Ex.No:1 **Basic Operations in R Programming**

Date:

Aim:

To Perform Basic Operations in R Programming.

Assignment:

```
# assignment
x <- 3
# leftward assignment
x <- value

x = value
x <<- value
# rightward assignment
value -> x
value ->> x
```

Evaluation

```
# evaluation
x
## [1] 3
```

Case Sensitivity

```
x <- 1
y <- 3
z <- 4
x * y * z
## [1] 12
x * Y * z
## Error in eval(expr, envir, enclos): object 'Y' not found
```

Basic Arithmetic

```
8 + 9 / 5 ^ 2
## [1] 8.36
8 + 9 / (5 ^ 2)
## [1] 8.36
8 + (9 / 5) ^ 2
## [1] 11.24
(8 + 9) / 5 ^ 2
## [1] 0.68
1 / 7
```

```
## [1] 0.1428571
options(digits = 3)
1 / 7
## [1] 0.143
pi
## [1] 3.141592654
options(digits = 22)
pi
## [1] 3.141592653589793115998
42 / 4 # regular division
## [1] 10.5
42 %/% 4 # integer division
## [1] 10
42 %% 4 # modulo (remainder)
## [1] 2
```

Miscellaneous Mathematical Functions:

```
x <- 10
abs(x) # absolute value
sqrt(x) # square root
exp(x) # exponential transformation
log(x) # logarithmic transformation
cos(x) # cosine and other trigonometric functions
```

Infinite, and NaN Numbers:

```
## [1] Inf
Inf - Inf # infinity minus infinity
## [1] NaN
x <- 2
y <- 3
# list all objects
ls()
# identify if an R object with a given name is present
exists("x")
# remove defined object from the environment
rm(x)
# you can remove multiple objects
rm(x, y)
# basically removes everything in the working environment -- use with caution!
rm(list = ls())
```

Result: Thus basic R Program has been executed.

Ex.No:2 **Vectors Manipulation**

Date:

Aim: To Implement various vector operations in R Programming.

R has five basic or "atomic" classes of objects:
• character
• numeric (real numbers)
• integer
• complex
• logical (True/False)

Creating Vectors:

The c() function can be used to create vectors of objects by concatenating things together.

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
> x <- vector("numeric", length = 10)
> x
[1] 0 0 0 0 0 0 0 0 0 0
```

Numeric vector:

```
x <- c(1,2,3,4,5,6)
```

Character vector:

```
State <- c("DL", "MU", "NY", "DL", "NY", "MU")

table(State)
```

To calculate mean for a vector use mean function

```
>x<-c(1,2,3,NA,5,6)

>mean(x)

[1] NA
```

To calculate mean for a vector excluding NA values, we can include na.rm = TRUE parameter in mean function.

```
>x<-c(1,2,3,NA,5,6)
```

```
>mean(x)

[1]  NA

>mean(x,na.rm=TRUE)

[1] 3.4
```

To Perform Sum from 3rd to 5th position:

```
>x<-c(1,2,3,4,5,6)
>sum(x[c(3,5)])
[1] 8
```

Convert a column "x" to numeric
data$x = as.numeric(data$x)

```
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10

> 9:5
[1] 9 8 7 6 5

> seq(from=2, to=6, by=0.4)
[1] 2.0 2.4 2.8 3.2 3.6 4.0 4.4 4.8 5.2 5.6 6.0

> seq(from=-1, to=1, length=6)
[1] -1.0 -0.6 -0.2 0.2 0.6 1.0

> rep(5,3)
[1] 5 5 5

> rep(2:5,each=3)
[1] 2 2 2 3 3 3 4 4 4 5 5 5

> rep(-1:3, length.out=10)
[1] -1 0 1 2 3 -1 0 1 2 3

> 2^(0:10)
[1] 1 2 4 8 16 32 64 128 256 512 1024

> 1:3 + rep(seq(from=0,by=10,to=30), each=3)
[1] 1 2 3 11 12 13 21 22 23 31 32 33
```

Result:

Thus Various types of Vector creation and operations is performed using R Programming.

Ex.No:3 **List Operations**

Date:

Aim: To Implement various List operations in R Programming.

```
> mylist <- list(x, y, z, gender, mydata)
> mylist
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 1 3 5 7 9

[[3]]
[1] 1 2 5 4 7

[[4]]
[1] "m" "f" "m" "m" "f"

[[5]]
  x y z gender
1 1 1 1      m
2 2 3 2      f
3 3 5 5      m
4 4 7 4      m
5 5 9 7      f
```

Subscripts to select the specific component of the list.

```
> mylist[[3]]
[1] 1 2 5 4 7
```

> x <- list(1:3, TRUE, "Hello", list(1:2, 5))
Here x has 4 elements: a numeric vector, a logical, a string and another list

To get a sub-list, use single brackets:
> x[c(1,3)]
[[1]]
[1] 1 2 3
[[2]]
[1] "Hello"

We can also name some or all of the entries in our list, by supplying argument names to list():
> x <- list(y=1:3, TRUE, z="Hello")

```
> x
$y
[1] 1 2 3
[[2]]

[1] TRUE
$z
[1] "Hello"
```

use the numeric
position if we prefer:

```
> x$y
[1] 1 2 3
> x[[1]]

[1] 1 2 3
```

The function names() can be used to obtain a character vector of all the names of objects in a list.

```
> names(x)
[1] "y" "" "z"
```

Result:

Thus Various types of List operations is performed using R Programming.

Ex.No:3 **Implementation of Factors**

Date:

Aim: To Implement Factor operations in R Programming.

```
# Creating a vector
x < -c("female", "male", "male", "female")
print(x)
 # Converting the vector x into a factor
# named gender
gender < -factor(x)
print(gender)
```

Output:

```
[1] "female" "male"   "male"   "female"
[1] female male   male   female
Levels: female male
```

Levels can also be predefined by the programmer.

