Python is also interpreted programming language.

For: C, C++ are compile based.

R- Progamming

- R is an interpreted computer programming language. (means code will executed line by line)
- It is also a software envoirment used to analyTe statistical information, graphical representation, reporting and data modeling.
- The modern implementation of S is R, a part of the GNU free software project, S-PLUS, a commercial product, was formely sold by TIBCO Software
- R is the implementation of S programming language, which is combined with lexical scopic semantics
- R-programming is very useful for machine learning.
- R was developed by Ross Ihaka & Robert Gentleman in the university of Auckland, New Zealand
- R Name is taken from both of developers

- The first project was considered in 1992. The initial version was released in 1995, and in 2000, a stable beta version was released.
- Latest version of R version 4.0.0 has been Released on 2020-04-24.

⇒ CHANGES FROM 1991 – 2016

1991 → Created in New zealand by Ross Ihaka and Robert Gentleman influenced by S developed by John Chambers at Bell Labs.

1993 – August, first announced of R at public

1995 – Martin Machler convinces Ross and Robert to use the general public License to make R a free S/w.

1996 A public mailing list is created

1997 The R core group is formed, The group controls the source code of R.

2000 R version 1.0.0 was released

2013 R version 3.0.2 was released

pre-define documentry > Rscript filename.R ↵

2014-16 → R version 3.2.X to 3.3.X

2016 → R version 3.4.0 was released

2020 → R version 4.0 released.

FEATURES OF R PROGRAMMING →

- R is a comprehensive programming language that provides support for procedural programming involving function as well as object oriented programming with generic function.
- R can easily integrated with many other technologies and frameworks like Hadoop and HDFS. It can also integrate with other programming languages like C, C++, python, Java, Fortran & JS.
- R is an open source programming language, it is completely free
- Variety of packages, more than 15K packages available like CRAN, GitHub, Bioconductor
- It can produce production quality

CRAN → Comprehensive R Archive network.

publication quality graphs and plots of any kind with it's base package

- The R language is interpreted it does not need compilers to convert the code into program
- R is cross-platform supportive, it can run on any OS. It is platform independent.
- In R you can perform a wide variety of complex operation on vector arrays, data frames & other data objects. all these operation perform at lightning speed.

Supports MLR → The MLR package which stands for Machine learning in R has become highly popular.

This package is useful for all machine learning algorithms and provides other tools that helps with Machine Learning as well as

→ Default documentry is "document".

→ You can set appearance via tool option.

Ex → Print ("Hello") → it will print hello.

Assignment Operator → We can also use equal (=) operator but standard assignment operator is (<-) same as (->), (<<-), (->>)

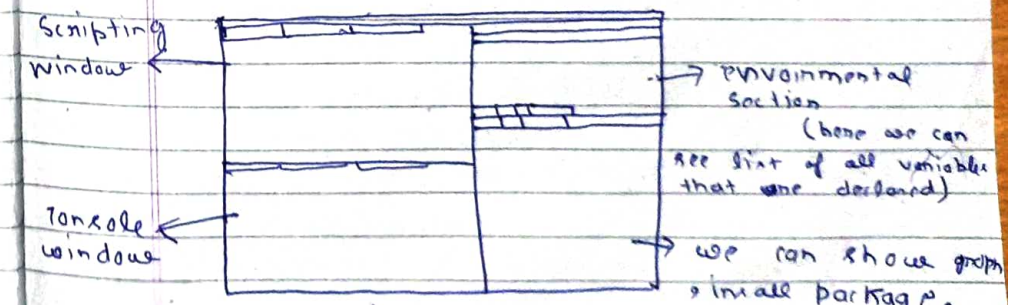
ex → b = 5
a = "hello"
c <- 5+4 or 5+4 -> c
d <- "Hello" or "Hello" -> d

source → To execute whole program

run → To execute line by line

```
print (b)
print (a)
print (c)
print (d)
```

Output ⇒ [1] 5
[1] "hello"
[1] 9
[1] "Hello"



→ /t not supports → /n supports
→ R is case sensitive.
→ Search cranistic for packages in google.

Note: (1) Use the '#' to single line comment

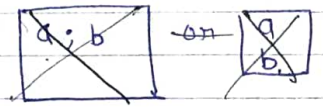
(2) Use the semicolon to print / declare diffnent thing in same line.

Ex → print (a) ; print (b)

same as

⇒ print (a)
print (b)

or



or

(3) we can start variable name with (.) or letter but it can't start with underscore (_).

(.) and (_) inspite of, we can't use any symbol in declaration.

★★ Cat function ⇒ We can use cat function to print multiple value at a time.

Ex → cat (a, " ", b, "\n", c)
↓ variable 1 ↓ variable 2 ↓ variable 3
" " → white space
"\n" → line space

output → 5 6
7

Default data type in R is numeric

⇔ if we add L or suffix of data it will consider as integer

⇒ Datatypes in R ⇐

1. Logical (TRUE, FALSE)
2. numeric → same as float (3.55, 7, -8)
3. integer → same (2, 3, -4, 7)
4. complex → 5+2i
★ 5. character → 'Hello' / "Hello"
6. raw → unuseful

Class function → tells about data type.

