# R Programming

04/03/2022

(1)

## Introduction to R:-

R is a programming language & software environment for statistical Analysis Graphics representation and reporting
R was created 'Ross Ihaka' and 'Robert Gentleman'
R is especially for Statics and mathematics.

## Why this R ?

R is a platform and set of libaries open source and free software means Do license we no need any agreements we can download freely

- It is a dynammic typed language
- this programming language having a capbility to deal with advanced Technologies like IOT, neural networks, machine learning, Robotics etc.
- It is functional based language.
- All functions are available in package
- It is a Interpreter programming language
- It is a distributed by C language (R community)
- We can attach all the packages to R environment
- It is easy to implement and develop.

## What is S?

07/03/2022

S is the programming language this is used for statistics But these is not free software this is Commerial product
- this is open source product

R made its first apperience in 1993.
In 1997 the core team modify the R source code "archive"

R basic Syntaxs:

In Commond prompt types - $

This will launch R interpreter and you will get a prompt ">"

form this '>' you can start typing your program

Eg:- > Hello world -> o/p - Hello world

> mystring ← "Hello world" ⇒ o/p -

Print (mystring) ⇒ o/p - Hello world

> Print ("Hello world") ⇒ o/p - Hello world

> print (2+3) ⇒ o/p - 5

Rscript file:

In windows we can executes Scripts ah your Commond Prompts with the help of R Interpreter called R script

The above code whether in text file save with text.R to complie this text.R as Rscript_ text.R

Rscript space file name.R ∀ in linux environment

Comments:

Use # symbol to give comment statement the

# my first program in R programming language

R does not support multipule Comment Statements

R data types:-

Preview of some Important R data structures:

In R programming language we can use the data structures It will help the arrange the data in a sequence order

R has a verity of data structures in that we will use some of the most frequently used structures to give an overview of R

Before we will discuss about R data types we have to know the Integer, Character, Complex, numeric, raw and logical

Variables are reserved memory location to store values.

there are many types of R Objects

There are:

1. Vectors
2. lists
3. Matrices
4. Arrays
5. Factors
6. data frames

Features of R :- 05/03/2022

The following are the Important features of "R"

- R is a well developed, simple and effective programming language which includes conditional, loops, user defined recursive functions and input & output facilities
- R has an effective data handling and storage facility
- R provides a set of operations for calculations on arrays, lists, vectors and matrices
- R provides a large, coherant and integrated collection of tools for data analysis
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers

*** Conclusion of R

R is worlds most widely used static programming language. Its the # one choice of data science and supported by a vibrant and talented community of contributs R is tough in Universities and de

Vectors:

In Vector object their are 6 datatypes of this automic vectors also termed as 6 classes of vectors

A vector is sequence of data elements which can store multiple values of similar data types.

eg:- a←1 ⇒ o/p 1
a←10 ⇒ o/p 10
a←20 ⇒ o/p 20

error message - a(1, 10, 20)

Error! unexpected error in a assigned to 20,

we can't store multiple values in variable

But by using Vectors we can store multiple values in variable

In this members in a Vector are called components

Here, Similar data type means all the values of a vector can be either numeric, character or logic

vectors are dimensionless

Simplest way to create a vector is to use the command c()

c fun which stands for combined

```
eg:- #create a vector
     csm ← c("girls" & "boys")
      Print (csm)
     # Print vector type
       Print (class (csm))
```

o/p girls & boys

character

⇒ Print ("csm is a combination of:" csm) x

Cat (csm is a combination of (csm)) ✓

Types of vectors:

1. logical: It produces true and false result

```
eg:- #create a logical data type
     csm ← True
     Print (csm)
```

```
# print vector type
Print (class (csm))
```

O/p : True, 10.
logical

2. Numeric : It produces the numbers

```
Eg: # create a
    csm ← -23
    Print (csm)
# print vector type
  Print (class (csm))
```

O/p = -23
Numeric

3. Integer := It produces the numbers result.

```
Eg: csm ← -23L
    Print (csm)
# print vector type
  Print (class (csm))
```

O/p = 23
Integer

4. Complex := It produces the complex results.

```
Eg: csm ← 2+3i
    Print (csm)
# print vector type
  Print (class (csm))
```

O/p = 2+3i
complex.

5. Character := It produces the character result

```
Eg: csm ← c("a", "good", "true", "23.4")
    Print (csm)
# print vector type
  Print (class (csm))
```

O/p = a, good, true, 23.4
character

6. Raw := It produces location of

```
Eg: csm ← charToRaw ("Hello")
    Print (csm)
# print vector type
  Print (class (csm))
```

O/p = 48 65 6c 6f
raw

List: A list is an "R" object which can be a may. different types of elements inside it like vector function and even another list inside it.

Syn eg:- # create a list

list 1 ← list (c(1,2,3,4), 2i.3, "hai")

# Print list

Print (list 1)

[1] 1,2,3,4

[2] 2i.3

[3] "hai"

Matrices:- A matrix is a two dimesional rectangle data set it can be created by using a vector input to the matrix function

eg:- # create a matrix

M= matrix (c('a','b','c','d','e','f'), nrows=2, ncol=3,

by row= TRUE

Print (M)

[1] a b c

[2] d e f

Arrays:- while matrices are confined with two dimensional arrays can be of any no of dimensions #

The Array function takes Dim() attribute which creates the required no of dimensions.

eg:- # create an array

a ← array (c('green', 'yellow'), dim(3,3,2)

Print (a)

o/p

$$\begin{bmatrix} \text{green} & \text{yellow} & \text{green} \\ y & g & y \\ g & y & g \end{bmatrix} \quad \begin{bmatrix} y & g & y \\ g & y & g \\ y & g & y \end{bmatrix}$$

Factors :- Factors are the 'R' objects which are created using a vector. which can it stores the vector along with the distinct values of the elements in the vector as labels

The labels are also character irrespective of whether it is numeric, integer or character or Boolean etc in the vector they are useful in statistical analysis and modeling. Factors are created using the function factor()

