R Programming

→ "R" is an open source programming language used for statistical computing.

→ It is one of the most popular programming languages today.

→ "R" was inspired by S+, it is similar to the "S" programming language. (Seattle washington).

→ "R" Programming language developed by "Ross Ihaka" and "Robert Gentleman" in 1993.

→ It includes machine learning algorithms, linear regression, time series, statistical inference to name a few.

⟹ Data analysis with "R" is done in a series of steps :-

1) Programming - "R" is a clear & accessible programming tool.

2) Transform - "R" is made up of collection of libraries designed specifically for data science.

3) Discover - Investigate the data, refine your hypothesis & analyze them.

4) Model - "R" provides a wide array of tools to capture the right model for your data

5) Communicate - Integrate codes, graphs, & o/p's to a report with "R" mark down

⇒ What is "R" used for?

→ Statistical Inference.

→ Data Analysis.

→ Machine learning algorithm.

⇒ "R" in Industry :-

→ Academic

→ Government.

→ Hospital

→ Healthcare.

→ Consulting

→ Insurance.

→ Energy.

→ Finance.

→ Retail

→ Media.

→ Manufacturing.

→ Tech.

→ Electronics.

⇒ "R" Packages :-

→ dplyr, ggplot 2, data.table, plyr, tidyr, etc.

⇒ Features of "R" :-

→ It is an open source programming language. Hence, you can install "R" for free.

→ Non coders can also understand and perform programming in "R" as it is easy to understand.

→ "R" has various data structures and operators. It can be integrated with other programming language like C, C++, Java, & Python.

→ "R" consists of various inbuilt packages. This makes reporting the results of an analysis easier by using "R".

⇒ Applications of "R" Programming :-

1) R in Research & Academics.
2) R in IT Sector.
3) R in Finance
4) R in E-commerce.
5) R in Social Media.
6) R in Healthcare

7) R in Banking

8) R in Manufacturing

9) R in Government use.

1. "R" in Research and Academics :-

→ "R" is a statistical research tool. It is still used to perform various statistical computations and analysis.

→ Statistical techniques like linear & non-linear modeling, time - series analysis, classification, classical statistic tests, clustering and others all are implemented by R & libraries.

→ "R" is also used for machine learning research & deep learning. other research involving large data sets.

"R" use cases in Research & Academics :-

✓ → Cornell recommends their researchers and students to use "R" for all their researches involving statistical computing.

2. "R" in IT Sector :-

→ In IT companies not only use "R" for their own business intelligence but offer such services to other small, medium & large scale business

→ Some big IT companies that use R :-

- Accenture
- IBM
- Infosys
- Paytm.
- Tata consultancy services
- Wipro.

"R" use cases in IT sector :-

→ Mozilla - Mozilla uses "R" to visualize web activity for their web browser firefox.

→ Microsoft - "R" uses as a statistical engine within the Azure Machine learning frame work. & also x box match making service.

→ Foursquare - Foursquare recommendation engine.

→ Google - To calculate the ROI of their advertising campaigns. to increase the efficiency of online advertising

3. "R" in Finance :-

→ "R" provides an advanced statistical guide for all the financial tasks & computations.

→ Moving averages, auto - regression, time - series analysis, stock - market modelling, financial data mining, down side risk assessment are all easily done through "R" & its libraries.

→ "R" data visualization powers can represent the findings of data analysis in multiple graphical formats like candle stick charts, density plots, and drawdown plots of high quality.

→ Companies like American Express, Bajaj Allianz Insurance, JP Morgan & Standard chartered use "R".

"R" Use cases in Finance :-

→ Lloyds of London.

→ Bajaj Allianz Insurance.

4. "R" in E-Commerce :-

→ E-Commerce companies use R to improve the user's experience on their site as well as marketing and finance purpose, to improve cross-Product selling.

→ Internet-based companies like various e-commerce sites gather and process structured & unstructured data from varying source.

"R" usecases in E-Commerce :-

→ Amazon

→ Flipkart

5. "R" in Social Media :-

→ Social Media companies like facebook use "R" for behavior analysis & sentiment analysis.
(supervised & unsupervised) (emotions)

→ "R" is also used to analyze traffic, user sessions & content, as in an effort to improve user experience.

"R" usecases in Social Media :-

→ Facebook

→ Twitter

6. 'R' in Banking:-

→ It is used for credit risk modeling & other forms of risk analytics.

→ It is also used for fraud detection, mortgage, haircut modeling, stat modeling, volatility modeling, loan stress test simulation, client assessment.

'R' usecase in Banking:-

→ ANZ (Australia & New Zealand).

→ Bank of America.

7. 'R' in Health care:-

→ It is used for drug discovery, bioinformatics, etc.

→ It is used to analyze & predict the spreading of various diseases, for analyzing genetic sequence, to analyze drug safety data.

'R' usecase in Health care:-

→ Merck

8. 'R' in Manufacturing:-

→ They analyze customer feedback to help streamline & improve their products.

'R' usecases in Manufacturing:-

→ Ford Motor Company.

→ John Deere.

9. "R" in Governmental use:-

→ "R" for record-keeping & processing their censues.

→ They also use for essential services like drug regulation, weather forecasting, disaster-impact analysis and much more.

"R" use cases in Governmental:-

→ Food & Drug Administration.

→ National weather sewice.

Installation of "R" & "R" Studio :-

→ Download "R" from

https://cran.us.r-project.org/.

→ Then you will get a windows asking for

Download R for linux.

" " " Mac OS

" " " windows.

→ Click on Download R for windows. you will get a downloaded file as a .exe file. then run it.

↳ Click "Yes"

↳ Select language as "english"

↳ Next. Continue until it complete.

↳ finish.

You tube link for installation of "R"

"Install R & R Studio on windows 7, 8 & 10"

by Michael Galarnyk"

⇒ " R Studio "

" http://www.rstudio.com/products/rstudio/download/

→ click on " RStudio Desktop " You will get a rar file click on it say " Yes ".
↳ Next continue
↳ Finish.

⇒ CRAN - " Comprehensive R Archive Network ".

⇒ Data Types :-

→ Logical - It is a special data type for data with only two possible values which can be constructed as TRUE / FALSE

→ Numeric - Decimal value is called numeric in R, and it is the default computational data type

→ Integer - L tells R to store the value as an integer.

→ Complex - A complex value in R is defined as the pure imaginary value i.

→ character - A character is used to represent string values.

→ Raw - A raw data type is used to holds raw bytes.

Introduction to "R"

⇒ Basic Syntax :-

Ex :- x <- "hello, world"

Print (x).

O/p - [1] "hello, world"

⇒ "R" Objects :-

→ They are classified into :-

(i) Vectors
(ii) Lists
(iii) Matrices
(iv) Arrays
(v) Factors
(vi) Data frames

⇒ Data Types :-

→ They are classified into :-

(i) Logical
(ii) Numeric
(iii) Integer
(iv) Complex
(v) character
(vi) Raw

3. Logical Data Type :-

→ Logical data type which is (True or False)
TRUE or FALSE

Ex:- x <- TRUE.

Class :- Print (class (x)).

O/P - [1] "Logical".

→ So here "R" provides many functions to examine features of vectors & other objects

- class () - what kind of object is it. (high-level).
- type of () - what is the object's data type (low-level).
- length () - how long is it? what about two dimensional object?
- attributes () - does it have any meta data?

Type of ():-

Ex:- x <- TRUE

Print (type of (x))

O/P - [1] "Logical".