Ex num ← 10L
→ class (num)
→ typeof (num)

output → "integer"
"integer"

Ex → l ← TRUE
class (l)

output → "Logical"

⇒ Convent datatypes in R →

(i) vone as. numeric (data) → convent any data to numeric →

Ex → a ← as. numeric (23 L)
a
b ← as. numeric ("abc 123")
" b
c ← as. numeric ("abc")
c

output → 23
NA
abc

0 के अलावा सबके लिए TRUE आयेगा।
char के लिए सबमें NA आयेगा।

(ii) variable ← as.integer (data) → convert data into integer.

Ex → a ← as.integer (5.445)
a
→ b ← as.integer (7+5i)
b
→ c ← as.integer (TRUE)
c

Output 5
7
1

⇆ Same as → as.complex ( ), as.logical ( ), as.character ( ).

Ex → a ← as.complex (7)
a
b ← as.logical (85)
b
c ← as.character (755)
c

Output 7+0i
"TRUE"
"755"

VECTOR → Vector is a collection of similar data type.

Syntax → varname <- c (value1, value2 ...)

Ex → a <- c (1, 2, 3, 4)
print (a)
1 2 3 4

⇆ **Operators in R- programming →**

(i) Arithmatic Operator → (+), (-), (*), (/), (%.%) → for reminder, (%/%) → for quotient, (^) → for power

Ex → a ← 7.5
b ← 2
~~print (a+b)~~
print (a+b)
print (a-b)
print (a*b)
print (a/b)
print (a%.%b)
print (a%/%b)
print (a^b)

Output → 9.5
5.5
15
~~3.75~~ 1.5
3.075
56.25

(ii) Relational operator → (<), (>), (==), (<=), (>=), (!=)

The output of relational operator is always true or false

Ex → a = 55
b = 75
print (a > b)
print (a < b)
print (a == b)

Output → FALSE
TRUE
FALSE

(iii) Logical operator → (&) → and, (|) → or, (!) → not
(&&), (||)
double and, double or

Ex → using vector →

d ← c (3,5, TRUE)
e ← c (2, 4, FALSE)

print (d && e) → checks only 1st element
print (d & e) → check all element
print (d | e) → check all element
print (d || e) → check 1st element
print (!e) →

Output → TRUE
TRUE TRUE FALSE
TRUE
TRUE TRUE TRUE
FALSE FALSE FALSE

⇒ If - else Statement in R- programming

⇒ Same as C, except 1 thing we have to write code of else just after the if block other wise it'll show error.

Syntax
```
if (condition)
{
  statement
}else
{
  statement
}
```
→ if we have used else in next line it show us the error.

Ex →
```
x ← 25L
if (is.integer(x))
{
  print("x is integer")
}else
{
  print("x isn't integer")
}
```

Output → x is integer

Note → (value %in% vector_variable) this operator is used to search for

a specific value in vector.

Example → a ← ("Hey", "How", 'are', 'you')
```
if ('hello' %in% a)
{
print("found")
}else
{
print("not found")
}
```

⇒ else-if statement → same as in 'C'.

Syntax
```
if (condition 1)
{
}else if (condition 2)
{
}- - -

- - -

}else
{

}
```

⇒ Switch case statement →

(i) Based on the index value →
switch(expression, case1, case2 --)

Ex →
```
x <- switch (2,
             "Ram",
             "Shyam",
             "Mohan"
             )
print (x)
```

output → Shyam

(ii) Based on matching value →

Ex →
```
y <-"20"

x <- switch (y,
             "4" = "Ram",
             "14" = "shyam",
             "20" = "Mohan",
             "25" = "Sumit"
             )

print (x)
```

output → Mohan

Note → compare same data type value ex →
string = string
numeric = numeric
integer = integer

Repeat

Next and break statement →

⇒ Next will skip the current itteration but doesn't exit from the loop. Just as continue.

Ex →
```
x <- 1:10
for (val in x) {
    if (val == 5) {
        next
    }
    print (val)
}
```

output →
```
1
2
3
4
6
7
8
9
10
```

⇒ Repeat is a loop without condition Break is used to exit from the loop.

Ex →
```
a<-1;
repeat {
    print ("Hello")
    if (a>5) {
        break
    }
    a <-a+1
}
```

output →
```
Hello
Hello
Hello
Hello
Hello
Hello
```

for loop in R →

Syntax ⇒ for (variable in vector)

Ex →
```
for (y in 1:10)
    print (y)
```

o/p →
```
1
2
3
4
5
6
7
8
9
10
```

Paste ⇒ Use to concat the string.

Ex →
```
for (y in 1:10){
    print (Paste ("Number :", y))
}
```

o/p → Number :1

★★ Print (paste ("Hello my name is", name, "my age is", age))

O/P) myname is abc
my age is 12

Ex) a <- c ('P', 'Q', 'R', 'S', 'T')

(i) for (i in a) (ii) for (a in a)
{
print (f)
}

(i) O/P → P Q R S T
P Q R S T
P Q R S T
P Q R S T
P Q R S T

(ii) O/P → P
Q
R
S
T

⇒ While loop in R ⇒

Syntax
While (Condition)
{
Statement
x <- x+1
}

Note ⇒ Don't write 1 in condition otherwise it'll go infite

x <- 1
Ex → v <- c ("a", 'b', 'c', 'd')
while (x < 5)
{
print (v)
x <- x+1
}

output → a b c d
a b c d

paste 0() is fasten than paste()
readline() take input as string, we have change into desired data type.
Ex → x <- as.integer(readline())

⇒ Take input from user →

(i) Readline () →
variable name
It will show
Syntax var_name <- readline (Prompt = "message")
Keyword

Ex → name <- readline (Prompt = "Enter your name")

O/P → Enter your name -

★
⇒ Diffrence b/w Paste() and paste0()

Paste concat the string using seppraten (Sep = " "), in default paste concat string with seppra-ten space.

while paste 0 simply concat the string.