The n levels functions use the count of levels functions

Eg:- apple_color ← c ('green', 'green', 'yellow', 'red', 'red', 'red', 'green')

factor_apple ← factor (apple_color) → print (factor_apple)

o/p:- [1] green, yellow, red

print (nlevels(factor_apple)) o/p : [2] 3

Frames(or) Data frames:

Data frames are tabular data objects unlike a matrix in data frame each column can contain different modes of data

1. The first column can be numeric and 2nd column can be character and 3rd column can be logical
2. It is the list of vectors of equal length using the data.frame() is create a tabular formate data

```
Eg:- # create a data frame
csm ← data.frame (gender = c("male", "male", "female")
                  height = c(152, 171.5, 165)
                  weight = c(81, 93, 78)
                  Age = c(42, 38, 26)
                  )
Print(csm)
```

o/p:-

| gender | height | weight | Age |
|---|---|---|---|
| male | 152 | 81 | 42 |
| male | 171.5 | 93 | 38 |
| female | 165 | 78 | 26 |

R_variables (8) 12/03/2022

A variable provides us with named storage that our program can manipulate

In R programming, a variable can store an atomic vector, group of vectors or a combination of many R objects.

A valid variable name consist of letters, numbers and the dot or underline characters. The variable name starts with a letter or dot not followed by number.

| variable name | Validity | Reason |
|---|---|---|
| Var_name2. | valid | Has letters, numbers, . and _ |
| Var_name% | Invalid | Has the character %. Only . (dot) and underscore is allowed. |
| .Var_name | valid | Can start with dot and end with character. |
| 2 var_name | Invalid | starts with a number |
| .var_name., Var.name | valid | can start with a dot but the dot should not be followed by the no. |
| .2var_name | Invalid | Starting letter number dot follows by number makes invalid |
| -Var_name | Invalid | start with - which is not valid |

Variable Assignment:

- The variables can be assigned values using left word, right word and equal operators.
- The values of the variables can be printed using print or cat() if you are using the variable must use print or cat()
- The cat function combines multiple items into a continuous print output

Eg:- # assignment using equal operator

Var.1 = c(0, 1, 2, 3)

# assignment using leftword operator

Var.2 <- c("learn", "R")

# assignment using ~~coming~~ rightword operator

c("learn", "R") -> var.3

TRUE 1

```
Print (var.1)
Cat ("var.1 is", var.1, "\n")
Cat ("var.2 is", var.2, "\n")
Cat ("var.3 is", var.3, "\n")
```

o/p : 0 1 2 3
var.1 is 0 1 2 3
var.2 is learn R
var.3 is 1 1

Note:- The vector c(TRUE, 1) has a mix of logical and numeric class. So logical class ~~forced to~~ Coursed (or) Combined to numeric class. making TRUE as 1

Datatype of a variable:

In R a variable itself is not declared of any datatype, rather it gets the data type of R objects assigned to it. So, R is a dynamically typed language, which means that we can the variables datatype of the same variable again and again. when using it in a program

Eg:- var_x = "Hello"
Cat("the class of var_x is", class(var_x), "\n")
Var_x = 34.5 o/p:- the class of var_x is character
cat("the class of var_x is", class(var_x), "\n")
var_x = 242 o/p:- the class of var_x is numeric
Cat("the class of var_x is", class(var_x), "\n")
o/p: the class of var_x is integer

Finding a variable:-

1. To know all the variables currently available in the work space we use "ls()(list)→ls() the ls() can use Patterns to match the variable names.

print (ls())
o/p:- "var.1" "var.2" "var.3", "New_name!", "name"
"var_x" "var_name", etc..

Note:- It is a sample output depending on what variables are declare in your environment.

The ls() can use patterns to match the variable name.

Eg:- # list the variables starting with pattern "var"

print(ls(pattern = "var"))

o/p:- "var.1", "var.2", "var.3", "var_a"

The variable starting with '.' (dot) or hiden can be listed using all.name = TRUE argument to ls()

Eg:- print(ls(all.name = TRUE))

o/p:- ".var_name" "name" "var.1" ".var_name.1"

"R.name"

To Delete a Variable :-

In R programming to delete a variable using rm()

Eg:- rm(var_3)

print(var_3)

o/p:- Error! var_3 not found.

Operators :-

An operator is a symbol that tells the compiler to perform specific mathematical or logical manupulation

. In R programming language is rich built in operations

Types of operators :

we have the following types of operators in R programming

1. Arithematic operators
2. Relational operators
3. logical operators
4. Assignment operators
5. Miscellaneous Operators.

Arithmetic Operators

The following operators supported by R language the operators act on each element of the vector

| Operator | Description | Example |
|---|---|---|
| + | Adds two vectors | v←c(2,5.5,6) t←c(8,3,4) Print(v+t) (11) o/p: 10, 8.5, 10.0 |
| - | Subtract two vector (second vector from the first vector) | Print (v-t) o/p: (-6, 2.5, 2) |
| * | multiple two vectors | Print (v*t) o/p: 16, 16.5, 24 |
| / | Divide two vectors | Print (v/t) o/p: 0.25, 1.833, 1.5 |
| %% | give the reminder of the first vector with the second vector | Print (v%%t) 2.0, 2.5, 2.0. |
| %/% | the result of the division of 1st vector with the 2nd vector (Quotient) | Print (v%/%t) 0 1 1 |
| ^ | The first vector rised to the exponent of 2nd vector | Print (v^t) o/p: 256, 166.375, 1296 |

Relational Operators. 15/03/2022

Relational operators supported by R language.