Length ():-

Ex:- x <- c ("hello", "hi")

Print (length (x))

O/P - [1] 2.

Ex:- x <- 1

Print (class(x))

Print (typeof(x))

O/p - [1] "numeric"

[1] "double".

(ii) Numeric Data Type :-

→ As Numeric data type it takes number & gives output as a numeric.

Ex:- x <- 20

Print (class(x))

O/p - [1] "numeric"

x <- 20.1

Print (class(x))

Print (typeof(x))

O/p - [1] "numeric"

[1] "double"

(iii) Integer :-

→ Integer data type it takes long integer.

Ex:- x <- 2L

Print (class(x))

O/p - [1] "integer"

x <- 2L

Print (class(x))

Print (typeof(x))

O/p - [1] "integer"

[1] "integer"

(iv) Complex data type :-

→ It checks if it is a complex no. it returns as a complex i.e., 1+2i.

Ex:- `x <- 1+2i`
`Print(class(x))`

O/P - [1] "complex"

`x <- 1i+2`
`Print(class(x))`
`Print(typeof(x))`

O/P - [1] "complex"
[1] "complex"

(v) Character data type :-

→ It shows as a character when we have given an input as a string.

Ex:- `x <- "TRUE"`
`Print(class(x))`

O/P - [1] "character"

`x <- "TRUE"`
`Print(class(x))`
`Print(typeof(x))`

O/P - [1] "character"
[1] "character"

→ There is a difference b/w character & a logic data type in logic data type we will not use double quotes [("TRUE") x we use only TRUE] in character we use double quote ["TRUE"] if we give like this.

Ex:- `x <- TRUE`
`Print(class(x))`

O/P - [1] "logical"

vi) Raw data type :-

→ If we write a word "hello" then it shows output as Raw.

Ex:- x <- charTORaw ("hello")
Print(class(x))
O/P - [1] "raw"

x <- charTORaw("hello")
Print(class(x))
Print(typeof(x))
O/P - [1] "raw"
[1] "raw"

Data Objects :-

i) Vectors :-

→ Vectors can be one of two types :-

- Atomic Vectors.
- Lists.

→ A vector is a collection of elements that are commonly of mode character, logical, integer or numeric.

→ We can create an empty vector with.

Vector() [By default it takes mode as logical]

→ It is more common to use direct constructor

such as character (), numeric etc.

Ex:- 1) Vector ()

o/p - logical (0).

) Vector ("character", length = 5).

o/p - [1] " " " " " " " " " "

) character (5).

o/p - [1] " " " " " " " " " "

) numeric (5)

o/p - [1] 0 0 0 0 0

) logical (5)

o/p - FALSE FALSE FALSE FALSE FALSE

→ we can also create vectors by directly specifying their content.

Ex:- x <- c(1,2,3)

Print (x). // Print (class (x))

o/p - [1] 1 2 3 ↳ o/p - [1] "numeric".

Ex:- x <- c(1L, 2L, 3L).

Print (x)

o/p - [1] 1 2 3

→ Using TRUE, FALSE will create a vector of mode logical.

Ex:- x <- c(TRUE, TRUE, FALSE, FALSE).

Print (x) ⟶ which prints normal assigned values

Print (class(x)) ↓ what is the type of data type which is logical.

→ Create a vector of mode character.

Ex:- x <- c ("abc", "hi", "hello").

Print(x).

Print(class(y)).

O/P - [1] "abc" "hi" "hello".

[1] "character".

Examining Vectors :- z <- c ("abc"; "hi"; "hello"
① ② ③

→ typeof (z). #→ length(z)

O/P - [1] "character". O/P - [1] 3

→ class(z). → str(z)

O/P - [1] "character" O/P - chr [1:3]

"abc", "hi" "hello".

Adding Elements :-

→ The function c() (for combine) can also be used to add elements to a vector.

⇒ x <- c ("abc", "hi", "hello").

x <- c (x, "CSE").

Print (x).

"IT".

O/p - [1] "abc" "hi", "hello" "CSE".

we can also insert another word like, "IT" it displays the o/p as above.

→ If "x" is inserted before CSE the o/p will be displayed in above format like it prints after the i/p statements.

→ If "x" is inserted after CSE then.

x <- c ("CSE", x).

Print (x).

O/p - [1] "CSE". "abc" "hi" "hello".

⇒ Vectors from a sequence numbers :-

→ we can create vectors as a sequence of numbers.

Ex:- x <- 1:10

seq (10)

O/p - [1] 1 2 3 4 5 6 7 8 9 10

→ seq(from=1, to=10, by=0.1).

O/P - [1] 1.0 1.1 1.2 ... 1.9
[16]

Ex:-

x <- c("CSE", "IT", "ECE")

print(x)

print(class(x))

O/P - [1] "ESE" "IT" "ECE"
[1] "character"

(ii) List:-

→ In "R" List act as a containers.

→ The contents of a list are not restricted to a single mode & can encompass any mixture of data types.

→ Lists are some time called as generic vectors.

→ Create list using list()

Ex:- x <- list(1, "a", TRUE, 1+4i).

Print(x).

Print(class(x))

Print(type of(x))

O/P -

```
[[1]]
 [1] 1
[[2]]
 [1] a
[[3]]
 [1] TRUE
[[4]]
 [1] 1+4i
[1] "list"
[1] "list"
```

Ex:-

```
x <- vector("list", length = 5)
length(x)
```

O/P - [1] 5

Ex:-

```
x <- 1:10
x <- as.list(x)
length(x)  ---> O/P -  [1] 10
print(x)   ---> O/P -
[[1]]
 [1] 1
[[2]]
 [1] 2
 :
[[10]]
 [1] 10
```

Ex:-

mlist <- list (a = "hello world", b = 1:10,
data = head(mtcars))

↓ it is default dataset

mlist

$a

O/p - [1] "hello world"

$b

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

$data

car names mpg cyl disp hp drat wt qsec ....

→ **head** - is used to represent the first 10 rows in the data set.

→ **Tail** - is used to represent the last 10 rows in the data set.

iii) Matrices :- → Syntax :-

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

→ A matrix is a two dimensional rectangular data set. It can be created using a vector i/p to the matrix function.

→ **Ex :-** m <- matrix (nrow = 2, ncol = 2)

print(m)

O/p -	[,1]	[,2]
[1,]	NA	NA
[2,]	NA	NA

dim(m)

O/p - [1] 2 2

Ex:- m = matrix(c("a", "b", "c", "c", "b", "a"), nrow = 2, ncol = 3, byrow = TRUE)

print(m)

	[,1]	[,2]	[,3]
[1,]	"a"	"b"	"c"
[2,]	"c"	"b"	"a"

O/P -

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

→ data - data contains the elements in the R matrix.

→ byrow - byrow, is a logical variable. Matrices are by default column-wise. By setting byrow as TRUE, we can arrange data row-wise in the matrix.

→ nrow - Defines the number of rows in the R matrix.

→ ncol - Defines the no. of columns in the R matrix.