Ex →
paste ("a", "b", "c")
paste ("a", "b", "c", Sep = "_")
paste0 ("a", "b", "c")

O/P → a b c
a-b-c
abc

⇒ Writing Function in C →

(i) Usen define function →

keyword

Syntax ⇒ fun_name <- function (arg1 , arg 2..)
{
body
}

Example ⇒ new <- function ( ) → function declare
{
for (i in 1:5){
print (i^2)
}
}
new ( ) → function call

Example → new 2 <- function (x, y, z)
{
a <- x+y+z
print (a)
}

new2(4,5,6)

Example → new 3 <- function (x=5, y=6) → default argumen
{
a <- x*y
print (a)
}
new 3 (7,8) → actual argument

o/p ⇒ 56

bcz actual argument override the default arguments.

Predefine function →

(i) absolute function → abs ( ) gives the absolute value.

(ii) Square root function → sqrt ( ) gives squnt of value

(iii) ~~floor~~ ceiling ( ) → gives the least integer value (greater than or equal to x).

(iv) floor ~~ceiling~~ ( ) → gives the greatest int value (less than or equal to x)

(v) trunc ( ) → gives the nearest intega value that is not greater than angument.

Example ⇒
x <- -5
y <- 43
z <- 74.39
print (abs (x))
print (sqrt (y))
print (floor (z))
print (ceiling (z))
print (trunc (z))

O/P → 5
6.557439
74
75
74

(vi) sin(), cos(), tan(), log(), exp()

Function Related to string → ⇒

1. Substring ⇒ To select the characters with in the given Range in a string.

Syntax ⇒ substr (st-name, from, to)

Ex → a<-"1 2 3 4 5 a b c"
substr (a, 3, 5)

O/P → "3 4 5"

2. to lower () → convert string in lower case

3. toupper () → convert string in upper case.

Ex → a <- "a b c d"
b <- "E f g H"
print (toupper (a))
print (to lower (b))

O/P → ABCD
efgh

4. grep () → search for a pattern in string or vector string.

Ex → a <- c ("abc", "def", "abcf")
b <- "^abc"
print (grep (b, a))

O/P → 1 3

5. sum () → to add all values under a vector.

Ex → a <- c(1, 2, 3, 10, 12)
b <- sum (c)
print (b)

O/P → 28

6. min (), max () → to print minimum or maximum value from a vector.

Ex → a <- (7, 8, 10, 11)
b <- min (a)
c <- max (a)
~~print~~ (
cat (b, c)

O/P ⇒ 7 11

⇒ **Data Structure in R →**

5 components in DS for R
Vector
matrix
array
list
data frames

(i) **Vector →** It stores same type of data. element of vectors known as component. It is sequential

⊙ length() function is used to find number of element in vector

⊙ vector is of two part →
(i) atomic vector
(ii) list

⊙ we uses c function to create vector. This function returns 1 dimensional array.

Ex a ← c(1,2,3,4,5)

☆ ⊙ we can also use (:) operator to declare vector, It stores element that one in range.

Ex b ← 3:5

⊙ we can use seq() function to declare we can also use by tag in it to gap b/w element

Ex a <- seq(3,5, by = .5)
print(a)

O/p 3.0, 3.5 4.0 4.5 5.0

Ex ⇒ a ← seq(1,4, length.out = 7)
print(a)

O/p ⇒ 1.0 1.5 2.0 2.5 3.0 3.5 4.0

(i) **Atomic Vector ⇒** It has 4 type ⇒

(a) **numeric vector ⇒** By default declare vector is of numeric type.

a ← c(1,2,3,.5,7)
print(class(a))

O/p ⇒ numeric

(b) **integer vector ⇒** Declare the same as above then use as.() function or use L with suffix of every component of vector.

Indexing starts from 1 in R program.

(c) Character vector → use as. function to convert vector into character vector or else use every component character.

(d) Logical vector → use as. function

- we use '[]' bracket to access the component of vector.
  ex → a[5]
- we can combine 2 vector in 3rd vector like this →

```
a ← c(1, 2, 3, 4)
b ← c(5, 6, 9)
c ← c(a1, a2)
```

- we can also apply arithmatic operation on vector

Note ⇒ if we give negative value in indexing then it will consider every value of vector except that's ~~abs~~ value.

```
Ex ⇒ a ← (1, 2, 8, 9, 15)
     print(a[-3])
```

O/p ⇒ 1, 2, 9, 15

- we can also print multiple index.

```
Ex  a ← c(1, 2, 8, 4, 5)
print(a[c(1, 3)])
```

O/p → 1, 8

(ii) List () → if we want to store different type of data in one then we use the concept of list.

unlike atomic vector where we use c() function to store data, in list we use list () function to store data.

```
Ex ⇒ a ← list('a', 1, TRUE, c(1,2,3))
     print(a)
```