- Each Element of the first vector is compared with the corresponding elements of the second vector
- The result of the comparison is boolean value (TRUE, FALSE).

| Operator | Discription | Example |
|---|---|---|
| < | Checks if each element of the 1st vector is less the corresponding element of the 2nd vector. | v←c(8,5.5,6,9) t←c(8,2.4,9) Print(v<t) TRUE, FALSE, FALSE, FALSE |
| > | checks if each element of the first vector is greater then the corresponding element of the 2nd vector | Print (v>t) FALSE, TRUE, TRUE, FALSE |

| Operator | Description | Example |
|---|---|---|
| == | Checks if each element of the 1st vector is equal to the corresponding elements of the 2nd vector | Print (v==t) FALSE, FALSE, FALSE, TRUE |
| <= | Checks if each element of the 1st vector is less or equal to the corresponding elements of the 2nd vector | Print (v<=t) TRUE, FALSE, FALSE, TRUE |
| >= | Checks if each element of the 1st vector is greater than or equal to the corresponding elements of the 2nd vector | Print (v>=t) FALSE, TRUE, TRUE, TRUE |
| != | Checks if each element of the 1st vector is not equal to the corresponding elements of the 2nd vector. | Print (v!=t) FALSE, TRUE, TRUE, TRUE, FALSE |

logical Operators:

The logical operators supported by R language.
It is applicable only to vectors of types numeric, logical or complex

All numbers greater than '1' are consider as logical value 'TRUE'
Each element of the first vector compared with the corresponding elements of 2nd vector. the result of a comparison is boolean value.

| Operator | Description | Example |
|---|---|---|
| & | It is element wise logical AND operator it combines each element of the 1st vector with the corresponding element of 2nd vector AND gives a output TRUE if both the elements are TRUE | Print (v&t) v=c(3,1,TRUE,2+3i) t=c(4,1,FALSE,2+2i) Print (v&t) TRUE, TRUE, FALSE, TRUE |
| \| | It is called element wise logical OR operator. It combines each element of the 1st vector with the corresponding elements of 2nd vector it gives a output TRUE if one of elements is TRUE | Print (v\|t) TRUE, TRUE, TRUE, TRUE |

| Operator | Description | Example |
|---|---|---|
| ! | It is called logical NOT operator. It takes each element of the vector and gives opposite logical value | v ← c(3,0, TRUE, 2+3i) Print (!v) O/P: FALSE, TRUE, FALSE, TRUE |
| & | It is called logical AND operator. It takes the 1st element of both vectors and gives the true value if both are TRUE | ⇒ The logical operator && and \|\| consider only the first element of the vector and give a vector of single element as output<br>v ← c(3,0, TRUE, 2+3i)<br>t ← c(4,1, FALSE, 2+2i)<br>Print (v && t)<br>O/P: TRUE |
| \|\| | It is called logical OR operator. It takes 1st element of both vectors and gives the TRUE when if any one is TRUE | v ← c(0,3, TRUE, 2+3i) t ← c(0,4 FALSE, 2+2i)<br>Print (v \|\| t)<br>O/P: FALSE |

Assignment Operators:

This operators are used to assign a value to a vectors.

| Operator | Description | Example |
|---|---|---|
| <-<br>=<br><<- | left assignment operators | v1 ← c(3,1) v2 ← c(TRUE, 2+3i)<br>v3 = c(0)<br>Print (v1) O/P: 3 1<br>Print (v2) O/P: TRUE 2+3i<br>Print (v3) O/P: 0 |
| ->>>>> | Right assignment operators | c(3,1) → v1 ; c(TRUE, 2+3i) → v2<br>c(0) → v3<br>Print (v1) O/P: 3,1<br>Print (v2) O/P: TRUE, 2+3i<br>Print (v3) O/P: 0 |

Miscellaneous Operators:

This operators are used for specific purpose and NOT general mathematical or logical computations.

| Operator | Decription | Example |
|---|---|---|
| : | It creates the series of the numbers in sequence for a vector | v ← 2:8<br>Print (v) 2, 3, 4, 5, 6, 7, 8 |
| %in% | This operator is used to identify if an element belongs to a vector the result gives boolean value | v1 ← 8 Print (v1 %in% v)<br>v2 ← 12 O/P: TRUE ;<br>t ← 1:10 FALSE |

This Operator is used to Identify

%*%

M= matrix
(c(2,5,6,1,10,14),
nrow=2, ncol=3, byrow
=TRUE))

t= M%*%t(M)

Print(t)

14

17/03/2022

Functions:

A function is a set of statements organised together to perform a specific task

In R, A function is a object so, the R interpreter is able to pass the control to the function along with the arguments that may be necessary for the function to accomplish the action

R has a large no. of built in functions and user can create there own functions also.

Function Defination:

An R function is created by using the key word function

The basic syntax of an R function is

Syntax:
```
function name ← function (arg1, arg2, ...)
{
  function body
}
```

function components: (The different parts of the fun

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

arguments: An argument is a place holder when a function is invoked you pass a value to the argument.

- arguments are optional, that means a functions may contain no arguments also arguments can have default value.

function body: The function body contains a collection of statements that define what the function does.

return value: The return value of the function is the last expression in the function body to be evaluated

- R has a built in functions which can be directly called in the program without defining them first

- We can also create and use our own functions referred as user defined functions



Built in functions:

The built in functions are:

seq()
sum()
print()
mean() etc..

Eg:- # Create a sequence of numbers from 32 to 44

print (seq (32, 44))

O/p: 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44

# find the mean of numbers from 1 to 5

print (mean (1:5))

$\frac{1+2+3+4+5}{5} = \frac{15}{5}$

O/p: 3

# create sum of numbers from 1 to 5

print (sum(1:5))

O/p: 15

User defined function:

we can create user define functions in R.

There are specific to what a user wants and once created they can be used like the built in function()

Eg:- new.function ← function(a)

```
{
for (i in 1:a)
{
  b ← i^2
} print (b)
}
new function(6)
```

Call the function new.function supplying "6" as an argument

O/p: 1, 4, 9, 16, 25, 36

Calling a function without an argument:

Eg:- new.function ← function()

```
{
for (i in 1:5)
{
  b ← i^2
} print (b)
}
new function ()
```

O/p: 1
4
9
16
25

22/03/2022

Calling a function with Argument values (By position and By name)

The arguments to be a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assign to the name the arguments.

```
Eg: new.function <- function (a,b,c)
    {
     result <- a*b+c
     Print (result)
    }
  # Calling a function by position
    new.function (3,3,1)
  # Calling a function by names
    new.function (a=11, b=5, b=2)
```

=> o/p : 10
         23

Calling a function with default arguments

We can define the value of the arguments in the function defination and call the function without supplying any argumen -ts to get the default value our result. But we can also call search functions by supplying new values of the argument and get non-default result.

```
Eg: new.function <- function ( a=5, b=6, c=7)
    {
     result <- a*b+c
     Print (result)
    }
  # Calling a function without giving any argument
    new.function ( )
  # calling a function with giving new values
    new.function ( 7, 8, 3)
```

o/p : 34, 59

Lazy Evalution function:

Arguments to functions are evaluted lazyly, which means so they are evaluated only when needed by the function body

```
Eg: # Create a function with arguments
    new.function <- function (a, b)
    {
     result <- a^2
     Print (result)
```

```
print(b)
}
new.function(6)
```

o/p : 36
6
Error! b doesn't perform any operation

Error! in print b argument is missing with no default value

Regression Analysis :-

The widely used statistical tool to establish a relationship model between two variables.