→ dimnames - takes two character arrays as I/p for row names & column names

Ex:-

```
mat.data <- c(1,2,3,4,5,6,7,8,9)
rnames <- c("row1", "row2", "row3")
cnames <- c("col1", "col2", "col3")
named.matrix <- matrix(mat.data,
nrow=3, byrow = TRUE, dimnames =
list(rnames, cnames))
named.matrix
```

O/P -

	col1	col2	col3
row1	1	2	3
row2	4	5	6
row3	7	8	9

⇒ Using rbind() & cbind() function - another way of creating an R matrix is to combine vectors as rows or columns using rbind() or cbind()

Ex:-

```
mat.data1 <- c(1,2,3)
mat.data2 <- c(4,5,6)
mat.data3 <- c(7,8,9)
mat <- cbind(mat.data1, mat.data2, mat.data3)
mat
```

O/P -

	mat.data1	mat.data2	mat.data3
[1,]	1	2	3
[2,]	4	5	6
[3,]	7	8	9

⇒ dim :-

→ We can also create an R matrix by changing a vector's dimensions using the dim() function

Ex:- mat <- c(2,3,5,9) → O/P [1] 2 3 5 9

mat

dim(mat) <- c(2,2) → O/P

mat

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

Ex:- mat. data <- c (1,2,3,4,5,6,7,8,9)

mat <- matrix (mat$_{\bullet}$ data, nrow = 3, ncol = 3, byrow = TRUE).

mat.

rownames <- ("r_1", "r_2", "r_3").

colnames <- ("c_1", "c_2", "c_3").

mat.

mat [3, 2]

O/P -

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

	c_1	c_2	c_3
r_1	1	2	3
r_2	4	5	6
r_3	7	8	9

[1] 8

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

mat [c(1,3), c(2,3)].

mat [3, , drop = FALSE] →

	[,1]	[,2]	[,3]
[1,]	7	8	9

iv) Array Data types -

→ Arrays can be any number of dimensions.

Ex:- x <- array (c ("green", "blue"), dim = c(3,3,2))

Print (x).

O/P -

, , 1

	[,1]	[,2]	[,3]
[1,]	G	Y	G
[2,]	Y	G	Y
[3,]	G	Y	G

, , 2

	[,1]	[,2]	[,3]
[1,]	Y	G	Y
[2,]	G	Y	G
[3,]	Y	G	Y

V) Factors Data type :-

→ Factors are r objects. which are created using a vector factors can be created using the factor() function. The nlevels gives the count of levels.

~~Ex:- x <- c("green", Y,~~

~~x <- c("G", "G", "Y", "R", "R", "R",~~

~~"G").~~

Ex:-

```
data = c(1,2,3,2,1,4,2,3)
fdata = factor(data)
fdata
Print(nlevels(fdata))
```

O/p -

```
[1] 1 2 3 2 1 4 2 3
Levels 1 2 3 4
[1] 4
```

(Vi) Data frames :-

→ Data frames are tabular data objects. First column can be numeric while second column can be character & third column can be logical

Ex:-

```
Student <- data.frame (
    S1 = c( 20, 15, 13),
    S2 = c (13, 12, 15),
    S3 = c (21, 22, 20))
Print (Student).
```

O/P -

	S1	S2	S3
1	20	13	21
2	15	12	22
3	13	15	20.

Ex:-

```
Sec <- data.frame (
    gender = c ("Male", "female", "male"),
    height = c (152, 171.5, 165),
    weight = c (81, 90, 78),
    Age    = c (42, 45, 50))
Print (Sec).
```

O/p -

	gender	height	weight	Age.
1	Male	152	81	42
2	female	171.5	90	45
3	Male	165	78	50.

⇒ Variables :-

→ A variable provide us with named storage that our programs can manipulate.

Variable Assignment :-

Ex:- `Var1 <- c(0,1,3,2)`

`Var2 <- c("Learn", "R")`

`Var3 <- c(TRUE, 1)`

`Print(Var1)`

`Cat("Var1 is", Var1, "\n")`

`Cat("Var2 is", Var2, "\n")`

`Cat("Var3 is", Var3, "\n")`

O/P — `[1] 0 1 3 2`

`Var1 is 0 1 3 2`

`Var2 is Learn R`

`Var3 is 1 1`

⇒ The cat function combines multiple items into a continuous print O/P.

Ex:-

`Var1 <- "hello"`

`cat("The class of var1 is", class(var1))`

O/P - The class of var1 is character.

⇒ Data type of a Variable :-

Ex:-

```
var2 <- "hello"
cat ("The class of var2 is", class(var2), "\n")
```

O/P - The class of var2 is character.

Finding variables :-

```
Print(ls())
Print(ls(pattern = "var"))
Print(ls(all.name = TRUE))
```

Deleting variables :-

```
rm()
rm(var1,3)
rm(list = ls())
Print(ls())
```

Types of operators :-

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Miscellaneous operators.

⇒ Decision making :-

→ It is to specify one or more conditions to be evaluated or tested by the program., along with a statement or statements to be executed

→ It has two conditions true (or) false.

→ Decision making is of if, if else, switch statements.

⇒ Loops :-

→ A loop statement allows us to execute a statement (or) group of statements multiple times.

→ repeat loop, while loop, for loop., break, & next statement.

Unit - I

⇒ Function :-

→ A function is a set of statements organized together to perform a specific task.

→ R has a large number of in-built function

→ An R function is created by using the key word function.

Syntax :-

```
function_name <- function(arg1, arg2, ...)
{
function body
}
```

⇒ Function Components :-

→ The different parts of function are :-

→ Function Name :- This is the actual name of the function. It is stored in R environment as an object with this name.

→ Arguments :- when a function is invoked, we pass a value to the argument.

- arguments are optional, function may contain no argument, arguments can have default values.

→ function Body :- It is a collection of statements that defines what the function does

→ Return value :- The return value of a function is the last expression in the function body to be evaluated.

#②

⇒ Built-in-function :-

→ Seq()

→ mean()

→ max()

→ sum(x)

→ Paste() etc.

Ex :- print(seq(32, 44))
print(mean(25:82))
print(sum(41:68))

⇒ User Defined Function :-

```
→ new.function <- function(a)
  {
  for(i in 1:a)
  {
   b <- i^2
   print(b)
  }
  }
   new.function(10).
```

⇒ Calling a function :-

```
→ new.function <- function(a)
  {
  for(i in 1:a)
  {
```

```
    b <- i^2
    print(b)
}
}
new.function(6)
```

O/p -
```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36.
```

⇒ calling a function without argument :-

→
```
new.function <- function()
{
for(i in 1:5)
{
print(i^2)
}
}
new function()
```

O/p -
```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

Cont....

⇒ Calling a function with Argument values (by position & by name).

→
```
new.function <- function (a, b, c).
{
result <- a * b + c
Print (result)
}
new . function (5, 3, 11)
new. function (a = 11, b = 5, c = 3).
```

O/P - [1] 26
[1] 58.

⇒ Calling a function with default argument:

→
```
new. function <- function (a = 3, b = 6)
{
result <- a * b
Print (result)
}
new . function ()
new. function (9,5).
```

O/P - [1] 18
[1] 45

⇒ Lazy Evaluation of function:-

→ new. function <- function (a, b)

{

print (a^2)

Print (a)

Print (b) }

new. function (6).

O/P - [1] 36

[1] 6

Print(b) : argument "b" is missing

value of b is not declared. If we want to declare the value of b then.

new. function (6, 10).

O/P - [1] 36

[1] 6

[1] 10

①

⇒ Function Types:-

→ R also has two types of functions.

→ Built in function

→ user defined function

→ There are lots of built in functions which we can directly call in the pgm without defining them.

→ R allows us to create our own functions

⇒ Built in function :-

Ex:- Print (seq (32, 46))

Print (mean (22:80))

Print (sum (41:70)).

O/p :- [1] 32 33 34 46

[1] 51

[1] 1665.

⇒ User-Defined function:-

Ex:- new.function <- function()

{

for (i in 1:5)

{

Print (i^2)

}

}

new.function()

O/p :- [1] 1

[1] 4

[1] 9

[1] 16

[1] 25

R Built in Functions :-

- Math function — abs(x), sqrt(x), log(x)
- String function — paste(... sep=" "), tolower(x), toupper(x)
- Statistical probability function — pbinom(q, size, prob)
  qbinom(" " ")
  rnorm(n, m=0, sd=1)
- Other statistical function.