```
gender <- factor(c("female", "male", "male", "female"));
print(is.factor(gender))
```

Output:

```
[1] TRUE
```

class():

```
gender <- factor(c("female", "male", "male", "female"));
class(gender)
```

Output:

```
[1] "factor"
```

Accessing elements of a Factor in R

Like we access elements of a vector, the same way we access the elements of a factor. If gender is a factor then gender[i] would mean accessing i^{th} element in the factor.

```
gender <- factor(c("female", "male", "male", "female"));
gender[3]
```

Output:

```
[1] male
Levels: female male
```

More than one element can be accessed at a time.

```
gender <- factor(c("female", "male", "male", "female"));
gender[c(2, 4)]
```

Output:
[1] male   female
Levels: female male

```
gender <- factor(c("female", "male", "male", "female"  ));
gender[-3]
```

Output:
[1] female male   female
Levels: female male
Modification of a Factor in R
After a factor is formed, its components can be modified but the new values which need to be assigned must be at the predefined level.
Example:

```
gender <- factor(c("female", "male", "male", "female"  ));
gender[2]<-"female"
gender
```

Output:
[1] female female male   female
Levels: female male

```
gender <- factor(c("female", "male", "male", "female"  ));

# add new level
levels(gender) <- c(levels(gender), "other")
gender[3] <- "other"
gender
```

Output:
[1] female male   other female
Levels: female male other

Factors in Data Frame :

```
age <- c(40, 49, 48, 40, 67, 52, 53)
salary <- c(103200, 106200, 150200,
         10606, 10390, 14070, 10220)
gender <- c("male", "male", "transgender",
         "female", "male", "female", "transgender")
employee<- data.frame(age, salary, gender)
print(employee)
print(is.factor(employee$gender))
```

Output:

```
  age salary      gender
1  40 103200        male
2  49 106200        male
3  48 150200 transgender
4  40  10606      female
5  67  10390        male
6  52  14070      female
7  53  10220 transgender
[1] TRUE
```

Result:

Thus Various types of Factor operations is performed using R Programming.

Ex.No:5 **Matrix Manipulation**

Date:

Aim: To Implement various Matrix operations in R Programming.

```
> matrix(1:12, nrow=3, ncol=4)
[,1] [,2] [,3] [,4]
[1,] 1 4 7 10
[2,] 2 5 8 11
[3,] 3 6 9 12
```

```
# R program to illustrate

# special matrices

 # Matrix having 3 rows and 3 columns

# filled by a single constant 5

print(matrix(5, 3, 3))
```

Output:

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

[1,] 5 5 5

[2,] 5 5 5

[3,] 5 5 5
```

Diagonal matrix:

A diagonal matrix is a matrix in which the entries outside the main diagonal are all zero. To create

such a matrix the syntax is given below:

```
# filled by array of elements (5, 3, 3)

print(diag(c(5, 3, 3), 3, 3))
```

Output:

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

[1,] 5 0 0

[2,] 0 3 0

[3,] 0 0 3

Identity matrix:

A square matrix in which all the elements of the principal diagonal are ones and all other elements are

zeros. To create such a matrix the syntax is given below:

```
# R program to illustrate

# special matrices

 # Identity matrix having

# 3 rows and 3 columns

print(diag(1, 3, 3))
```

Output:

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

[1,] 1 0 0

[2,] 0 1 0

[3,] 0 0 1

Matrix metrics

```
# R program to illustrate

# matrix metrics

 # Create a 3x3 matrix

A = matrix( c(1, 2, 3, 4, 5, 6, 7, 8, 9),  nrow = 3,  ncol = 3,  byrow = TRUE )
```

```
cat(“The 3x3 matrix:”)

print(A)

 cat(“Dimension of the matrix:”)

print(dim(A))

 cat(“Number of rows:”)

print(nrow(A))

 cat(“Number of columns:\n”)

print(ncol(A))

 cat(“Number of elements:\n”)

print(length(A))
```

Accessing elements of a Matrix:

```
# R program to illustrate
# access rows in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

# Accessing first and second row
cat("Accessing first and second row\n")
print(A[1:2, ])
```

Output:

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9
```

```
Accessing first and second row
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
```

Accessing columns:

```
# R program to illustrate
# access columns in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

# Accessing first and second column
cat("Accessing first and second column\n")
print(A[, 1:2])
```

Output:

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

Accessing first and second column
     [, 1] [, 2]
[1, ]    1    2
[2, ]    4    5
[3, ]    7    8
```

Accessing elements of a matrix:

```
# R program to illustrate
# access an entry in metrics
 # Create a 3x3 matrix
```

```
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

# Accessing 2
print(A[1, 2])
 # Accessing 6
print(A[2, 3])
```

Output:

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

[1] 2
[1] 6
```

Accessing Submatrices:
We can access submatrix in a matrix using the **colon(:)** operator.

```
# R program to illustrate
# access submatrices in a matrix
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

cat("Accessing the first three rows and the first two columns\n")
print(A[1:3, 1:2])
```

Output:

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

Accessing the first three rows and the first two columns
     [, 1] [, 2]
[1, ]    1    2
[2, ]    4    5
[3, ]    7    8
```

Modifying elements of a Matrix

In R you can modify the elements of the matrices by a direct assignment.

Example:

```
# R program to illustrate
# editing elements in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

# Editing the 3rd rows and 3rd column element
# from 9 to 30
# by direct assignments
A[3, 3] = 30
 cat("After edited the matrix\n")
print(A)
```

Output:

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9
```

After edited the matrix

```
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8   30
```

Matrix Concatenation

Matrix concatenation refers to the merging of rows or columns of an existing matrix.

Concatenation of a row:

The concatenation of a row to a matrix is done using **rbind()**.

```
# R program to illustrate
# concatenation of a row in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

# Creating another 1x3 matrix
B = matrix(
  c(10, 11, 12),
  nrow = 1,
  ncol = 3
)
cat("The 1x3 matrix:\n")
print(B)

# Add a new row using rbind()
C = rbind(A, B)
 cat("After concatenation of a row:\n")
print(C)
```

Output:

```
 The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9
The 1x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]   10   11   12
```

After concatenation of a row:

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

Concatenation of a column:

The concatenation of a column to a matrix is done using **cbind()**.