One variable is called predictor variable → independent

other variable is called response variable

whose value is derived from the predictor variable. So, in linear regression this two variables are related through an equation where exponent of both these varibles is '1'

mathematically a linear relationship represents a straight line when plotted as a graph.

non-linear relationship where the exponent of any variable is not equal to '1' creates a curve

mathematically equation for a linear regression is $y = ax + b$

a & b are the constant are called co-efficients. x is the predictor variable and y is the response variable.

Steps to establish a regression:

1. Carry out the exponent of gathering a sample of observed values that is nothing but data set.
2. Create a relationship model using the lm() in 'R'
   lm(linear model)
3. Find the co-efficients from the model created and create the mathematical function using these (variables)
4. Get summary of the relationship model to know the average error in prediction also called residentials. To predict the
5. To predict the values use the predict() in 'R'

22/03/2022

Example :-

Below is the sample data representing the observation values of height 151, 174, 138, 186, 128, 136, 179, 163, 152, 131

Values of weights 63, 81, 56, 91, 47, 57, 76, 72, 62, 48.

lm():=

This function creates a relationship between predictor and the response variables

lm (formula, data)

→ formula is a symbol present in relationship between x & y

→ data is a vector on which the formula will be applied

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)

O/P: Call

lm (formula = y~x)

Coefficients

| (Intercept) | x |
|---|---|
| -98.4551 | 0.6146 |

Get the summary:

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)

Print (summary (relation))

O/P: Call

lm (formula = y~x)

Residuals

| Min | 1Q | Median | 3Q | Max |
|---|---|---|---|---|
| -6.3002 | -1.6629 | 0.0412 | 1.8844 | 3.9745 |

Coefficient

| (Intercept) | estimate | Std error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| | -38.45504 | 8.04901 | -4.776 | 0.00139 ** |
| | 0.67461 | 0.05191 | 12.997 | 1.16e06 *** |

Residual standard error 3.253 on 8 degree of freedom

predict():-

24/3/2022

(19)

Syntax: predict (object, new data)

object :-

which is already created using lm()

New data :-

It is a vector containing the new value for predictor variable.

— New page

Regression Analysis of exam (grades):-

-This is a brief statistical regression analysis

-There is not much actual programming in this example but it illustrate how same of the datatype we just discussed are used including R's object.

Ex:-

we have a file exams quiz.txt containing grader form a class

| mid | final | averages |
|---|---|---|
| 2.0 | 3.3 | 4.0 |
| 3.3 | 2 | 3.7 |
| 4.0 | 4.3 | 4.0 |
| 2.3 | 0.0 | 3.3 |

The numbers corresponding to letters on a four point scale (i.e., 3.3 is grade B+ .

→ Each line contains the data for one student

→ consisting of the mid term examination, final examination and average quiz grade.

Reading the datafile :-

```
> Examsquiz ← read.table ("Example.txt", header = FALSE")
```

→ In our file doesn't include a header line naming the variable in each student record. so, we specify header = FALSE in the function column.

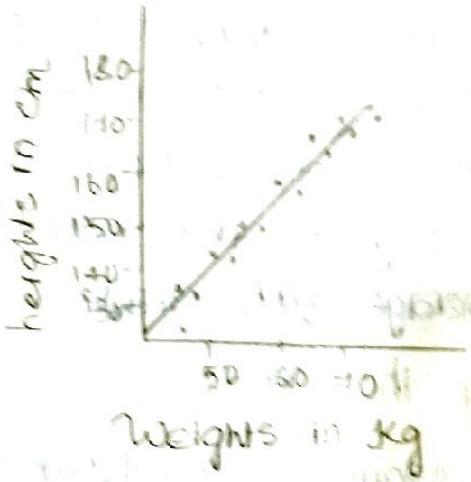
Continuous on 24/03/2022

```
# predict weight of new person
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)
   a ← data frame (x = 170)
   result ← predict (relation a)
   Print (result)
```

O/p: 1
76.22

Visualize the regression graphically:

```
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)
Png (file = "linear regression.png")
plot (y, x, col = "blue", main = "height & weight regression")
Obline (lm (x~y), col = 1.3, pch = 16, xlab = "weight in kgs",
ylab = "height in cms")
# save the file filename.off
```



Now data is in examsquiz, which is in R object of class data.frame > class(examquiz) (21)

o/p: "data frame"

Check that the file was reading correctly

> head(examquiz)

| | $V_1$ | $V_2$ | $V_3$ |
|---|---|---|---|
| o/p: 1 | 2.0 | 3.2 | 4.0 |
| 2 | 3.3 | 2.0 | 3.1 |
| 3 | 4.0 | 4.3 | 4.0 |
| 4 | 2.3 | 0.0 | 3.3 |

- lacking a header for the data, are named as columns ($V_1, V_2, V_3$)
- row numbers appear on the left

To predict the exam2 score (given in the 2nd column of examquiz) from exam1 exar (first column)

lma ← lm(examquiz[,2] ~ examsquiz[,1])

the lm function call tasre instructs or to fit this prediction equation.

Predicted exam2 = $\beta_0 + \beta_1$ exam1

$\beta_0, \beta_1$ are constants to be estimated from over our data.