```
mean(a, trim=0, na.rm = FALSE)
sd(x)
median(x).
```

Preview of some important R data structures.

- Same as we defined before like Data types & Data objects.

Regression Analysis :-

- Regression analysis is a group of statistical Process used in R programming & statistics to determine the relationship b/w dataset variables.
- Generally this regression analysis is used to determine the relationship b/w the dependent & independent variables of the dataset.

→ Regression analysis helps to understand how dependent variables changes and other independent variable is changes to when one of the other independent variables are kept constant.

→ This helps in building a regression model and further helps in forecasting the values w.r.t a change in one independent variables

→ There are 4 types of regression analysis techniques :-

1) Linear Regression
2) Logistic Regression
3) Multinominal Logistic Regression
4) Ordinal Logistic Regression.

1) Linear Regression :-

→ Linear regression is one of the most widely used regression techniques to model the relationship b/w two variables.

→ It uses a linear relationship to model the regression line.

→ There are 2 variables used in the linear relationship equation

- predictor variable
- response variable.

$$y = ax + b.$$

where,

y is the response variable

x is the predictor variable.

a & b are the coefficients.

→ The regression line created using this technique is a straight line.

→ The response variable is derived from Predictor variable.

→ Predictor variables are estimated using some statistical experiments.

→ Linear regressions are widely used but these techniques is not capable of Predicting probability

⇒ The steps to create a relationship.

- Carry out the experiment of gathering a sample of observed values of height & weight.
- for creating relationship model using lm() function in R.
- Finding the coefficients from the model

⇒ The lm() function in linear regression is

lm (formula, data).

formula - is a symbol presenting relation b/w x & y.

data - is the vector on which the formula will be applied.

⇒ lm - is a linear model.

Ex:-

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

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

relation <- lm (y ~ x).

Print (relation).

Call:
lm (formula = y ~ x)

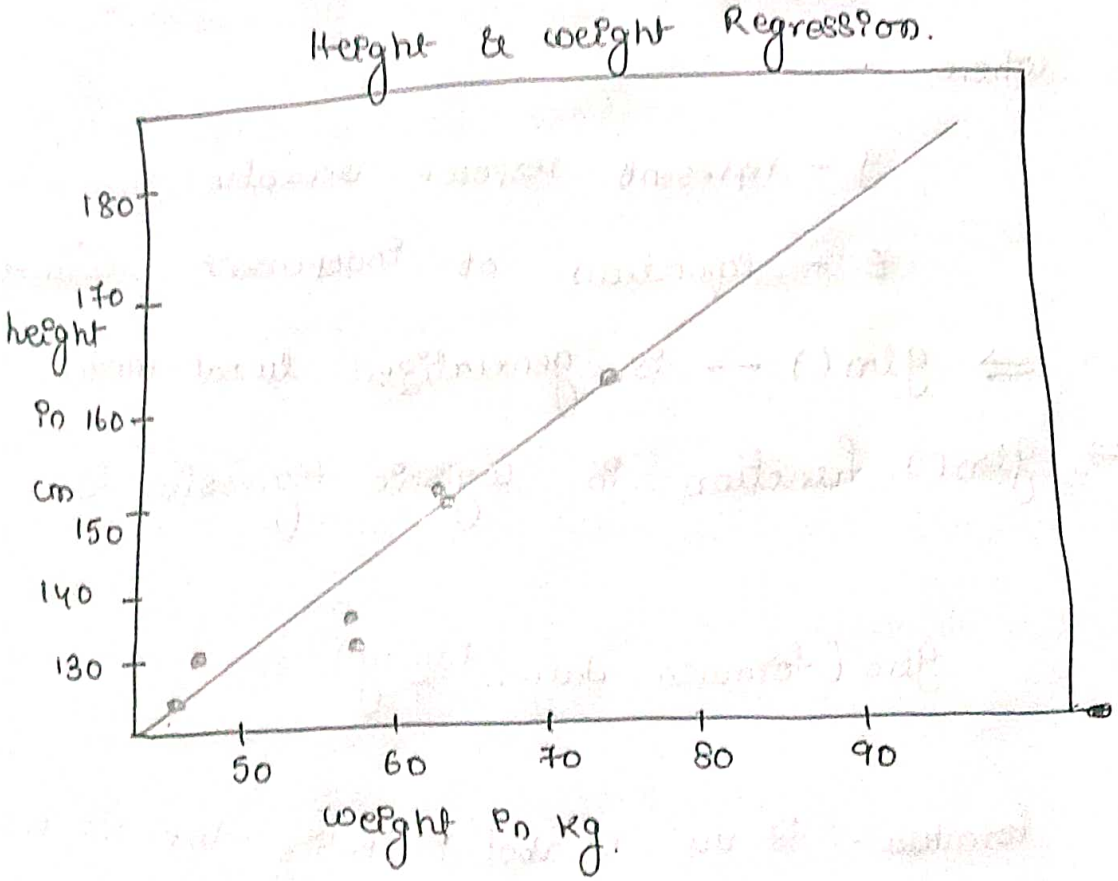
o/p - Coefficients

(Intercept)	x
-38.4551	0.6746

Ex:-

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 181)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x).
png(file = "linear regression.png").
plot(y, x, col = "blue", main = "Height & weight
Regression", abline(lm(x~y)), cex = 1.3, pch=16, xlab = "weight in kg", ylab = "Height in
cm")
dev.off().
```

default cex is 1 ↓ for scaling. (1.5)



Pch - plot character. like (● □ △)

xlab, ylab are titles.

Png - plot graphically.

abline - used to add one or more straight lines to a graph

2) Logistic Regression:- (LR)

→ Logistic regression has an advantage over linear regression as it is capable of predicting the values with in the range.

→ Logistic regression is used to predict the values with in the categorical range.

$$y = \frac{1}{1 + e^{-z}}$$

where

y - represents response variable

z - equation of independent variables

⇒ glm() — is generalized linear model.