⇒ Naming of list ⇒ we can give the name of different data inside a list and access via these name.

```
Examples ⇒ list1 ← list(c('a', 'b', 'c'), c('1', '2', '3')
                        , c('P', 'Q', 'R'))

names(list1) ← c("X", 'Y', 'Z')

print(list1[3])      → via index
print(list1["Z"])    → via naming
print(list1$Z)       → via dollar sign
```

O/P ⇒ P & R
P & R
P & R

⇒ Unlist function ⇒ We can't perform all arithmatic function in a list so we use unlist function to convert it to vector

Example →

```
a ← list (1:4)
a ← unlist (a)
b ← list (2:5)
b ← unlist (b)
print (a+b)
```

O/P ⇒ 3, 5, 17, 9

NOTE → we can also ~~use merge function to~~ merge list.

Example

```
c ← list (a, b)
print (c)
```

⇒ VECTOR FUNCTON ⇒

(1) Rep() function ⇒ We can repeat the value and the vector using this function.

(i) time constraint → to repeat the loop

(ii) each constraint → to repeat value multiple time

(iii) length.out constraint → to print how many time you want to repeat the value

Example →

```
a ← (4, 5, 9, 12)
rep (a, time = 3)
rep (a, each = 2)
rep (a, length.out = *)
```

Output → 4 5 9 12 4 5 9 12 4 5 9 12
4 4 5 5 9 9 12 12
4 5 9 12 4 5 9

(2) Sequence function → to specify the sequence with the given range. from a given value to given value.

★ any = at least one (then true)
★ all = all value must (then true)

Example Seq (from = 3, to = 7, by = .5)
~~seq~~
a ← c(1, 2, 3, 5, 7)
Seq (from = a[3], to = a[5], length.out = 11)

Output ⇒ 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0

3.0, 3.4, 3.8, 4.2, 4.6, 5.0, 5.4, 5.8, 6.2, 6.6, 7.0

★ ★
(3) Any and all function ⇒ ~~If~~ a for a given condition if any of the value satisfy the condition then any() function will return TRUE. But the all() function will return TRUE only if all the value satisfy the condition.

Example ⇒ a <- (1:10)
any (a > 7)
all (a < 7)

Output ⇒ TRUE
FALSE

√. Imp

ARRAYS ⇒ Arrays are the data object which allow us to store data in more than two dimension.

Syntax ⇒ array_name ← array (data, dim = ())
to declare the dimension and quantity of matrix.

★
Example ⇒ V1 ← c(7, 8, 9, 10)
V2 ← c(15, 17, 18, 19, 21, 23)
V3 ← array (c(V1, V2), dim = c(3,3))

★
Output
7	10	18
8	15	19
9	17	21

23	9	17
7	10	18
8	15	19

⇒ Naming the array ⇒ we can name an array using dimname parameter

Example ⇒ Col ← c('C1', 'C2', 'C3')
Row ← c('R1', 'R2', 'R3')

mat ← ('mat1', 'mat2')

V3 ← array (c (V1, V2), dim = c (3,3,4)
, dimnames = list (row, col, mat))

V3

NOTE We can access the matrix via ([])

print (V3 [1,2,2])

it will print 1st row's 2nd column of 2nd matrix.

★ MATRIX ★

to store value in 2 dimension

Syntax ⇒

mat_name ← matrix (data, nrow =
, ncol = , byrow = FALSE/TRUE)

↓

if it is TRUE
then value will be assigned by row

Ex → mat ← matrix ((1:12), nrow = 4
ncol = 3, byrow = TRUE)
mat

output →

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

use %*% operator for matrix multiplicati

use t(matrix_name) to transpose matrix

NOTE (1.) naming is same as arrays using dim_names.

(2.) we can alter the element of matrix directly.

Ex mat [4,2] ← 5
mat
mat [mat > 9] ← 0
mat

O/P →

1	2	3
4	5	6
7	8	9
10	5	12

1	2	3
4	5	6
7	8	9
0	0	0

(3) cbind() and rbind function are used to add rows or column in matrix. (Temporary)

```
V1 ← (5, 6, 7) ; V2 ← (8, 9, 10, 11)
rbind (mat, V1)
cbind (mat, V2)
Z
```

Z

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

→ ~~Factor data type) used in database~~

⇒ **DATA FRAME** ⇒ it's like a 2-D array structure. unlike an array the data we store in the column of data frames can be of various types.

or

A data frame is a 2-D Array-like structure or a table in which a column contains one ~~se~~ values of one variable and rows contain one set of values from each column.

A data frame is a special case of the list in which each component has equal length.

Create the data frame → we use data.frame() function to create data frame

~~Ex → emp.data <- data~~

Example ⇒

```
data <- data.frame(
    id = c(1,2,3,4),
    name = c("Siud","Ram","Shyam",
             "Mohan"),
```

→ function, start bracket, use (,), use (=) operator

```
Join_date = as.Date(c("2020-01-01",
                      "2020-01-02",
                      "2020-01-03",
                      "2020-01-04"),
```

as.Date() use to make date

StringAsFactors = FALSE

it will convert string datatatype to factor

) → closing bracket

```
print(data)
```

O/p ⇒

id	name	Join_date
1	Siud	2020-01-01
2	Ram	2020-01-02
3	Shyam	2020-01-03
4	Mohan	2020-01-04

⇒ Here every column has diffrent data type & length of every column is same. we can't do this in vector.