→ The exam1 scources which are stored in 1st column of our data frame are collectively refer to as examsquiz[,2] that means the exam2 scoures are similarly refered. Cal

→ call to lm() above predicts the 2nd column of examquiz from the 1st. we also could have a written

lma ← lm(examsquiz$V2 ~ examsquiz$V1)

→ The result return by lm() are now in an Object that we have stored in the variable lma.

→ It is an instance of the class we can list its components by calling attributes function. (attribute())

> attributes(lma)

$names

[1] "coefficients" "residuals" "effects" "remains"

[5] "fitted.values" "assign" "qr" "df.residuals"

[9] "xlevels" "calls" "term" "model"

$class ↵ "lm"

> lma$ coef
(Intercept ) examquiz [,1]
1.1205209 0.5899803

lma$ coef - is a vector, printing it is simple when you print object lma itself

> lma
Call:
lm (formula = examsquiz [,2] ~ examsquiz [,1])
Coefficients
(Intercept ) examsquiz [,1]
lm 1.12 0.581

whos job is to print objects of class lm the print lm() and that is what the function displays. we can get a more detailed printe out of the contents of lma by called

Summary

It triggers a call to summary lm behind the scenes, we get a reggression specific summary

> Summary (lma)
Call:
lm (formula = examsquiz [,2] ~ examsquiz [,1])
Residuals

| Min | 1Q | median | 3Q | max |
|---|---|---|---|---|
| -3.4804 | -0.1239 | 0.3426 | 0.7261 | 0.7261 | 1.2225 |

Coefficient : estim std. error tvalue Pr(>|t|).
(Intercept ) 1.1205 0.6375 1.758 0.08709
examsquiz . 2 [,1] 0.5900 0.2030 2.907 0.00610

To estimate a prediction equation for exam2 from both the exam1 and exam2 quiz scores use the "+" notation

> lmb ← lm (exam squiz [,2] ~ examsquiz [,1] + examsquiz [,3])

The '+' doesnot mean that we compute the sum of the two quantities. It is "merely" a delimiter in our list of predictor variables

## Startup and Shutdown

R's behaviour can be customised using startup files

- If they are R commands that you would like to execute at the beginning of every R section. we can place them in a file called .Rprofile located either in our computer home directory or in the directory. from which we are running R

eg:- To set the text editor that invokes if you call edit() we can use a line in .Rprofile

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

- R's option() is used for configuration
- In that .Rprofile on Linux machine at home.
  - .libpaths ("/home/nm/R")

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

we can always set check our current directory by wd(--)

> getwd()

we can our working directory by calling >setwd() with the desire directory as a corted argument eg:- >setwd("q")

- we saved work space in in a file named .RData. which is located either in the directory from which we invoked the R section (Linux) or in R installation directory (windows)

.Rhistory file:

Which records our commands to remind ourself how that work space was created.

R--Vanilla:

which can skip loading all those files and saving of our section at the end by running R that means want to speedup startup & shutdown

- we can find more information about startup file by using

>?startup or > help('startup')

26/03/2022

Getting help

In R's built in help facilities and then those available on the Internet.

help() function:- To get online help, invoke help.

> help()

For instance to get information on the seq() > help(seq)

The shortcut to a help() is a ? > ?seq.

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

To get help on the < operator > ?"<"

To say about for loops > ?"for"

Example() function:-

The example() will actually run some examples,

> example (seq)

Seq > seq (0, 1, length.out = 11) o/p: 0, 1.0, 2.0, 3.0, 4, 5, 6, 7, 8, 9, 10, 11

Seq > seq (start = 1, norm (20) o/p: 1, 2, 3 - - - 20

Seq > seq (1, 9, by = 2) o/p: 1, 3 5 7

Seq > seq (1, 6, by = 3) o/p: 1 4

Seq > seq (1.575, 5.125, by = 0.05) o/p: 1.575, 1.629 - -

Seq > seq (7) o/p: 1, 2, 3 4 5 6 7

The sequence function generates various kinds of numeric sequence in arithematic progression.

In R's excellent graphic functions does the example function will give a graphic illustion

Example (persp):

This displays a series of same graphs for the persp() function we press enter in the R console when we are ready to go to the next one

Note that the code for each example is shown in the console so, we can experiment by tweaking the arguments.

27/03/2022

Vectors:

Vectors are the most basic R' data objects and there are six types of automic vectors they are:

logical, integer, double, complex, character & Raw.

Vector Creation:-

Single Vector Creation:

Even when we write just one value in R, it becomes vector of length '1' and belongs to one of the vector types.

1. print ("abc") - type 'automic vector of type-character.
2. print (12.5) - double value or Numeric
3. print (5L) - automic vector of type - Integer
4. print (TRUE) - automic vector of type - logical
5. Print (2+3i) - complex. o/p: 2+3i

6. print charToRaw ("Hello") → Raw.
O/p: 68 65 53 6c 35

Multiple element vectors:

Using : (colun) operator with Numeric data

eg:- # create a sequence from 5 to 13

a ← (5:13) O/p: 5 6 7 8 9 10 11 12 13
Print (a)

# Using sequence of : operator from 3.8 to 11.2

x ← (3.8 : 11.2) x ← 6.6 : 11.2 Print(x) #

seq ← seq (3.8, 1, length.out = 11.2) → # using sequence()

O/p: 3.8 4.8 5.8 6.8 7.8 8.8 9.8 10.8

#using sequence of : operator from 3.8 to 11.2

x ← 6.6 : 11.2 Print (x)

O/p: 6.6 7.6 8.6 9.6 10.6

# using sequence function.

Seq (0, 3, length.out = 10)

O/p: 0 3 6 9

# Create vector with element from 5 to 9 incrementing by 0.4

Print (seq( 5, 9, by = 0.4))

O/p: 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0

# using the c function.

The non character values are concentrated to character type. If one of the element is a character.

S ← c ("apple", "class", TRUE)

Print (s)

O/p: apple class TRUE

Print (class (s))

O/p: Character

# Accessing vector elements:

Elements of a vector R accessed using indexing. the square brackets are used for indexing. indexing starts with position one. In this a negative value in the index drops elements from result.