⇒ glm() function in logistic regression is

glm (formula, data, family).

formula - is the symbol presenting the r/s b/w the variables

data - dataset giving the values of these variables

family - is binomial for LR

Ex:-

```
input <- mtcars[, c("am", "cyl", "hp", "wt")]
print(head(input))
```

O/P

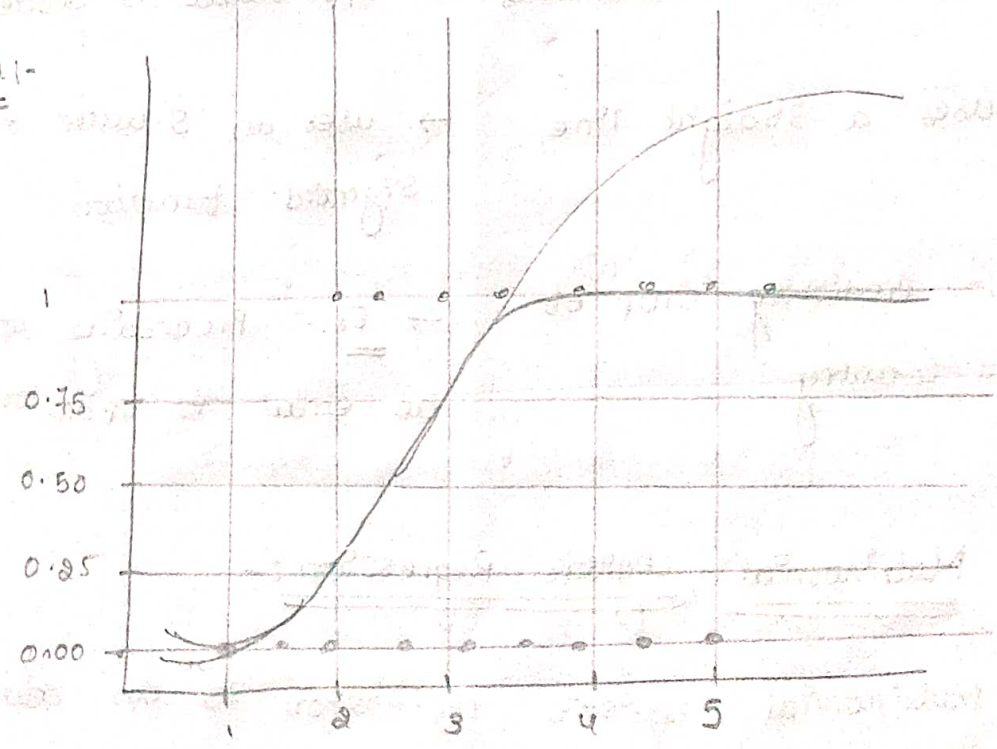
car names	am	cyl	hp	wt
	1	6	110	2.620
	⋮	⋮	⋮	⋮

Ex:-

```
input <- mtcars[, c("am", "cyl", "hp", "wt")].
am.data = glm(formula = am ~ cyl + hp + wt,
    data = input, family = binomial).
print(summary(am.data)).
```

O/P :-

Ex:-



Linear Regression	Logistic Regression
→ A linear regression is the i/p b/w a dependent variable & one or more independent variables	→ A statistical model that predicts the probability of an outcome that can only have 2 values.
→ Used to solve regression Problems	→ used to solve classification problems.
→ Estimate the dependent variable when there is a change in the independent variable	→ calculates the possibility of an event occuring.
→ O/p value is continuous	→ O/p value is discrete
→ uses a straight line	→ uses an S curve or sigmoid function.
→ Ex:- Predicting GDP of a country	→ Ex:- Predicting whether an email is spam (or) not

3) Multinomial Logistic Regression :-

→ Multinomial logistic regression is an advanced technique of logistic regression which takes more than 2 categorical variables unlike

logistic regression which takes a categorical variable.

Ex:- In species they found a new variety of species so as to determine many factors like size, shape, eye color, the environmental factor of its living, etc.

→ In multiple regression we have more than one predictor variable & one response variable.

$$y = a + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

y - response variable.

$a, b_1, b_2 \cdots b_n$ - coefficients.

$x_1, x_2 \cdots x_n$ - predictor variables.

→ lm() function creates the relationship model between the predictor & response variable.

→ lm($y \sim x_1 + x_2 + x_3 + \cdots$, data).

→ formula - The relation b/w the response variable & predictor variable.

data - vector on which the formula will be applied.

Ex:-

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

o/p -

→ create relationship model & get coefficients

Ex:-

```
input <- mtcars [, c("mpg", "disp", "hp", "wt")]
model <- lm (mpg ~ disp + hp + wt, data = input)
Print (model).
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).
```

Op :-

→ Create equation for regression model.

Ex:-

$$y = a + x\,disp \cdot x_1 + xhp \cdot x_2 + xwt \cdot x_3$$

(or)

$$y = 37.15 + (-0.000937) * x_1 + (-0.0311) * x_2 + (-3.8008) * x_3$$

$$= 37.15 + (-0.000937) * 221 + (-0.0311) * 102 + (-3.8008) * 2.91 = 22.7$$

iv) Ordinal Logistic Regression :-

→ Ordinal logistic regression is also an extension to logistic regression.

→ It is used to predict the values as different levels of category

⇒ Implementation of logistic Regression in R programming.

→ glm() function.

→ Ex:- Syntax :-

glm(formula, family = binomial)

formula - represents an equation on the basis of which model has to be fitted.

family - type of function to be used (binomial for logistic regression)

→ help ("glm").

Regression Analysis for Exam Grade:-

⇒ rnorm (number, mean, variance)

→ number - total no. of students

→ mean - By default it takes "0"

→ variance - By default it takes "1".

⇒ Program:-

→ # generate random sub marks with mean = 20, & variance = 2.

sub <- rnorm (40, 20, 2).

→ # Sorting sub marks in ascending order.

sub <- sort (sub).

→ # Generate vector with pass & fail of values of 40 students

result <- c (0,0,0,1, 0, 0,0,0,0,1, 1,0,0,0,
1,1,0,0,1,0,0,0,1,0,0,1,1,0,1,1,
1,1,1,0,1,1,1,1,0,1).

→ # Data frame.

df <- as. data. frame (cbind (sub, result))

→ # Print data frame.

Print (df)

Continue.

→ # output to be present as PNG file.

png (file = "Logistic Regression GFG.png")

→ # plotting sub on x-axis & result on y-axis

plot (Sub, result, xlab = "subject marks",
ylab = "probability of passing")

→ # create a logistic model.

g = glm (result ~ Sub, family = binomial, df)

→ # create a curve based on prediction using the regression model

curve (predict (g, data.frame (Sub = x), type = "resp"), add = TRUE)

→ # This Draws a set of points.

→ # Based on fit to the regression model.

points (Sub, fitted (g), pch = 25)

→ # Summary of the regression model.

Summary (g).

→ # Saving the file

dev.off()

→ R if Statement :-

→ The if statement consists of Boolean expression followed by one or more statements.

→ The if statement is the simplest decision making stmt which helps us to take a decision on the basis of the condition.

→ If the condition is true then the statement will be executed.

Syntax:-

```
if (boolean expression)
{
if condition is true then stmt will be
executed.
}
```

Ex:- 1) To perform it is integer or not.

```
x <- 24L
y <- "CSE"
if (is.integer(x))
{
Print ( x, " is an integer")
cat
}
```

o/p — 24 is an integer.

Ex-2:-

```
x <- 20
y <- 24
count = 0
if (x<y)
{
cat (x, "is a smaller number \n")
count = 1
}
if (count == 1)
{
cat ("Block successfully execute")
}
```

O/p - 20 is a smaller number.
Block successfully execute.

⇒ If - else Statement :-

→ In the if Statement., the inner code is execute when the condition is true. The code which is outside the if block will be executed when the if condition is false.

→ R Programming treats any non-zero & non-null values as true, & if the value is either zero or null, then it treats them as false.

Syntax:-

```
if (boolean expression)
{
 stmt (s) will be executed if the boolean
 expression is true.
}
else
{
stmt (s) will be executed if the boolean
expression is false.
}
```

Ex-1:-

```
x <- c ("hi", "hello", "welcome", "to",
        "cse").

if ("cse" %in% x)
{
 print ("cse is found")
}
else
{
print ("cse is not found").
}
```

O/P:- [1] "cse is found".