```
# R program to illustrate
# concatenation of a column in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)

# Creating another 3x1 matrix
B = matrix(
  c(10, 11, 12),
  nrow = 3,
  ncol = 1,
  byrow = TRUE
)
cat("The 3x1 matrix:\n")
print(B)

# Add a new column using cbind()
C = cbind(A, B)
 cat("After concatenation of a column:\n")
print(C)
```

Output:

```
 The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9
The 3x1 matrix:
     [, 1]
[1, ]   10
```

```
[2, ]   11
[3, ]   12
```

After concatenation of a column:

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

Dimension inconsistency: Note that you have to make sure the consistency of dimensions between the matrix before you do this matrix concatenation.

```
# R program to illustrate
# Dimension inconsistency in metrics concatenation
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("The 3x3 matrix:\n")
print(A)
 # Creating another 1x3 matrix
B = matrix(
  c(10, 11, 12),
  nrow = 1,
  ncol = 3,
)
cat("The 1x3 matrix:\n")
print(B)
 # This will give an error
# because of dimension inconsistency
C = cbind(A, B)
 cat("After concatenation of a column:\n")
print(C)
```

Output:

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9
The 1x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]   10   11   12
```

Error in cbind(A, B) : number of rows of matrices must match (see arg 2)

Deleting rows and columns of a Matrix
To delete a row or a column, first of all, you need to access that row or column and then insert a negative sign before that row or column. It indicates that you had to delete that row or column.
Row deletion:

```
# R program to illustrate
# row deletion in metrics
 # Create a 3x3 matrix
A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9),nrow = 3, ncol = 3, byrow = TRUE)
cat("Before deleting the 2nd row\n")
print(A)

# 2nd-row deletion
A = A[-2, ]

cat("After deleted the 2nd row\n")
print(A)
```

Output:

```
Before deleting the 2nd row
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

After deleted the 2nd row
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    7    8    9
```

Column deletion:

```
 # R program to illustrate
# column deletion in metrics
 # Create a 3x3 matrix
A = matrix(  c(1, 2, 3, 4, 5, 6, 7, 8, 9),  nrow = 3  ncol = 3, byrow = TRUE )
cat("Before deleting the 2nd column\n")
print(A)

# 2nd-row deletion
A = A[, -2]
```

```
 cat("After deleted the 2nd column\n")
print(A)
```

Output:

```
Before deleting the 2nd column
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

After deleted the 2nd column
     [, 1] [, 2]
[1, ]    1    3
[2, ]    4    6
[3, ]    7    9
```

Matrix Muliplication:

```
data <- c(1, 2, 3, 0, 1, 2, 0, 0, 1)
A <- matrix(data, nrow = 3, ncol = 3)

data <- c(0, 1, 1, 1, 0, 3, 1, 3, 3)
B <- matrix(data, nrow = 3, ncol = 3)


AB <- A %*% B

print("Matrix A")
print(A)
print("Matrix B")
print(B)
print("Matrix Multiplication Result")
print(AB)
```

Output:

```
[1] "Matrix A"
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    2    1    0
[3,]    3    2    1
[1] "Matrix B"
     [,1] [,2] [,3]
[1,]    0    1    1
```

```
[2,]    1    0    3
[3,]    1    3    3
[1] "Matrix Multiplication Result"
     [,1] [,2] [,3]
[1,]    0    1    1
[2,]    1    2    5
[3,]    3    6   12
```

Result:

Thus the various implementation of Matrix operations in R Programming was executed successfully.

Ex.No:6 **Histogram – ungrouped frequency data**

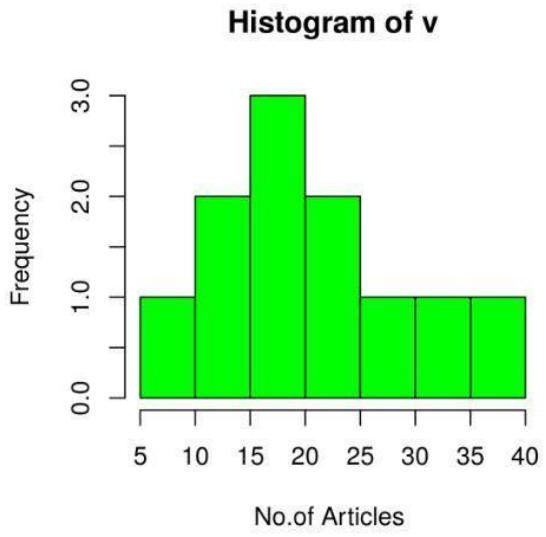
Date:

Aim: To draw a histogram for ungrouped frequency data in R programming

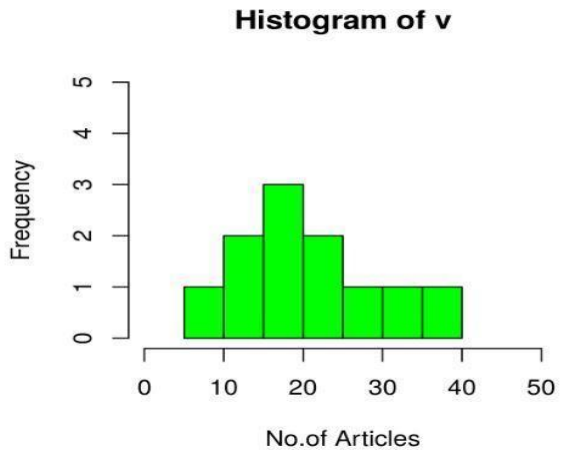
Program:

```
# Create data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32,14, 19, 27, 39)

# Create the histogram.
hist(v, xlab = "No.of Articles ", col = "green", border = "black")
```