Result: TRUE, FALSE OR 0 AND 1
Can also be used for Indexing.

29/03/2022

Ex: a ← c (Sun, Mon, Tue, Wed, Thu, Fri, Sat)
Print (a[1, 2, 6])
a ← c (TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE)
Print (a)
Print (a[-2, -5]) → except 2 and 5 i.e MON, Thu are displayed)

Op: [1] Sun, Mon, Fri
[1] Sun, Fri
[1] Sun, Mon, Wed, Fri, Sat
y ← [c(0, 0, 0, 0, 0, 0, 1)]
Print (y)
O/p: [1] Sat

Salar: Scalars are typically constructed with compounds such as arrays, mat's, sets structures, etc. A scalar As arrays A Scalar is a data structure is the basic datatype that holds only a single atomic value at a time. using scalars more complex datatypes are constructed Most commonly used scalar types in R' integers, Numerics, Floating, characters, complex etc.
It states a single value then we used different scalars.

Ex:- a ← 9
b ← a+1 (b=10)
b ← b-a (b=1)
Print (b)
O/p: [1] 1

In Scalar any type of object can be overweight.

Types of scalars:

Here we are discuss only the most familiar scalars They are: Integer, Characters, float, complex, Array (m x n), Complex, Numeric.

Ex: Integer

```
a ← 5L
Print (a)
a ← 's'
Print (a)
a ← 6+3i
Print (a)
```

```
a ← 12-c
Print (a)
a ← c (Jan, Feb, Mar, Apr, May)
Print (a[1,2,5])
```

Arrays:

These are R data objects which can store data in more than 2D. Array can store only data type An array is created using the array function (array()) it takes vectors as input and used the values in the dim- parameter to create an array

Ex: If we create an array of dim (2,3,4) then it creates 4 rectangle matrices each with 2-rows, 3-columns.

→ Two (3×3) matrices each with 3-rows & 3-columns

```
V1 ← c(5, 9, 3)
V2 ← c(10, 11, 12, 13, 14, 15)
result ← array(c(V1, V2), dim = c(3,3,2))
print(result)
```

O/P:

```
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

Naming Columns & Rows

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

Ex: V1 ← c(5, 9, 3)

V2 ← c(10, 11, 12, 13, 14, 15)

```
Column.names ← c("col1", "col2", "col3")
Row.names ← c("row 1", "row 2", "row 3")
matrix.names ← c("mat 1", "mat 2"
Result ← array (c(V1, V2), dim = c(3,3,2),
dimnames = list (row.names, column.names, matrix.names))
Print (result)
```

o/p

| | mat 1 | | | mat 2 | | |
|---|---|---|---|---|---|---|
| | col 1 | col 2 | col 3 | col 1 | col 2 | col 3 |
| row 1 | 5 | 10 | 13 | 5 | 10 | 13 |
| row 2 | 9 | 11 | 14 | 9 | 11 | 14 |
| row 3 | 3 | 12 | 15 | 3 | 12 | 15 |

Print (result (2,,2) row 1, ROW 2

Print (result (1,3,1) - o/p row: col 1 = 5, col 2 = 10, col 3 = 13

Print (result (3,1,1) → o/p: col 1; row 1 5, row 2 9, row 3 3

→ Print (result (3,,2)

Print 3rd row of 2nd matrix

30/03/2022

## Manipulating array elements.

An array is made up of matrices in multiple Dimensions the operations on elements of array are carried out accessing elements of the matrix

Ex: $V_1 \leftarrow c(5,9,3)$

$V_2 \leftarrow c(10,11,12,13,14,15)$

array 1 ← array (c($V_1$, $V_2$), dim = c(3,3,2))

$V_3 \leftarrow c(8,1,10)$

$V_4 \leftarrow (6,0,11,3,14,1,2,6,9)$

array2 ← array (c(v3, v4), dim = c(3,3,2))

matrix1 ← array1 [,,2], matrix number

matrix2 ← array2 [,,2]

result ← matrix1 + matrix2

Print (result)

op:- $M_1$ $M_2$

$$\begin{bmatrix} 10 & 20 & 26 \\ 18 & 22 & 28 \\ 6 & 24 & 30 \end{bmatrix} + \begin{bmatrix} 9 & 6 & 3 \\ 1 & 0 & 14 \\ 10 & 11 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 9 & 6 \\ 6 & 1 & 10 \\ 9 & 10 & 11 \end{bmatrix} = \begin{bmatrix} 21 & 35 & 35 \\ 25 & 23 & 42 \\ 25 & 45 & 42 \end{bmatrix}$$

Note:- In dimensions we use dimes function in that we mention number of rows, number of columns, no. of matrices for instances, dim (3,3,2) means no. of rows = 3; no of columns = 3.

no. of matrices are = 2.

→ But, in operations we mention any column, row and matrix called number = 2 and matrix number = 1

→ calculating across array elements we can do calculations across the elements in an array using the apply() function.

Syntax: apply (x, margin, fun)
(x ↑ data)

x is an array, margin is the name of the date with fun is the function to be applied across the elements of the array

Ex: $v_1 \leftarrow c(5, 9, 3)$

$v_2 \leftarrow c(10, 11, 12, 13, 14, 15)$

new.array ← array (c($v_1$, $v_2$), dim = c(3,3,2))

Print (new.array)

result ← apply (new.array, x[1], sum) (only one row ↑)

Print (result)

$$\begin{bmatrix} 5 & 10 & 13 \\ 9 & 11 & 14 \\ 3 & 12 & 15 \end{bmatrix} \begin{bmatrix} 5 & 10 & 13 \\ 9 & 11 & 14 \\ 3 & 12 & 15 \end{bmatrix}$$

o/p: [5+5+10+10+13+13  9+9+11+11+14+14  3+3+12+12+15+15]

= [56  68  60],,

Matrix:

31/03/2022

Matrices are the 'R' objects in which the elements are arranged in a 2D Rectangular layout, they contain elements of the same Atomic type. We can create a matrix containing only characters and only logical values, They are not of much use. We use matrices containing numbers and elements to be used in mathematical calculations. A matrix can be created by using matrix() function.

Syntax:

matrix (data, nrow, ncol, by row, dimnames)

data is the input vector which becomes the data elements of the matrix. nrow is the no.of rows to be created ncol is the no.of columns to be created and by row is a logical clue if by row is 'true' then the input vector elements are arranged by row. otherwise, it is arranged in columns. 'dimnames' is the names assigned to the rows and columns

Eg: matrix (data, nrow, ncol, by row, dimnames)

M ← matrix (c(3:14), nrow = 4, by ncol = 3, by row=TRUE)

Print (M)

N ← matrix (c(3:14), nrow = 4, ncol=3, by row = FALSE)

Print (N)

Output:

M

| | [,1] | [,2] | [,3] |
|---|---|---|---|
| [1,] | 3 | 4 | 5 |
| [2,] | 6 | 7 | 8 |
| [3,] | 9 | 10 | 11 |
| [4,] | 12 | 13 | 14 |

N

| | [,1] | [,2] | [,3] |
|---|---|---|---|
| [1,] | 3 | 7 | 4 |
| [2,] | 4 | 8 | 12 |
| [3,] | 5 | 9 | 13 |
| [4,] | 6 | 10 | 14 |