NOTE ⇒ use str(data_frame_name) to print structure and datatype of data frame.

Example → str(data)

O/p ⇒

```
data.frame : 4 obs. of 3 variables
$ id        : num  1 2 3 4
$ name      : chr  "Siud" "Ram" "Shyam" "Mohan"
$ Join_date : Date Format: "2020-01-01", "2020-01-02", "2020-01-03"
```

NOTE if we want to specific row or column retrieval

Example # for column →

```
f1 ← data.frame(data$name, data$id)        (data frame name)
print(f1)
```

for row →

```
f2 ← emp.data                              (will print 2nd row)
f2 ← data[2, ]
print(f2)
```

for specific row & column →

```
f3 ← data[c(2,3), c(1,3)]                  print whole 2 & 3 row
```

(for 2nd & 3rd row of respective 1st and 4th column)

print(f3)

also can use

f3 ← c[2,3]

o/p →

2020-01-03

O/p →

data.name	data.id
Sud	1
Ram	2
Shyam	3
Mohan	4

id	name	Join-date
2	Ram	2020-01-02

id	Join-date
2	2020-01-02
3	2020-01-03

To Add a column or Row →

⇒ # rbind →

```
X ← list(5, "Rohan", "2020-01-05")
rbind(data, X)
```

⇒ # cbind →

```
Y ← c("Kanpur", 'Delhi', 'Noida', 'Banglore', 'Lucknow")
cbind(data, Address = Y)
```

↓ it will add whole column with address name

To Delete a column or Row →

⇒ # delete row →

```
data ← data[-2, ]
```

(it will delete 2nd Row of data frame data)

⇒ # delete column →

```
data$Join-date ← NULL
```

(it will delete the whole Join-date column)

⊙ we can use summary() function to print summary of the created data frame

CATEGORICAL VARIABLES

A categorical or discrete variable is one that has two or more categories variable.

Ex → week, Gender

It's of two types →

(i) Nominal
(ii) Ordinal

(i) Nominal Variable → A nominal variable has no ~~instrinct~~ intrinc ordering to it's categories

Example → Gender having two categories (male & Female) with no intrinsic ordering.

(ii) Ordinal variable → It has a clear ordering.

Example temprature as a variable with 3 orderly categories. (low, medium & high)

FACTOR → R factor is used to store categorical data as level. It can store both character & integer types of data These factors are created with the help of factor() function, by taking a vector as input.

Factors have labels which are associated with the unique integer stored in it. it contains predefined set values known as levels and by default R always sort level in alphabetical order.

→ ATTRIBUTES of a Factor →

(i) X → it is the input vector which is to be transformed into a factor

(ii) levels → It is an input vector that represent a set of unique values which are taken by x.

(iii) labels → It is a character vector which corresponds to the number of labels.

(iv) Exclude → It is used to specify the

value which we want to be excluded.

(v) **Ordered** → It is a logical attribute which determines if the levels are ordered

(vi) **nmax** → used to specify the upper bound for the maximum number of level.

Example →

```
dir <- c('E', 'W', 'N', 'S')
factor(dir)
a <- c('a', 'b', 'c', 'a')
factor(a)
```

O/P →

```
E W N S
Levels: E W N S
a b c a
Levels: a b c
```

Example →

```
dir <- c("North", "East", "West", "South")
factor(dir, labels = c('N', 'E', 'W', 'S'))
factor(dir, exclude = 'South')
```

↓ it will exclude level of South

Example

```
v <- gl(3, 4, labels = ('a', 'b', 'c'))
print(v)
```

(gl() function used to create level)

(no. of levels do u want) (no. of colors of level do u want)

O/P → a a a a b b b b c c c c

NOTE → We can access factor using [ ]

Ex →

```
data[c
dir[2]
dir[c(2,3)]
dir[-1]
```

O/P →

```
East
East West
East west South
```

2. We can modify factor using [ ]

Ex → `dir[2] <- "North"`

⇒ **function related to Factor :**

(i) **is. factor** ⇒ check whether input is factor or not & return True / False

(ii) **as. factor()** → convert vector into factor

(iii) **is. ordered** → check whether is ~~a~~ factor is ordered or not.

(iv) **as. ordered()** → Take unorder ve factor as input & arrange in order.

⇒ DATA VISUALIZATION In R-PROGRAMMING ⇒

It's an efficient technique for gaining insight about data through a visual medium with the help of visualization technique, a human can easily obtain information about hidden patterns in data that might be neglected.

(i) R - Pie chart ⇒ Pie chart is a representation of value in the form of slices of a circle ~~which~~ with diffrent colors.

The pie chart is created with the help of pie() function. which take positive number as vector input.

Syntax ⇒

Pie (X, Labels, Radius, main, col, clock wise)

where,

- X is a vector that contains the numeric value used in pie chart
- Labels are used to give the description

graphics package is already installed in system library.

-or to the slices.