Ex-2:-

```
a <- 'u'
if (a == 'a' || a == 'e' || a == 'i' || a == 'o' ||
a == 'u' || a == 'A' || a == 'E' || a == 'I' ||
                    a == 'O' || a == 'U')
```

```
{
cat (" character is a vowel \n")  }
else
{
cat (" character is a consonant ")
}
cat (" character is = "; a)
}
```

o/p :- character is a vowel
character is = u.

⇒ R else if statement :-

→ This statement is also known as nested-if else statement

→ The if statement is followed by an optional else if .... else statement.

syntax :-

```
if (boolean expression1)
{
This block executes when the boolean
expression 1 is true.
}
else if (boolean expression 2)
{
---          2 is true.
}
else if (b          e          3)
```

{

be 3 is true.

this

}

else

{

this block executes when none of the above condition is true.

}

Ex-1:-

```
age <- readline (prompt = " enter age: ")
age <- as.integer (age).
if (age< 18)
Print ( " you are a child")
else if (age >30)
Print (" you are old ")
else
  Print ( " you are adult").
```

⇒ R Switch Statement :-

→ A switch stmt is a selection control mechanism that allows the value of an expression to change the control flow of a program.

Syntax:-

```
Switch (expression , case1 , case 2, -----)
```

Ex:- x <- switch(3,
"Hi",
"Hello",
"CSE",
"DS")
Print(x)

Op - [1] "CSE"

Ex:- Val1 = 6
Val2 = 7
Val3 = 'S'
result = switch(Val3,
'a' = cat('Addition = ', Val1 + Val2),
'b' = cat('Subtraction = ', Val1 - Val2),
'c' = cat('division = ', Val1 / Val2),
's' = cat('multiplication = ', Val1 * Val2))
Print(result).

Op - [1] Multiplication = 42 NULL

⇒ R Next Statement :-

Ex:-

```
a <- 1
while (a <= 10)
{
if (a == 5)
next
Print(a)
a = a+1
}
```

O/P :-

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

⇒ Break statement :-

Ex:-

```
a <- 1
repeat
{
Print("hello").
if (a >= 5)
break
a <- a+1
}
```

⇒ For loop :-

Ex:-

```
fruit <- c('Apple', 'Banana', 'Pineapple',
           'Grapes', 'mango')

for (i in fruits)
{
Print(i)
}
```

O/P -

```
[1] "Apple"
[1] "Banana"
[1] "Pineapple"
[1] "Grapes"
[1] "mango"
```

⇒ R Repeat loop:-

Ex:-

```
V - c ("hello", "repeat", "loop")
count <- 2
repeat
{
Print(V)
count <- count +1
if (count > 5)
{
 break
}
}
```

⇒ while loop:-

Ex:-

```
V <-c ("hello", "cse", "DS")
cnt <-2
while (cnt < 7)
{
 Print(V)
cnt = cnt +1
}
```

o/p -

Start up & shut Down :-

→ R profile located either in your home directory (or) in the directory from which you are running

→ The latter directory is searched for this file first, which allows you to have custom profiles for particular projects.

→ To set the text editor that R invokes if you call edit(), you can use a line in.

• Rprofile like this.

```
options(editor = "/usr/bin/vim")
```

→ R's option() function is used for configuration that is tweak various settings.

```
.libPaths("/home/nm/R")
```

→ This automatically adds a directory that contains all my auxiliary packages to my R search to path.

```
>getwd()
```

→ By the above command we can check current directory by typing "getwd()"

→ we can change our working directory by calling setwd() with the desired direct

>setwd("a/")

→ we can find more information about start files by

>?Startup

⇒ Getting Help :-

→ To get online help invoke help().

>help()

→ To get information on the seq() function.

>help(seq).

→ The shortcut to help() is a question mark

>?seq.

→ Special characters and some reserved words must be quoted when used with the help() function.

> ? "<".

for instance to get help on the "<" operator.

→ And if we want to see what online manual has to say about for loop.

> ? "for".

⇒ The example of function :-

→ seq(0, 1, length.out = 11)

o/p :- [1] 0.0 0.1 0.2 0.3 ---------- 1.0

→ seq(stats::rnorm(20))

[1] 1 2 3 ---------- 20.

→ seq(1, 9, by = 2)

[1] 1 3 5 7 9

→ seq(1, 9, by = pi)

[1] 1.000000 4.141593 7.283185

→ seq(1,6, by=3)

[1] 14

→ seq(1.575, 5.125, by = 0.5)

[1] 1.575 1.625 1.725 . . . . . 2.125

[13] 2.175

[25] 2.775

[37] 3.375

[49] 3.975

[61] 4.575

→ seq(17). (or) seq → 1:17

[1] 1 2 3 4 5 17

→ example(persp).

= for displaying a sample graph we use (persp) we will be getting different different graph by pressing enter.

→ ?? 'multivariate normal' — this is used to search the ~~multivariate~~ multivariate normal it displays all of its matching and different forms of multivariate normal.

Vectors:-

Vector:-

→ A vector is a collection of elements which is mostly commonly of mode character, integer, logical, or numeric.

→
```
x <- c(12,13,14,16).
x <- c(x[1:3], 15, x[4]).
x
```

O/P [1] 12 13 14 15 16.

→ How to find the length of a vector.

```
x <- c(1,2,3)
length(x)
```

[1] 3.

→ `m <- matrix(c(1,2,3,4), nrow=2, ncol=2).`

m	[,1]	[,2]
[1,]	1	3
[2,]	2	4

→ `> m + 10:13`

	[,1]	[,2]
[1,]	11	15
[2,]	13	17

→ Creating Instance y for two elements.

y <- vector(length=2). → o/p as logical default

(or)

y <-c(5,12) → [1] 5 12.

y[1] <- 5
y[2] <- 12. } this can't work

⇒ Recycling :-

→ c(1,2,4) + c(6,0,9,20,22).

And with the o/p we will get a warning

[1] 7 2 13 21 24

⇒ m <- matrix(c(1,2,3,4,5,6), nrow=3, ncol=2)

m.

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

⇒ m + c(1,2)

	[,1]	[,2]
[1,]	2	6
[2,]	4	6
[3,]	4	8

Common vector operations :-

→ 2 + 3

[1] 5

→ x <- c(1,2,4)

x + c(5,0,-1)

[1] 6 2 3

→ x <- c(1,2,4)

x/c(5,4,-1)

[1] 0.2 0.5 -4.0.

→ u + 2 (2,3)

[1] 5.

→ 2 * c(5,0,-1)

[1] 5 0 -4

→ x <- c(1,2,4)

t <- c(5,4,-1)

~~x %*% c(5,4,-1)~~

print(x.t)

[1] 1 2 0.

Vector Indexing :-

→ y <- c(1.2, 3.9, 0.4, 0.12)

y [c(1,3)]

[1] 1.2 0.4

→ y[2:3]

[1] 3.9 0.4

→ v <- 3:4

y[v]

[1] 0.40 0.12

→ x <- c(4,2,17,5)

y <- x[c(1,1,3)]

y

[1] 4 4 17.

→ z <- c(8,12,13)

z[-1]

[1] 12 13

→ z[-1:-2]

[1] 13

→ z <- c(5,12,13)

z[1:(length(z)-1)] (or)

[1] 5 12.

→ z[-length(z)]

[1] 5 12

→ Generating useful vectors with operator.