```
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
p ← matrix (c(3:14), nrow = 4
  by row = TRUE, dimnames = list (rownames, colnames))
Print (p)
```

Output:-

| | col1 | col2 | col3 |
|---|---|---|---|
| row1 | 3 | 4 | 5 |
| row2 | 6 | 7 | 8 |
| row3 | 9 | 10 | 11 |
| row4 | 12 | 13 | 14 |

Accessing the elements of matrix:

Elements of matrix can be accessed by using col or row Index of the elements we consider the matrix P

```
Ex: M ← matrix (c(3:14), nrow = 4, ncol = 3, by row = TRUE)
N ← matrix (c(3:14), nrow = 4, ncol = 3, by row = FALSE)
rownames = c("row1", "row2", "row3", "row 4")
colnames = c("col1", "col2", "col3")
p ← matrix (c(3:14), nrow = 4, ncol = 3, by row = TRUE,
      dimnames = list (rownames, colnames))
Print (p[1,3])
Print (p[4,2])
```

Output: col3

row1 5

Output: col2

row4 13

Matrix computation:

Various mathematical operations are performed on matrices using the OR operator. The dimension (no. of rows & columns) should be same for the matrix involved in the operation.

Matrix addition & substraction.

- Matrix 1 ← matrix (c(3,9,-1,4,2,6), nrow=2, ncol=3)
  Matrix 2 ← matrix (c(5,2,0,9,3,4), nrow=2, ncol=3)
  Print (matrix 2)
  result ← matrix1 + matrix 2
  Cat ("Result of addition ", "\n")
  Print (result)
  result ← matrix 1 − matrix 2
  Cat ("result of subtraction ", "\n")
  Print (result)

Output:- matrix 1

| | [,1] | [,2] | [,3] |
|---|---|---|---|
| [1,] | 3 | -1 | 2 |
| [2,] | 9 | 4 | 6 |

matrix 2

| | [,1] | [,2] | [,3] |
|---|---|---|---|
| [1,] | 5 | 0 | 3 |
| [2,] | 2 | 9 | 4 |

Result:

| | [,1] | [,2] | [,3] |
|---|---|---|---|
| [1,] | 8 | -1 | +5 |
| [2,] | 11 | 13 | 10 |

Result:-

| | [,1] | [,2] | [,3] |
|---|---|---|---|
| [1,] | -2 | -1 | -1 |
| [2,] | -1 | -5 | 2 |

common vector Operations 01/04/2022

In R their are various operations which is performed on the vector we can add, subtract, multiply or divide two or more vectors from each other (33)

vector operations.

1. combing vector
2. Arithematic Operations.
3. logical index vector
4. Numeric Index Vector
5. Dupicate Index
6. Range Indexes
7. Out of order Indexes
8. Named vectors

1. combing vector:

The c() function is not only used to create a vector, it is also used for to combine two vectors. By combining two or more vectors it forms a new vector which contains all the elements of each vector.

Eg: p<-c(1,2,4,5,7,8)
q<-c("add","sub","Mul","dIV")
r<-c(p,q)

O/P: 1 2 4 5 7 8
"add" "sub" "mul" "dIV"

2. Arithematic operation:

we can perform all the arithmatic operations on vector. the AO are performed by number by number or divide two vectors

Eg:- a<-c(1,2,3,4,5)
b<-c(6,7,8,9,10)

> a+b O/P 7 9 11 13 15 > a/b O/P $\frac{1}{6}$ $\frac{2}{7}$ $\frac{3}{8}$ $\frac{4}{9}$ $\frac{15}{10}$
> a-b O/P -5 -5 -5 -5 -5 > a%%b O/P .

3. logical Index Vector:

In R we can perform a new vector from a given vector. this vector has the same length as the original vector. the vector members are TRUE only when the corresponding members of the original vector are included in the slice Otherwise it will be FALSE

Eg:- a<-c("add","sub","mul","div")

b<-c(TRUE, FALSE, TRUE, FALSE)

a[b]

o/p: "add" "mul"

4. Numeric Index Vector: In R we specify the index b/w square brackets [] for indexing a numerical value. If our index is negative it will return all the values except for the index which we have specify

Eg: a<-c("add","sub","mul","div")

a[-1] o/p: "sub" "mul" "div"

a[4] "div"

a[-2] "add" "mul" "div"

a[10] "NULL"

5. Duplicate Index: An index vector allows duplicate values which means we can access one element twice in one operation

Eg:- a<-c("add","sub","mul","div")

b[c(2,4,4,3)]

o/p:- sub div div mul

6. Range Indexes: Range index is used to slice our vector to form a new vector. For slicing we use colon operator

• Range indexes are very helpful for the situation involving in a large operators.

Eg:- a<-c("add","sub","mul","div")

b<-a[1:3]

o/p:- "add" "sub" "mul"

7. Out of order Index: In R the index vector can be out of order in which a vector slice with the order of first and second values reserved (fixed)

Eg:- a<-c("add","sub","mul","div")

a[c(4,3,2,1)]

o/p: "add" "sub" "div" "mul" "sub" "add"

8. Named vector numbers:

We first create our vector of characters as