- Radius describes the title of the chart.
- main describes the title of chart
- col decides color pattern
- ~~anti~~ clock wise is logical value that indicates the c-wise or a-wise direction in which slices are drawn

Example → X ← c ("12, 13, 15, 50)

labels ← c ("UK", "NZ", "Jap", "Ind")

~~Pie (X, labels)~~

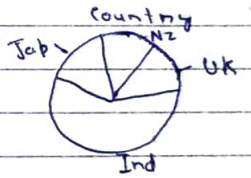
Colors ← c ("blue", "Green", "Red", "Yellow")

or

colors ← rainbow (length(x))

pie (X, labels, main = "Country", col = colors)

O/P →



Legend tag → Use to show the Indicate box which shows the description & summary of pie chart

Ex Syntax → legend (X, Y, legend, Fill, col, bg)

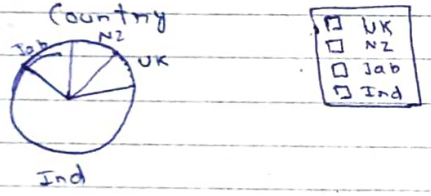
Here,

- X and Y are co-ordinates where to position the legend.
- Fill is the color to use for filling the boxes besides the legend text
- col defines color of line & points besides legen text
- bg is bkgrnd color for legend box.

Ex → legend ("topright", c("UK", "NZ", "Jap", "IND"), cex = 0.8, fill = colors)

cex for size of box

O/P →



(*) FOR 3D →

⇒ 3-Dimensional Pie chart ⇒ R provides a plotrix package whose pie3D() function is used to create an attractive 3D pie chart. The parameter of pie3D() function remain same as pie() function.

NOTE ⇒ First we need to install plotrix package before doing this.

we can also use round function to show percentage in of every slice.

Example ⇒ library(plotrix)

↓

It will load the plotrix package

```
X ← c(20, 40, 60, 50, 45)
labels ← c("Ind", "Ame", "Sri", "Nep", "Bru")
pie3D(X, labels, main = "Country",
      col = rainbow(length(X)))
```

(ii) R- bar chart → A bar chart is a pictorial representation in which numerical values of variable are represented by length or height of lines or rectangles of equal width. A bar chart is used for summarizing a set of categorical data. In bar chart, the data is shown through rectangular bars having the length of the bar proportional to the value of variable.

Syntax ⇒ barplot(H, x, y, main, names.arg, col)

Here,

H is a vector or matrix contain numeric values used in bar chart

(X, Y) are ~~co-ordinate~~ labels for X and Y- axis respectively

main - title of bar chart

names.arg → a vector of names that appear under each bar

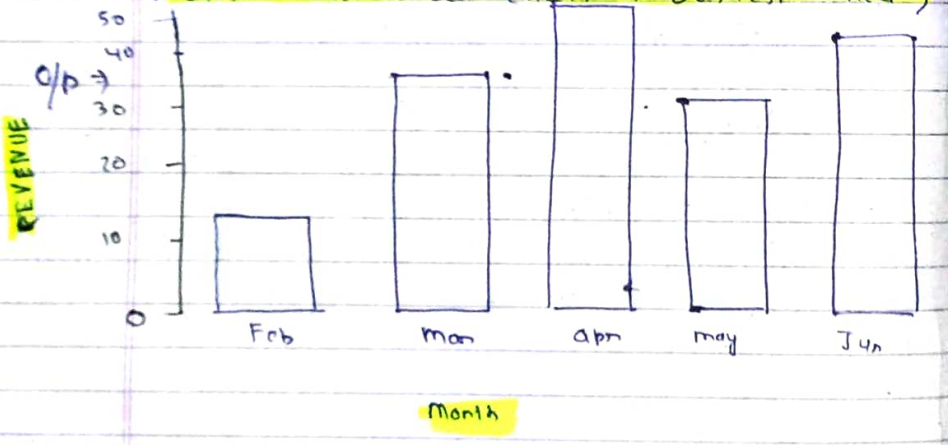
col → It is used to give colors to the bars in graph.

Example :

```
H1 ← c(12, 13, 15, 18, 20)
barplot (H)
```

OR

```
H2 ← c(12, 35, 54, 31, 41)
M2 ← c("Feb", "Mar", "Apr", "May", "Jun")
barplot (H2, names.arg = M2, xlab = "Month", ylab = "Revenue", col = "yellow", main = "Revenue bar chart", border = "Red")
```



NOTE ⇒ we can use matrix in place of vector to show various state in one bar.

(iii) R- histogram → A histogram is a type of bar chart which shows the frequency of the numbers of values which are compared with a set of values range. The histogram is used for the distribution, whereas a bar chart is used for comparing different entities.

In the histogram each bar represents the height of number of values that present in given range.

For creating a histogram we use hist() function.

SYNTAX ⇒

```
hist(V, main, xlab, ylab, xlim, ylim, breaks, col, border)
```

Here →

X-lim is used to specify the range for x-axis.

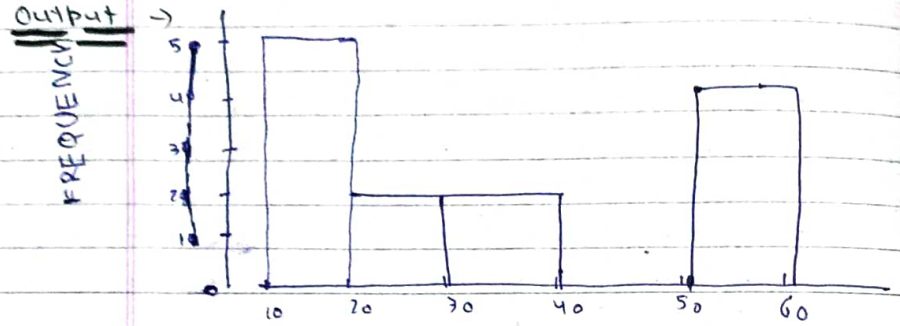
Y-lim is used to specify the range for Y-axis.

break is used to mention the width of the bar.

