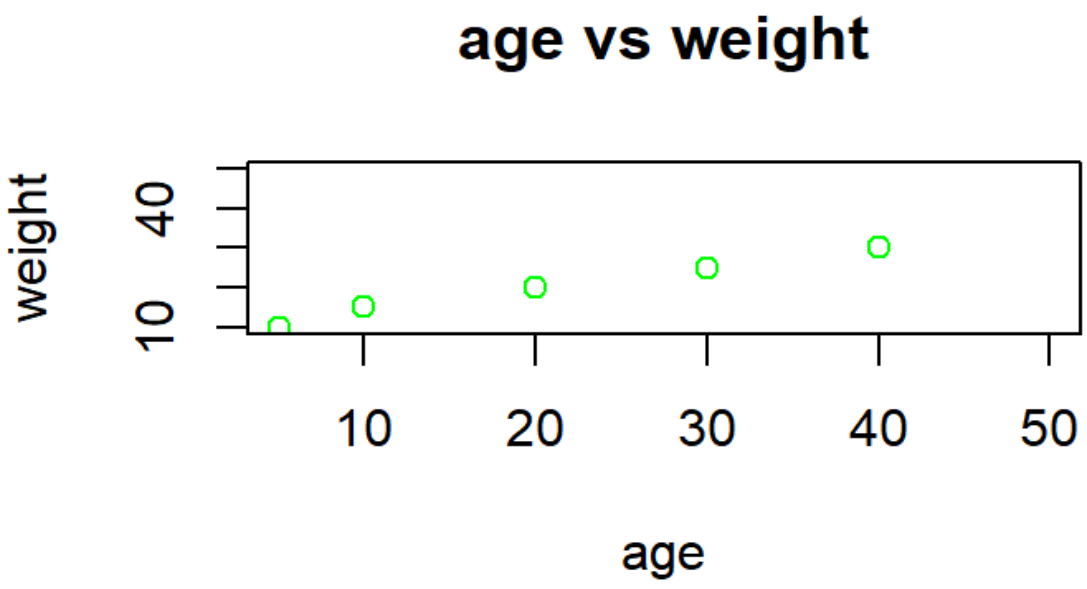
main specifies the title name of the graph

PROGRAM:

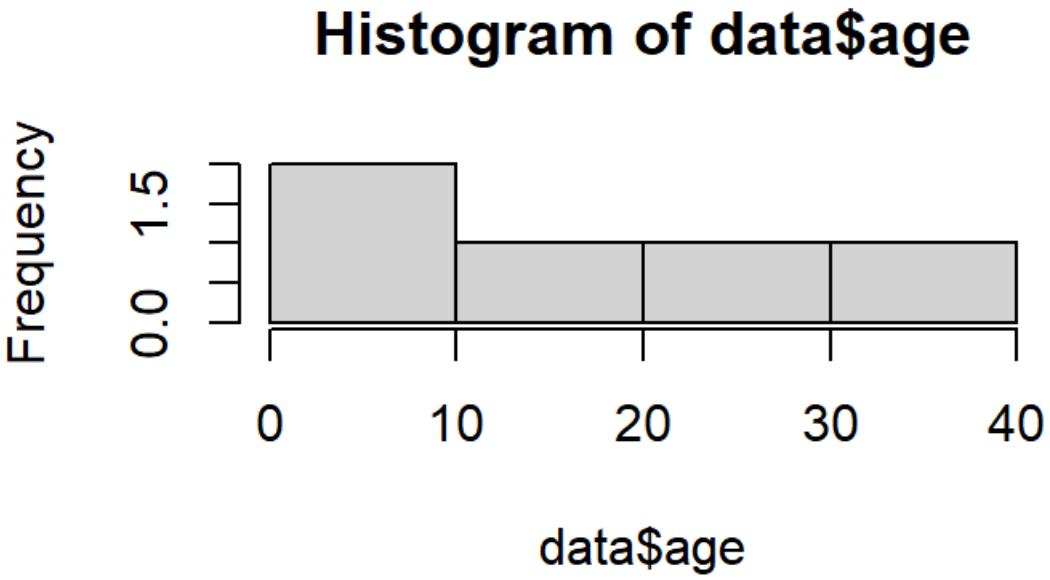
```
# Create a data Frame
data<-data.frame(age=c(5,10,20,30,40),
                 weight=c(10,15,20,25,30),
                 stringsAsFactors=FALSE)
print(data)
#creating a graph
plot(x=data$age,y=data$weight,main="age vs
weight",xlab="age",ylab="weight",xlim=c(5,50),ylim=c(10,50),pch=1,col="green")
#Creating a histogram using age
hist(data$age)
#Creating a pie chart
pie(data$weight,data$age)
```

Output:

```
> print(data)
  age weight
1   5     10
2  10     15
3  20     20
4  30     25
5  40     30
>plot(x=data$age,y=data$weight,main="age vs
weight",xlab="age",ylab="weight",xlim=c(5,50),ylim=c(10,50),pch=1,col="green")
```



>hist(data$age)



>pie(data$weight,data$age)