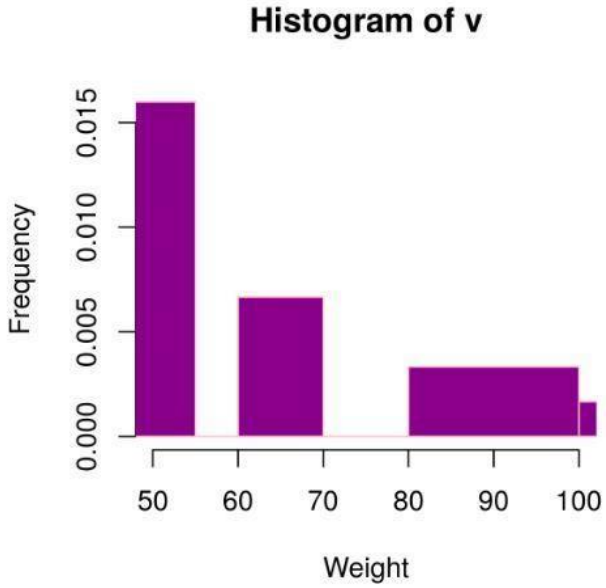
Output:




```
# Create data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19, 27, 39)
  # Create the histogram.
hist(v, xlab = "No.of Articles", col = "green",border = "black", xlim = c(0, 50),ylim = c(0, 5),
breaks = 5)
```




```
# Creating data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32, 14,19, 27, 39, 120, 40, 70, 90)

# Creating the histogram.
hist(v, xlab = "Weight", ylab ="Frequency", xlim = c(50, 100), col = "darkmagenta", border =
"pink",    breaks = c(5, 55, 60, 70, 75, 80, 100, 140))
```



Result:

Thus the various display method of histogram for ungrouped data in R Programming was executed successfully.

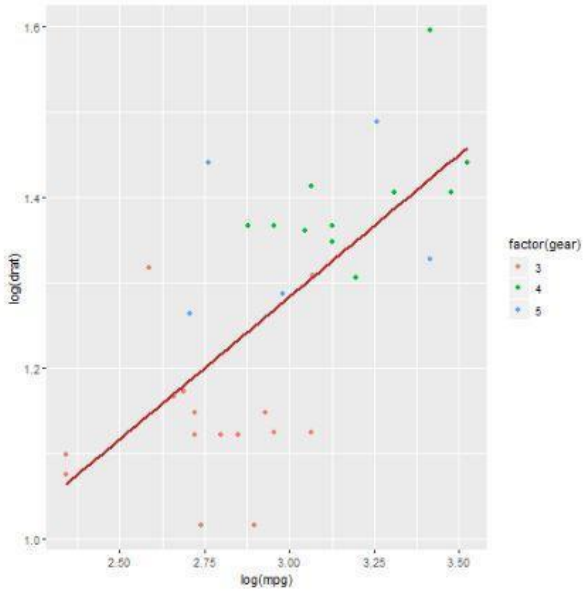
Ex.No:7 **Simple Linear Regression**

Date:

Aim: To fit a simple linear regression equation for a sample data and visualize using a scatter plot in R programming.

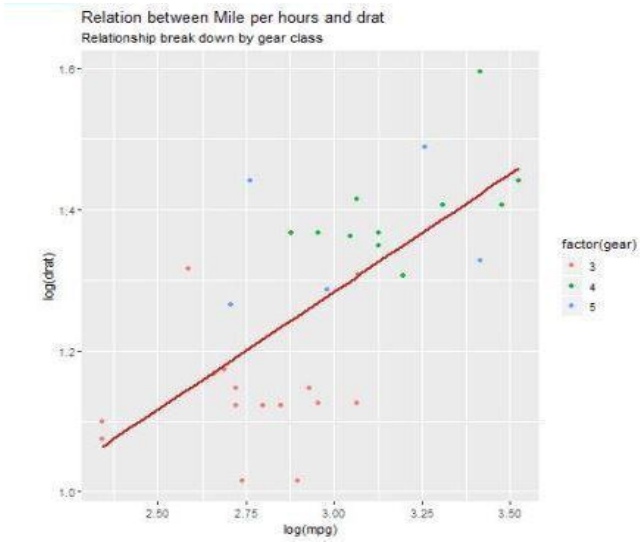
Program:

```
# Loading ggplot2 package
library(ggplot2)

# Creating scatterplot with fitted values.
# An additional function stst_smooth
# is used for linear regression.
ggplot(mtcars, aes(x = log(mpg), y = log(drat))) + geom_point(aes(color = factor(gear))) +
      stat_smooth(method = "lm", col = "#C42126", se = FALSE, size = 1)
```




```
# Loading ggplot2 package
library(ggplot2)

# Creating scatterplot with fitted values.
# An additional function stst_smooth
# is used for linear regression.
new_graph<-ggplot(mtcars, aes(x = log(mpg), y = log(drat))) + geom_point(aes(color =
factor(gear))) + stat_smooth(method = "lm", col = "#C42126", se = FALSE, size = 1)

# in above example lm is used for linear regression
# and se stands for standard error.
# Adding title with dynamic name
```

new_graph + labs(title = "Relation between Mile per hours and drat", subtitle = "Relationship break down by gear class", caption = "Authors own computation")



Result:

Thus the simple linear regression for the given dataset has been represented by scatter plot in R Programming was executed successfully

Ex.No:8 **One Sample T Test**

Date:

Aim: To demonstrate one sample T Test using R programming.

Program:

```
set.seed(0)
sweetSold <- c(rnorm(50, mean = 140, sd = 5))
t.test(sweetSold, mu = 150) # Ho: mu = 150
```

Output:

```
        One Sample t-test

data:  sweetSold
t = -15.249, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 150
95 percent confidence interval:
 138.8176 141.4217
sample estimates:
mean of x
 140.1197
```

```
my_data <- data.frame( name = paste0(rep("M_", 10), 1:10), weight = round(rnorm(10, 20, 2), 1)
)

head(my_data, 10)
```

```
   name weight
1   M_1   17.6
2   M_2   20.6
3   M_3   22.2
4   M_4   15.3
5   M_5   20.9
6   M_6   21.0
7   M_7   18.9
8   M_8   18.9
9   M_9   18.9
10 M_10   18.2
```

```
# One-sample t-test

res <- t.test(my_data$weight, mu = 25)
```

```
# Printing the results

res


    One Sample t-test
data:  my_data$weight
t = -9.0783, df = 9, p-value = 7.953e-06
alternative hypothesis: true mean is not equal to 25
95 percent confidence interval:
 17.8172 20.6828
sample estimates:
mean of x
    19.25
```

Result:

Thus the one sample T Test in R Programming was executed successfully.