Example → V ← c(12, 24, 16, 38, 21, 13, 55, 17, 39, 10, 00, 59, 58)

hist (v, xlab = "Weight", ylab = "Frequcy", col = "green", border = "red")

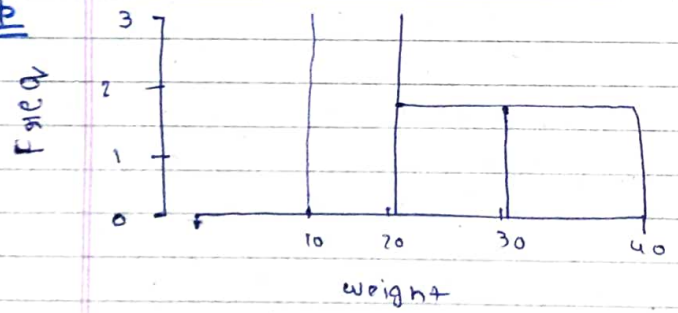
Output →



⇒ Use of xlim & ylim →

hist (v, xlab = "weight", ylab = "Freq", col = "yellow", border = "green", xlim = c(0,40), ylim = c(0,3), breaks = 4)

O/p



by default plot() will create the graph with point.

(iv) Line Graphs in R → There are points connecting the data to show the continious change, the time in a line graph can move up & down based on the data.

R programming plot() function, which has the following syntax→

Syntax ⇒

plot (v, type, col, xlab, ylab)

Here →

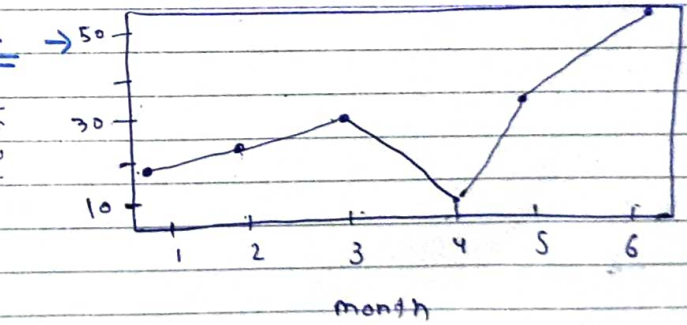
type parameter takes the value
(l) to draw the lines only
(p) to draw the points only
(o) to draw the both

Example →

v ← c(18, 22, 28, 7, 31, 52)

plot (v, type = "o", col = "blue", xlab = "month", ylab = "temp")

Output →

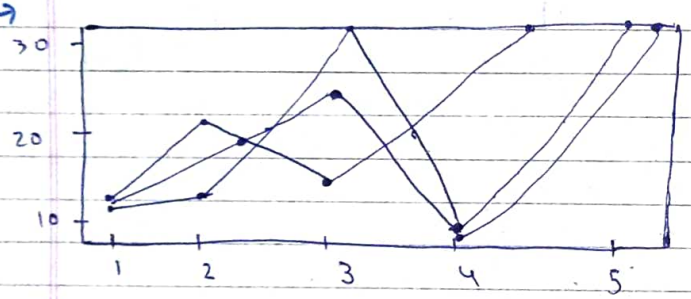


Line chart containing multiple lines →

use the line function along with plot function.

```
v ← c (13, 22, 28, 7, 31)
w ← c (11, 13, 32, 6, 35)
x ← c (12, 22, 15, 34, 35)
plot (v, type = "o", col = "green",
      ylab = "Mon", xlab = "Temp")
line (w, type = "o", col = "Red")
line (x, type = 'o', col = "blue")
```

op →



mtcars ~~mtplot~~ is available data frame in R programming

★★★

How to make Scatter plot in R programming →

The scatter plots used to compare variables

A comparison b/w variables is required when we need to define how much one variable is affected by another variable. In a scatterplot, the data is represented as a collection of points, Each point on the scatter plot defines the value of the two variable. one variable is selected for vertical axis and other for the horizontal axis

In R there are two ways of creating scatter plot, i.e → using plot() function and using the ggplot2 package's function

SYNTAX →

plot (X, Y, main, xlab, ylab, xlim, ylim, axes)

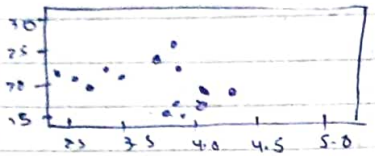
X, Y → data for horizontal & vertical axes respectively.

axes → It indicates whether both axes are to be drawn on the plot or not.

Example → data ← mtcars [, c ('wt', 'mpg')]
plot (X = data $ wt, Y = data $ mpg, xlab =

"weight", ylab = "Milage", xlim = c(2.5,5)
, ylim = c(15,30) , main = "Weight vs Milage")

Op →



⇒ Using ggplot2 package →

in ggplot2 package provides ggplot() and geom_point() function for creating a scatter plot. The ggplot() function takes a series of the input item.