→ 5:8

[1] 5 6 7 8

→ 5:1

[1] 5 4 3 2 1

→ ?<-2

1:?-1

[1] 0 1

→ 1:(?-1)

[1] 1

⇒ Repeating vector constants with rep()

→ z <- rep(8,4)

z

[1] 8 8 8 8

→ rep(c(5,12,13),3)

[1] 5 12 13 5 12 13 5 12

→ rep(1:3,2)

[1] 1 2 3 1 2 3

→ rep(c[5,12,13), each=2)

[1] 5 5 12 12 13 13

→ using all() and any()

→ x <- 1:10 → all (x > 88) → all(x > 0)

any (x > 8) [1] FALSE [1] TRUE.

[1] TRUE

Vectorized operations:-

→ operations :- applying +, *, >, & transcedental functions - Square roots, logs, trig functions etc.

a) vector in, vector out :-

1) u <- c (5, 2, 8)
v <- c (1, 3, 9)
u > v

o/p :- [1] TRUE FALSE FALSE.

2) w <- function (x) return (x + 1) x <- c(5,2,8)
> w(x)

o/p :- [1] 6 3 9

3) f <- function (x, c)
return ((x + c) ^ 2)
f(1:3, 0)

⇒ Vector in, matrix out

1)
```
y <- function (z)
return (c( z, z^2))
z <- 1:8
y(z)
```

matrix as o/p

```
y <- function (z).
return (c, (z, z^2))
z <- 1:8
y(z)
```

Not needed

```
matrix (y(z), ncol = 2).
```

	[,1]	[,2]
[1,]	1	1
[2,]	2	4
[3,]	3	9
[4,]	4	16
[5,]	5	25
[6,]	6	36
[7,]	7	49
[8,]	8	64

⇒ using sapply() (or simplify apply):-

→ sapply (x, f) applies the function() to each.

element of x & then converts the result to a matrix.

Ex:-

```
y <- function(z)
return(c(z, z^2))
sapply(1:8, z^2)
```

```
y <- function(z)
return(c(z, z^2))
v <- sapply(z, function(x) x^2)
```

z	1	2	3	4	5	6	7	8
[1,]	1	4	9	16	25	36	49	64

O/P:-

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
[1,]	1	2	3	4	5	6	7	8
[2,]	1	4	9	16	25	36	49	64

⇒ NA and NULL values :-

→ 1) x <- c(88, NA, 12, 168, 13)

x

O/P :- [1] 88 NA 12 168 13.

2) mean(x)

x contains a NA value.

[1] NA

3) mean(x, na.rm = T) # here na value is taken as TRUE.

O/P :- [1] 70.25.

4) Same example with null :-

```
x <- c(88, NULL, 12, 168, 13)
mean(x)
```

o/p :- [1] 70.25

→ There are multiple NA values, one for each mode.

1)
```
x <- c(5, NA, 12)
mode(x[1])
```
[1] "numeric".

2)
```
y <- c("abc", "def", NA)
mode(y[2])
```
[1] "character".

⇒ Using NULL :-

1)
```
z <- NULL
for (i in 1:10)
if (i %% 2 == 0)
z <- c(z, i)
z
```
o/p :- [1] 2 4 6 8 10

2)
```
seq(2, 10, 2)
```
[1] 2 4 6 8 10

4) 2*1:5

o/p - [1] 2 4 6 8 10.

5) U <- NULL

length (u)

o/p - [1] 0

6) V <- NA

length (v)

o/p - [1] 1

⇒ filtering :-

→ z <- c (5, 2, -3, 8)

w <- z [z * z > 8].

w

[1] 5 -3 8

→ Replace all elements larger than a 3 with a 0.

x <- c(2, 4, 5)

x [x > 3] <- 0

x

⇒ filtering with the subset() function :-

→ using subset() can exclude NA values

→ filtering can also be done with the subset function.

1) x <- c(6, 1:3, NA, 12)

x[x>5]

[1] 6 NA 12

2) subset(x, x>5)

[1] 6 12.

⇒ Selection function which():-

→ if we want to find position of z.

z <- c(5, 2, -3, 8)

which(z * z > 8)

o/p - [1] 1 3 4

⇒ Vectorized if-then-else :-

x <- 1:10

y <- ifelse(x %% 2 == 0, 5, 12)

y

[1] 12 5 12 5 12 5 12 5 12 5

Using any() & all() :-

→ any() - This function in "R" checks whether at least one value remains true of the logical vector.

→ all() - This function in "R" checks whether all values remains true of the logical vector.

Example of any function in "R" :-

1) `v1 <- c(3,4)`
`v2 <- c(101,4)`
`t <- c(1,2,3,4,5,6,7,8)`
`any(v1 %in% t)`
O/P - [1] TRUE

`any(v2 %in% t)`
O/P - [1] TRUE.

2) `x <- 1:10`
`any(x > 8)`
O/P - [1] TRUE

3) `any(mtcars$cyl %in% 4)`
O/P - [11] TRUE

Example of all function in "R" :-

1) `all(v2 %in% t)`
O/P - [1] FALSE

2) `all(x > 88)`
[1] FALSE

3) `all(x > 0)`
[1] TRUE

4) `all(mtcars$cyl %in% 4)`
O/P - [1] FALSE

8) all (returns $ any %in% c(1,0))

o/p - [1] TRUE

⇒ x1 <- c(1, 5, 3, -3, 5, -7, 8)

x1

o/p - [1] 1 5 3 -3 5 -7 8

⇒ all (x1 < 0)

[1] FALSE

⇒ any (x1 < 0)

[1] TRUE

⇒ x2 <- c(x1, NA)

x2

o/p - [1] 1 5 3 -3 5 -7 8 NA

⇒ all (x2 > -10)

[1] NA

full code

[1] TRUE

x1 <- c(1, 5, 3, -3, 5, -7, 8)

x2 <- c(x1, 10)

⇒ all (x2 > -10, na.rm = TRUE)

[1] TRUE

in order to exclude NA value from our calculation.

⇒ any (x2 < -10, na.rm = TRUE)

[1] FALSE

↓

This contains no smaller value than 10

⇒ all (is.na(x2))

[1] FALSE

⇒ any (is.na(x2))

[1] TRUE

⇒ If we write x2 <- c(x1, NA)

Apply function :-

→ The apply() family belongs to R base package & is populated with functions to manipulate slices of data from matrices, arrays, lists & data frames.

→ This avoid explicit use of loop constructs.

→ They act on an input list, matrix or array & apply a named function with one or several optional arguments.

⇒ The called function could be :-

- An aggregating function like mean, or sum.
- other transforming or subsetting functions.
- other vectorized functions, which yield more complex structures like lists, vectors, matrices, & arrays.

→ the apply() functions form the basis of more complex combinations & helps to perform operations with a very few lines of codes.

→ the family is made up of the apply(), lapply(), sapply(), vapply(), mapply(), rapply(), & tapply() functions.

1) apply()

→ apply (x, margin, fun, ...)

↳ All should be capital letters

where

→ x - is an array or a matrix if the dimension of array is 2.

→ margin - defines how the function is applied.

If it is 1 then it applies over rows,

2 then columns.

Margin = c(1,2). It applies to both rows & columns.

→ FUN - which is the function that you want to apply to the data. It can be any R function including user defined function (UDF)

Ex:-

m = matrix (c(1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3), nrow = 5, ncol=

m → it prints 5×6 matrix as a matrix.