Ex.No:9 **Two Sample T Test**

Date:

Aim: To demonstrate two sample T Test using R programming.

Program:

```
shopOne <- rnorm(50, mean = 140, sd = 4.5)
shopTwo <- rnorm(50, mean = 150, sd = 4)
t.test(shopOne, shopTwo, var.equal = TRUE)
```

Output:

```
        Two Sample t-test

data:  shopOne and shopTwo
t = -12.883, df = 98, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.384388  -8.345285
sample estimates:
mean of x mean of y
 140.5360  150.4008
```

Paired Sample T-test

```
sweetOne <- c(rnorm(100, mean = 14, sd = 0.3))
sweetTwo <- c(rnorm(100, mean = 13, sd = 0.2))
t.test(sweetOne, sweetTwo, paired = TRUE)
```

```
        Paired t-test

data:  sweetOne and sweetTwo
t = 29.31, df = 99, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.9892738 1.1329434
sample estimates:
mean of the differences
               1.061109
```

Result:

Thus the two sample T Test in R Programming was executed successfully.

Ex.No:10 **Graphs using Plot function**

Date:

Aim: To create graphs using various plot functions using R programming.

Programs:

```
library(datasets)
data(mtcars)

head(mtcars)
```

Output:

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

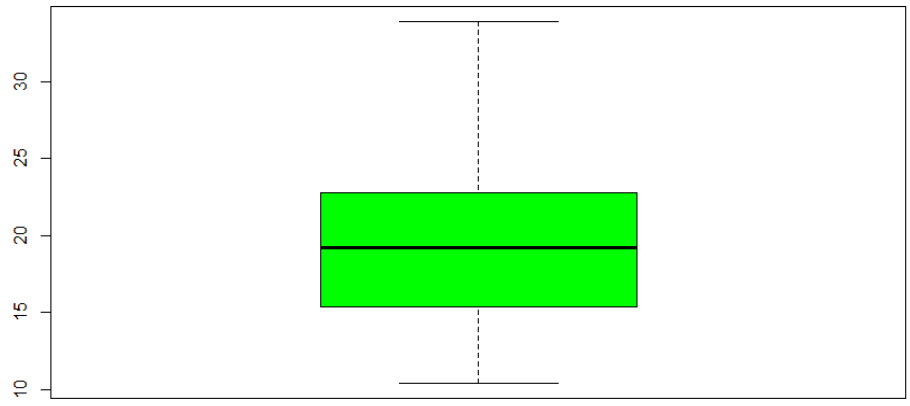
```
summary(mtcars)
```

Output:

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90
```

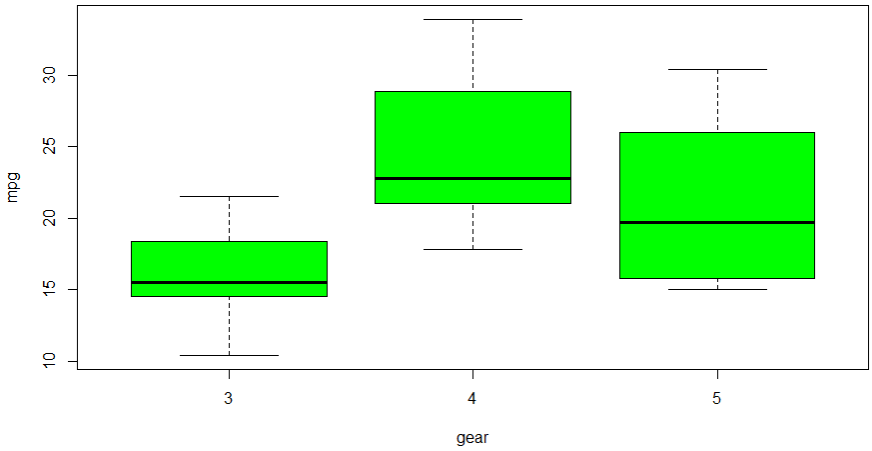
```
boxplot(mtcars$mpg, col="green")
```

Output:



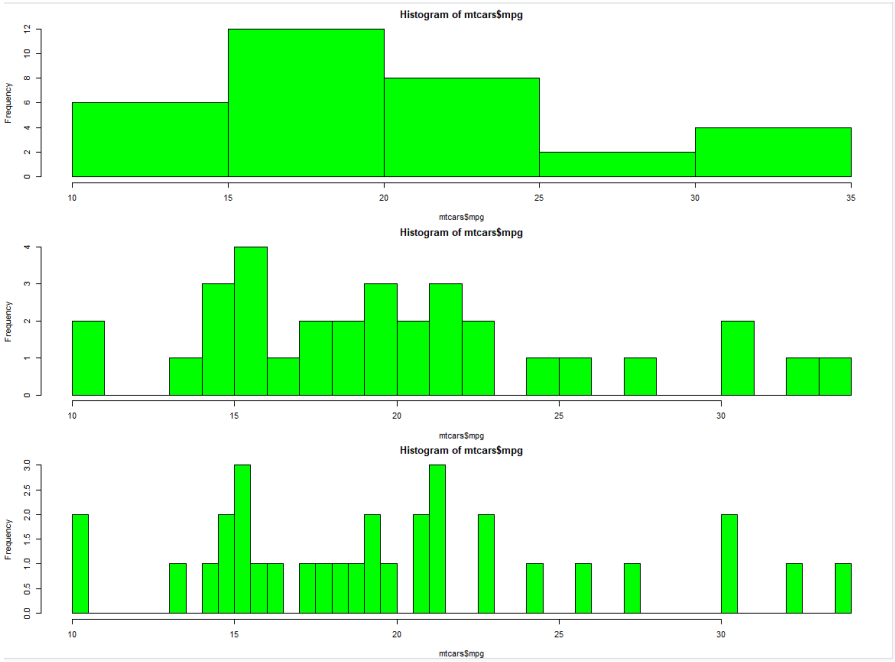
boxplot(mpg~gear, data=mtcars, col = "green")

Output:



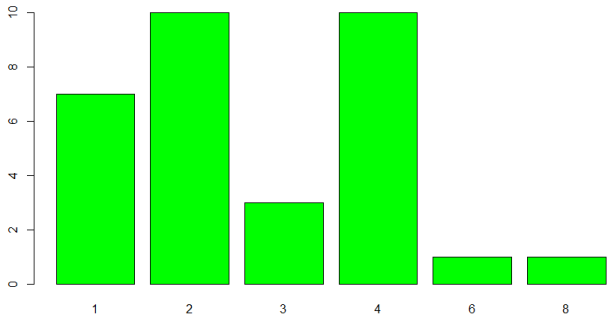
hist(mtcars$mpg, col = "green") ## Plot 1
hist(mtcars$mpg, col = "green", breaks = 25) ## Plot 2
hist(mtcars$mpg, col = "green", breaks = 50) ## Plot 3

Output:



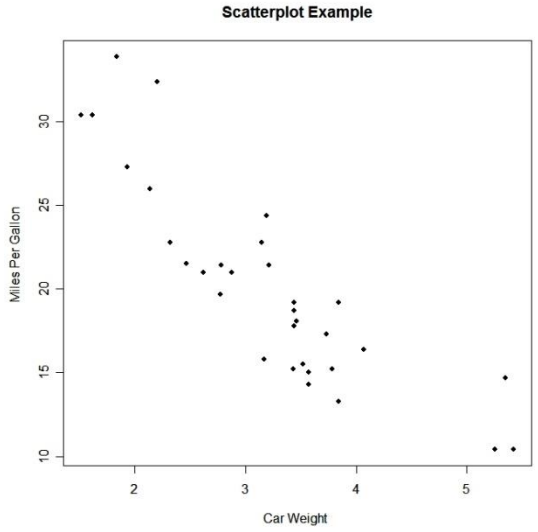
barplot(table(mtcars$carb), col="green")

Output:



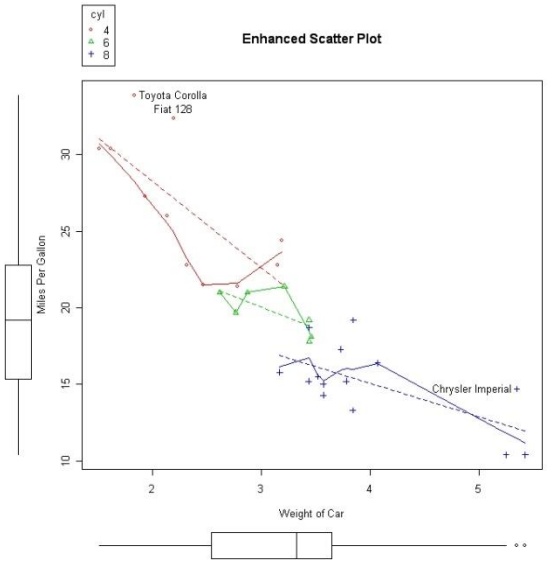
attach(mtcars)
plot(wt, mpg, main="Scatterplot Example", xlab="Car Weight ", ylab="Miles Per Gallon ", pch=19)

Output:



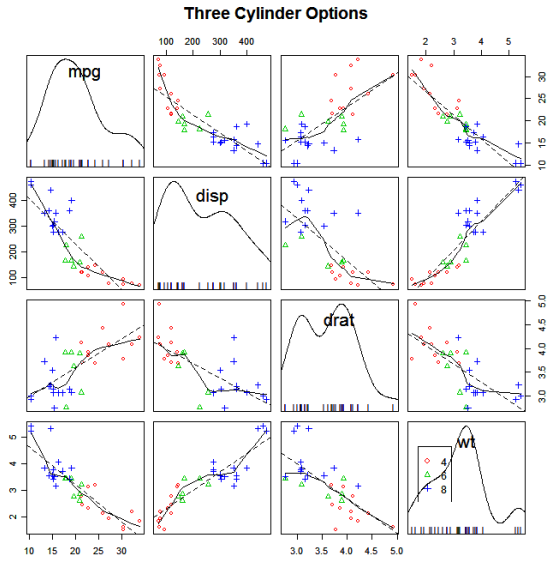
library(car)
scatterplot(mpg ~ wt | cyl, data=mtcars, xlab="Weight of Car", ylab="Miles Per Gallon", main="Enhanced Scatter Plot",labels=row.names(mtcars))

Output:



# Scatterplot Matrices from the car Package
library(car)
scatterplot.matrix(~mpg+disp+drat+wt|cyl, data=mtcars, main="Three Cylinder Options")

Output:



Result:

Thus the various kinds of graphs using plot function in R Programming was executed successfully.

Ex.No:11 **3D Plots**

Date:

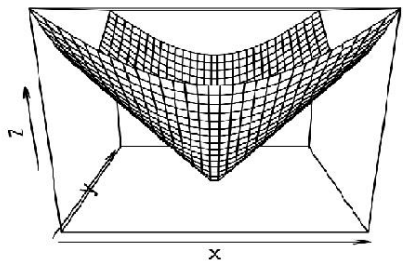
Aim: To create 3Dplots using R programming.

Program:

Simple Right Circular Cone

```
# To illustrate simple right circular cone
cone <- function(x, y){
sqrt(x ^ 2 + y ^ 2)
}