The 1st parameter is an input vector, and the 2nd is the aes() function in which we add the aestatic x-axis and y-axis.

Example

```
library(ggplot2)

ggplot(mtcars, aes(x=drat, y=mpg))
  + geom_point()
```

↓ use the + sign

NOTE ⇒ The aes() function inside the geom_point() function controls the colors for group →

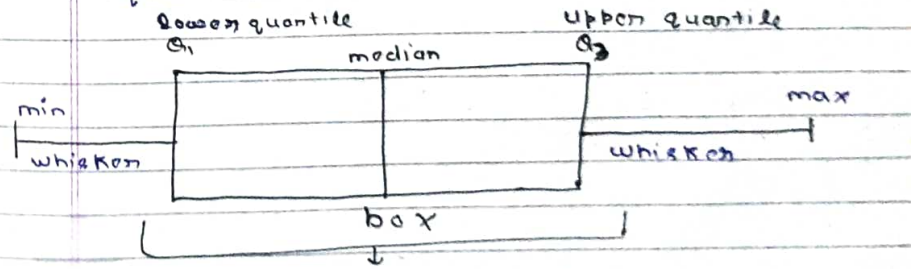
```
ggplot(mtcars, aes(x=drat, y=mpg))
  + geom_point(aes(colour = factor(gear)))
```

↓ give the unique value of gear variable.

BOX PLOT IN R-PROGRAMM - ING →

A box plot (also known as box and whisker plot) is a type of chart often used in explanatory data analysis to visually show the distribution of numerical data and skewness through displaying the data quantiles (or percentiles) and averages.

Box plot show the five number summary of a set of data : including the minimum score, first (lower) quantile, median, third (upper) quantile and maximum score.



Inter quantile range (IQR)

Example → Finding the five number summary.

A sample of boxes of raisinx box these weight (in grams).

25, 28, 29, 29, 30, 34, 35, 37, 38

Make a box plot of data →

step→1 order data in ascending order →

25, 28, 29, 30, 34, 35, 37, 38

step→2 find median.

median = $\frac{30+34}{2}$ = 32

step→3 find quantiles.

1st quantile is the median of the data point to left of the median →

25 28 (29) 29 30

Q_1 = 29

3rd quantile is the median of the data point right of the median →

34, 35 (35), 37, 38

Q_3 = 35

step→4 Complete 5 number summary by finding min & max.

min = 25

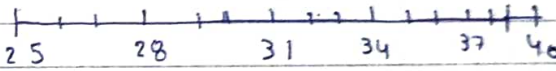
max = 38

The five number summary →

25, 29, 32, 35, 38

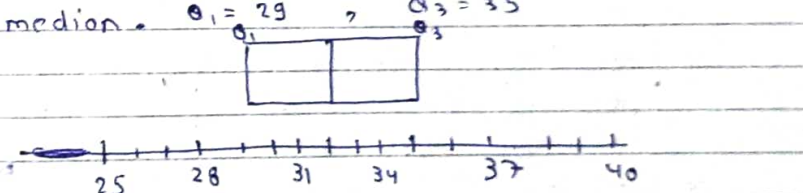
MAKING A Box plot →

step→1 scale & lable on axis the five



weight (grams)

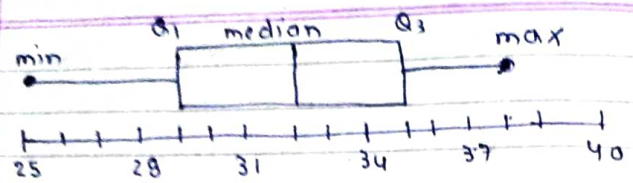
STEP→2 Draw a box from Q_1 to Q_3 with a vertical line through median.



weight (grams)

Step→3 Draw a whiskers from Q_1 to the min & Q_3 to the max.

min = 25 , max = 38



now this graph is called box plot.

⇒ we use box plot() function to create a box plot.

SYNTAX →

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

Here →

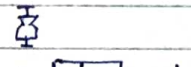
- x is a vector or Formula
- data is the data frame
- notch is the logical value set as true to draw the notch.
- Varwidth is also logical value set as true to draw the width of the box same as sample size
- names is the group of lables that will be printed under each box plot
- main is used to give title to the graph.

by default notch is FALSE.

notch helps us to find out how medians of different data group matches with each other.

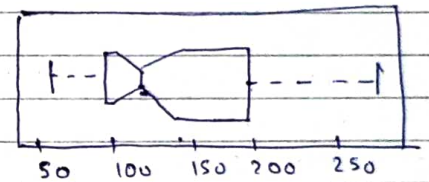
Example →

```
boxplot(mtcars$hp,
        main = "mtcars Data Frame",
        xlab = "x",
        ylab = "y",
        col = "orange",
        border = "brown",
        horizontal = TRUE,
        notch = TRUE,
        )
```

if false 

if False

O/P →



⇒ The function box plot() can also take in formula of the form y~x to the value of x. where y is a numeric vector which is grouped according to the value of x.

Example →

```
boxplot(mpg ~ cyl, data = mtcars,
        xlab = "Qunty of cylndrs",
        ylab = "miles per galon",
        main = "R box plot",
        col = "orange",
        border = "blue")
```

O/P →