(or)

x <- matrix (rnorm (30), nrow = 5, ncol = 6)

apply (x, 2, sum) (or) if we take first one

apply (m, 2, sum).

It is going to add nroco wise & apply the o/p

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	1	6	2	7	3	8
[2,]	2	7	3	8	4	9
[3,]	3	8	4	9	5	1
[4,]	4	9	5	1	6	2
[5,]	5	1	6	2	7	3
	15	31	20	27	25	23

[1] 15 31 20 27 25 23

2) The lapply() function :-

→ If we want to apply a given function to every element of a list & obtain a list as a result.

→ If we want to know about apply() function then ?apply().

Ex:-

```
a = matrix (c(1,2,3,4,5,6,7,8,9), nroco = 3, ncol = 3)
a
```

```
  1  4  7
  2  5  8
  3  6  9
```

b = matrix (c (1,2,3,4,5,6,7,8,9), nrow = 3, ncol

b

1	4	7
2	5	8
3	6	9

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

c

1	4	7
2	5	8
3	6	9

l <- list (a,b,c)

lapply (l, "[", , 2)

[[1]]

[1] 4 5 6

[[2]]

[1] 4 5 6

[[3]]

[1] 4 5 6

lapply (l, "[", 1,)

[[1]]

[1] 1 4 7

[[2]]

[1] 1 4 7

[[3]]

[1] 1 4 7

"[" — Selection operator

We can't replace "[" this & operate in different way

→ The "[" notation is the select operator. It is to extract all the elements of 3rd line of b matrix

```
b[3,];        o/p - 3 6 9
```

→ If we want to execute a single element for each matrix

```
lapply(l, "[", 1, 2)

[[1]]
[1] 4

[[2]]
[1] 4

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

→ lapply(l, "[", 1,)

```
[[1]]          [[2]]          [[3]]
[1] 1 4 7      [1] 1 4 7      [1] 1 4 7
```

3. Sapply() function :-

→ The sapply() function works like lapply() which is going to try to simplify the o/p. Sapply() is a wrapper function for lapply()

→ lapply (l, "[", 2, 1)

[[1]] a
[1] 2
[[2]] b
[1] 2
[[3]] c
[1] 2

Note:-

a, b, c

$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$.

→ Sapply (l, "[", 2, 1)

[1] 2 2 2

→ Sapply (l, "[", 2, 1, simplify = F)

⇒ Applying the lapply() function would give us a list unless we pass simplify = FALSE as a parameter to sapply()

[1] 2 2 2

→ unlist (lapply (1, "[", 3, 2))

[1] 6 6 6

⇒ The rep() function :-

→ rep() - replicates

```
z <- sapply(1, "c", 1, 1)
z
[1] 1 1 1
```

[1 values of list is initialized to before pages]

```
z <- rep(z, c(3,1,2))
z
[1] 1 1 1 1 1 1
```

⇒ c(3,1,2) 3 times the first, one time the second & two times the third.

1. mapply() function :-

→ mapply() is a multivariate, apply. In short mapply() applies a function to multiple list or multiple vector arguments.

→ `q <- matrix(c(rep(1,4), rep(2,4), rep(3,4), rep(4,4), 4, 4)`

q

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

(or)

```
a1 <- mapply (rep, 1:4, 4)
a1
```

```
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

⇒ Functions related to apply() :-

→ Sweep() function :-

```
b-means <- apply (b, 2, mean)
b-sdev <- apply (b, 2, sd)
b-trans1 <- sweep (b, 2, b-means, "-")
Print (b-trans1)
```

```
0 0 0
0 0 0
0 0 0
```

```
b-trans1
```

```
0 0 0
0 0 0
0 0 0
```

```
b-trans2 <- sweep (b-trans1, 2, b-sdev, "/")
```

b-trans 2

NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN

→ b-trans <- sweep (sweep (b, 2, b-means, "-"), 2, b-sdev, "/")

b-trans

NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN.

→ The aggregate () function :-

aggregate (x, by, FUN, ... simplify = TRUE).

head (mtcars, 15)

tail (mtcars, 5)

sapply (mtcars, class)

dim (mtcars)

unique (mtcars $ dep pc)

column repeat.

⇒ Vectorization as an alternative to loop & apply function as aggregate(), by(), sapply etc.

Filtering:-

→ The following properties are mentioned:

- Rows are considered to be a subset of i/p
- Rows in the subset appear in the same order as the original data frame.
- columns remain unmodified.
- The number of groups may be reduced, based on conditions.
- Data frames attributes are preserved during the data filter.
- Row number may not be retained in the final o/p.

→ The data frame rows can be subjected to multiple conditions by combining them by

And, or.

→ using indexing method & which() function :-

→ multiple conditions can also be combined using which() method in R.

→ the which() function in R returns the position of the value which satisfies the given condition.

which (vec, arr.ind = F).

vector subjected to conditions.

%in% - check value in the vector specified.

val %in% vec.

Ex:-

data_frame = data.frame (col1 = c ("b", "b", "d", "e", "d"), col2 = c(0, 2, 1, 4, 5), col3 = c(TRUE, FALSE, FALSE, TRUE, TRUE))

Print (data_frame) → [1] "original dataframe"
original

Print (data_frame) →

	col1	col2	col3
1	b	0	T
2	b	2	F
3	d	1	F
4	e	4	T
5	d	5	T

```
data_frame_mod <- data_frame [ which (data_frame
$col1 %in% c ("b", "e") | dataframe $col
>4), ]

Print ("modified data frame")   → [1] "modified
                                    data frame"
Print (data_frame_mod)
```

↓	col1	col2	col3
1	b	0	T
2	b	2	F
4	e	4	T
5	d	5	T

→ using dplyr Package :-

→ dplyr library can be installed which is used to reform data manipulation.

→ The filter () function is used to produce a subset of the data frame, all rows that satisfy the specified condition.

→ The filter method can be applied for both grouped & ungrouped data.

→ The expression includes (==, >, >=) (&, |, xor ()).

((between(), near()) as well as NA.

filter (df, cond)

data frame — condition.

Ex:-

```
library ("dplyr")
data_frame = data.frame (col1 = c("b", "b", "d",
                                  "e", "e"),
col2 = c(0,2,1,4,5),
col3 = c (TRUE, FALSE, FALSE, TRUE, TRUE))
print ("original data frame")
print (data_frame).
data_frame_mod <- filter (
  data_frame, col1 == "b" & col3 != TRUE)
print("Modified data frame")
print (data_frame_mod).
```

O/p -

[1] "original data frame"

	col1	col2	col3
1	b	0	T
2	b	2	F
3	d	1	F
4	e	4	T
5	d	5	T

[1] "modified dataframe"

	Col1	Col2	Col3
1	b	2	FALSE.

⇒ Using Subset Method :-

→ The subset() method in base R is used to return subsets of vectors, matrices or data frames which satisfy the applied conditions.

→ The subset() method is concerned with the rows. The rows numbers are returned while applying this method.

Syntax :-

subset (df, Cond) — to filter the data upon

/

The dataframe object

⊙ Same.

data-frame-mod <- subset (data-frame, col1 == 1 | col2 > 4)

Print ("Modified dataframe")

Print (data-frame-mod)

Q/P

[1] "original data frame"

	Col1	Col2	Col3
1	b	0	T
2	b	2	F
3	d	1	F
4	e	4	T
5	d	5	T

[1] "Modified data frame"

	Col1	Col2	Col3
1	b	0	T
2	b	2	F
5	d	5	T