# prepare variables.
x <- y <- seq(-1, 1, length = 30)
z <- outer(x, y, cone)

# plot the 3D surface
persp(x, y, z)
```

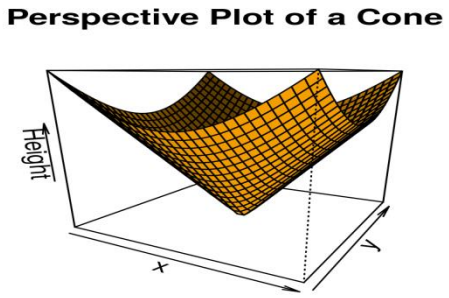
Output:




```
# Adding Titles and Labeling Axes to Plot
cone <- function(x, y){
sqrt(x ^ 2 + y ^ 2)
}

# prepare variables.
x <- y <- seq(-1, 1, length = 30)
z <- outer(x, y, cone)
```

```
# plot the 3D surface
# Adding Titles and Labeling Axes to Plot
persp(x, y, z,
main="Perspective Plot of a Cone",
zlab = "Height",
theta = 30, phi = 15,
col = "orange", shade = 0.4)
```

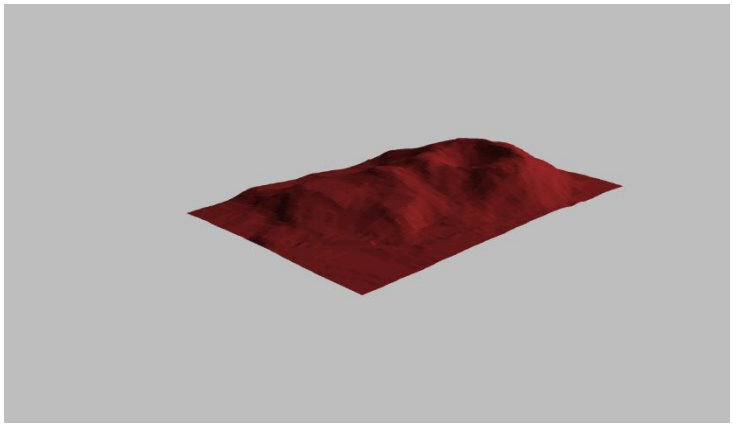
Output:



Visualizing a simple DEM(Digital elevation model)

```
# Visualizing a simple DEM model

z <- 2 * volcano     # Exaggerate the relief
x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)


# Don't draw the grid lines : border = NA
par(bg = "gray")
persp(x, y, z, theta = 135, phi = 30,
      col = "brown", scale = FALSE,
    ltheta = -120, shade = 0.75,
      border = NA, box = FALSE)
```

Output:



Result:

Thus 3D Plot functions in R Programming was executed successfully.

Ex.No:12 **Normal Distribution**

Date:

Aim:

To Write R program for various types of Normal distribution.

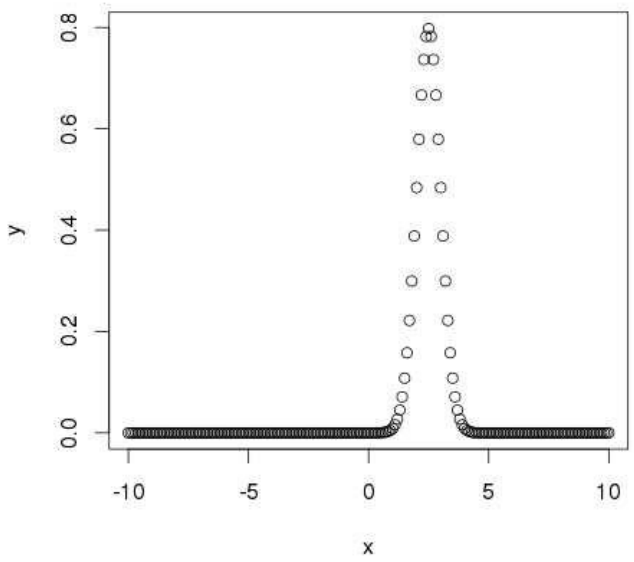
Program:

dnorm():

This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.
x <- seq(-10, 10, by = .1)

# Choose the mean as 2.5 and standard deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)

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

plot(x,y)
```

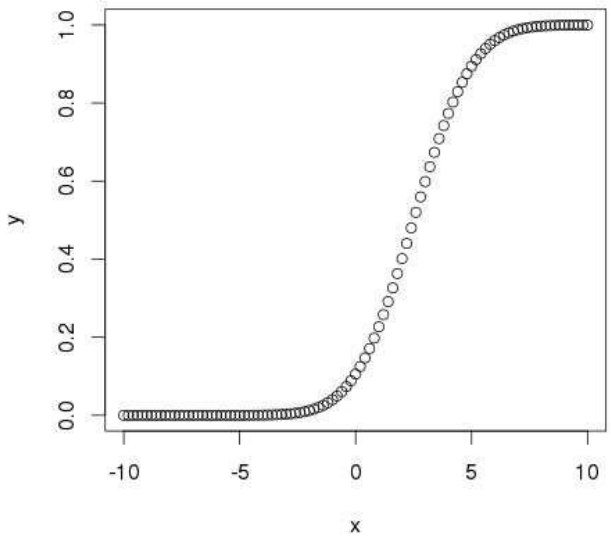
Output:



pnorm():

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.
x <- seq(-10,10,by = .2)

# Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)

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

# Plot the graph.
plot(x,y)
```

Output:

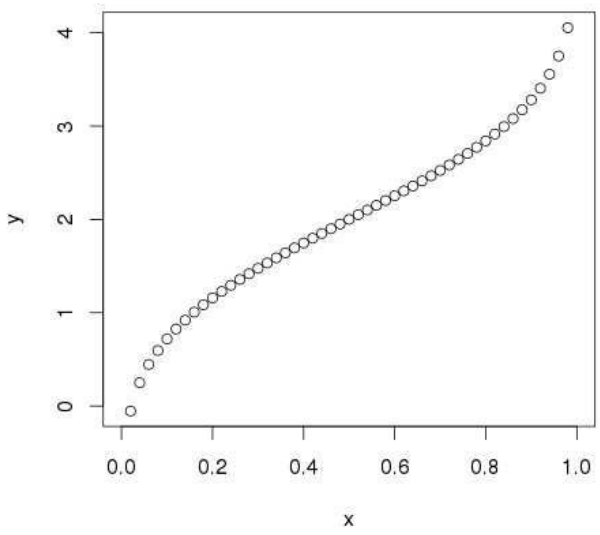


qnorm():

```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)

# Choose the mean as 2 and standard deviation as 3.
```

```
y <- qnorm(x, mean = 2, sd = 1)

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

# Plot the graph.
plot(x,y)
```

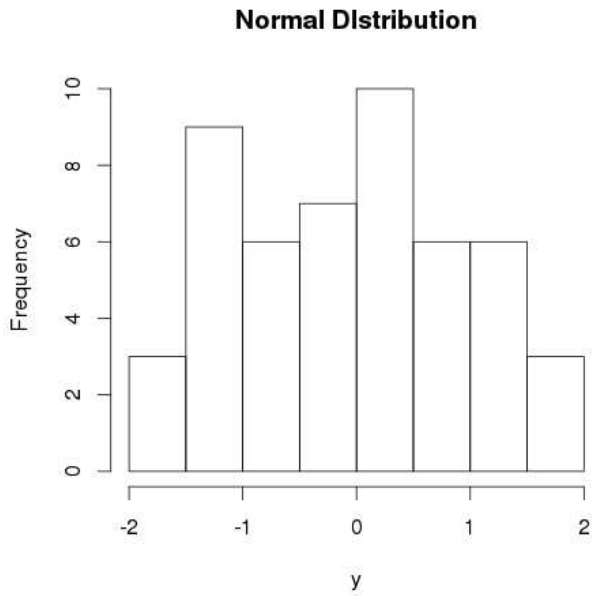
Output:



rnorm()

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)

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

# Plot the histogram for this sample.
hist(y, main = "Normal DIstribution")
```

Output:



Result:

Thus various types of Normal Distribution in R Programming was executed successfully.

Ex.No:13 **Multiple Regression**

Date:

Aim:

To Write R program for Multiple Regression to display Coefficients.

Program:

```
input <- mtcars[,c("mpg","disp","hp","wt")]

print(head(input))
```

```
                   mpg   disp   hp    wt
Mazda RX4         21.0  160    110   2.620
Mazda RX4 Wag      21.0  160    110   2.875
Datsun 710         22.8  108     93   2.320
Hornet 4 Drive      21.4  258    110   3.215
Hornet Sportabout  18.7  360    175   3.440
Valiant             18.1  225    105   3.460
```

```
input <- mtcars[,c("mpg","disp","hp","wt")]

# Create the relationship model.
model <- lm(mpg~disp+hp+wt, data = input)

# Show the model.
print(model)

# Get the Intercept and coefficients as vector elements.
cat("# # # # The Coefficient Values # # # ","\n")

a <- coef(model)[1]
print(a)

Xdisp <- coef(model)[2]
Xhp <- coef(model)[3]
Xwt <- coef(model)[4]
print(Xdisp)
print(Xhp)
print(Xwt)
lm(formula = mpg ~ disp + hp + wt, data = input)
```

Output:

```
Coefficients:
(Intercept)        disp          hp          wt
  37.105505     -0.000937       -0.031157    -3.800891

# # # # The Coefficient Values # # #
(Intercept)
   37.10551
        disp
-0.0009370091
         hp
-0.03115655
       wt
-3.800891
```

Result:

Thus Multiple Regression using in R Programming was executed successfully.

Ex.No:14 **One Way ANOVA**

Date:

Aim:

To Write R program for One Way ANOVA.

Program:

```
data <- PlantGrowth
install.packages("dplyr")
dplyr::sample_n(data, 10)
weight group
1 5.87 trt1
2 4.32 trt1
3 3.59 trt1
4 5.18 ctrl
5 5.14 ctrl
6 4.89 trt1
7 5.12 trt2
8 4.81 trt1
9 4.50 ctrl
10 4.69 trt1

levels(data$group)
[1] "ctrl" "trt1" "trt2"

data$group <- ordered(data$group, levels = c("ctrl", "trt1", "trt2"))
library(dplyr)
group_by(data, group) %>%
summarise(count = n(),mean = mean(weight, na.rm = TRUE),sd = sd(weight, na.rm = TRUE))

group count mean sd
<ord> <int> <dbl> <dbl>
1 ctrl 10 5.03 0.583
2 trt1 10 4.66 0.794
3 trt2 10 5.53 0.443

res.aov <- aov(weight ~ group, data = data)
summary(res.aov)
```

```
Df Sum Sq Mean Sq F value Pr(>F)
group 2 3.766 1.8832 4.846 0.0159 *
Residuals 27 10.492 0.3886
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Result:

Thus One Way ANOVA in R Programming was executed successfully.

Ex.No:15 **Two Way ANOVA**

Date:

Aim:

To Write R program for Two Way ANOVA.

Program:

```
data <- ToothGrowth
install.packages("dplyr")
dplyr::sample_n(data, 5)
len supp dose
1 15.2 OJ 0.5
2 22.5 VC 1.0
3 25.5 OJ 2.0
4 17.3 VC 1.0
5 7.3 VC 0.5
```

Recode the levels after converting the dose to a factor.

```
data$dose <- factor(data$dose,
levels = c(0.5, 1, 2),
labels = c("D0.5", "D1", "D2"))
str(data)
data.frame': 60 obs. of 3 variables:
$ len : num 4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
$ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
$ dose: Factor w/ 3 levels "D0.5","D1","D2": 1 1 1 1 1 1 1 1 1 1 ..
```

Make frequency tables by:

table(data$supp,data$dose)

D0.5 D1 D2

OJ 10 10 10

VC 10 10 10

res.aov2 <- aov(len ~ supp + dose, data = data)

summary(res.aov2)

Df Sum Sq Mean Sq F value Pr(>F)

supp 1 205.4 205.4 11.45 0.0013 **

dose 1 2224.3 2224.3 123.99 6.31e-16 ***

Residuals 57 1022.6 17.9

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1---

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

Result:

Thus Two Way ANOVA in R Programming was executed successfully.