```
a ← c("add", "sub")
> a
o/p: add sub
```

Once our vector characters are created we name the first vector member has * and 2nd member as end (start)

```
> name(a) ← ("start", "End")
> a
o/p :=  Start  End
        add    sub
```

a[start] → o/p add
it only gives the start values.

- we retrive the 1st member by its name as follow above eg:-
- we can revese the order with the help of character string index vector

```
a[c("end", "start")]
```

o/p: end start
sub add.

Vector manupulation:

Vector arthematic:=

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

```
eg:- > V1 ← c(3,8,4,5,0,11)
     > V2 ← c(4,11,0,8,1,2)
     > add.result ← V1 + V2
     > print(add.result)   o/p: 7 19 4 13 1 13
     > sub.result ← V1 - V2
     > print(sub.result)   o/p: -1 -3 4 -3 -1 9
     > mul.result ← V1 * V2
     > print(mul.result)   o/p: 12 88 0 40 0 22
     > div.result ← V1 / V2
     > print(div.result)   o/p: 0.75 0.72 Inf 5 0.0 5.5
```

Vector element recycle: If we apply Arithmatic operations to the vectors of unequal length then the elements of the shorter vector are recycle to complete the operations

```
eg:- > V1 ← c(3,8,4,5,0,11)
     > V2 ← c(4,11)
```

> add ·result ← v1 + v2

O/p: 1 19 8 16 4 22

07/04/2022

Vector element sorting

Elements in the vector can be sorted using the sort() function

Eg:, v ← c(3, 8, 4, 5, 0, 11, -9, 304)

Sort ·result ← sort(v)

Print (sort·result) O/p: -9 0 3 4 5 8 11 304

revsort·result ← sort (v, decreasing = TRUE)

Print (revsort.result) O/p: 304 11 8 5 4 3 0 -9

V ← c("Blue", "Red", "Yellow", "Voilet")

Sort. result ← sort (V)

Print (sort.result). O/p: Blue Red Voilet Yellow

revsort.result ← sort (v, decreasing = TRUE)

Print (revsort:result) O/p:; Yellow Voilet Red Blue

Applications of vectors:

- In machine learning for principle component analysis vectors are used. They are extended to eigen values eigen vectors and then used for performing decomposition in vector space.
- The inputs which are provided to the deep learning model or in the form of vectors, this vector consist of standadised data which is supplied to the input layer of the neural network
- In the development of support vector machines algorithms vectors are used (for classifications)
- Vector operations are utilized in newtal networks for various operations like image recoginzation and text processing

Vectorized Operators :-

08/04/2022

One of the most effective way to achive speed in R code is to use operations that are vectorized means the function applied to a vector is actually applied individually to each element

Vector IN Vector OUT

Eg: U ← c(5, 2, 8)

V ← c(1, 3, 9)

U > V O/p: TRUE FALSE FALSE

eg:- Sqrt(1:9) o/p: $\sqrt{1}$ $\sqrt{2}$ $\sqrt{3}$ $\sqrt{4}$ $\sqrt{5}$ $\sqrt{6}$ $\sqrt{7}$ $\sqrt{8}$ $\sqrt{9}$

y ← c(1.2, 3.9, 0.4)

z ← round(y) o/p: 1 4 0

eg:- y ← c(12, 5, 4)

y+4 o/p: 16 9 8

VECTOR IN MATRIX OUT:

The vectorized function we have been working with so far have scalar return values. calling square root() function on a no. given as a number. If we apply this function to an 8 element vector with we get 8 numbers. the another 8 elements are outputs.

Eg: x ← matrix Syn: rep() → (no, no.of tyms)

c ← matrix (1:4, 2, 2) o/p: $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

> y ← matrix (rep [10, 4], 2, 2) o/p: $\begin{bmatrix} 10 & 10 \\ 10 & 10 \end{bmatrix}$

> x+y o/p: $\begin{bmatrix} 11 & 13 \\ 12 & 14 \end{bmatrix}$

NA & NULL (Not available (missing value), Nothing (NULL)):

In statistical data set we often encounter missing value which we represent in R with the value NA, NULL. on other hand represent that value in question simply does not exist. rather then being enustant but unknown

NA: NA is a logical constant of length 1 which contains a missing value indicator.

Eg: x ← (88, NA, 12, 168, 13)

> x o/p: NA

> mean(x)

mean (x, na.rm = TRUE) o/p: 56.2

eg:- x ← c(5, NA, 12)

mode (x[1]) o/p: NA

y ← c("abc", "efg")

mode (y[1]) o/p: "abc" "efg"

NULL: NULL represents the NULL object in R. NULL is used mainly to represent the list with zero length. and is often return by expression and functions. whose value is undefined

Eg:- z ← NULL
> for (i in 1:10)
> if (i%%2 == 0)
> z ← (z,i)  o/p: 1 2 3 4 5 6 7 8 9 10

Eg:- rv ← c(1,2,3,4,5)
> rv
cat("Vector to NULL")
rv ← NULL

To Function in NULL
- Is.NULL
- As.NULL

Is.NULL: The Is.NULL is an inbuilt function in R that checks if the variable has a NULL value or NOT. It returns the logical vector TRUE or FALSE.

Eg:- v ← c(1,2,3,4,5)
> rv
cat("Vector to NULL")
rv ← NULL
Is.NULL(rv)
o/p: TRUE

as.NULL: as.NULL ignores its arguments and returns the value NULL

Eg:- v ← c(1,2,3,4,5)
> rv
cat("Vector to NULL")
rv=NULL/## v=NULL ← as.NULL(rv)
rv=null o/p: 1 2 3 4 5

Filtering:- Another feature reflecting the functional language "Nature of R is filtering" This allows us to extract the vectors elements that satisfies certain conditions. Filtering is one of most common operations in R. As statistical analysis often focus on data that satisfies the conditions of intrest.

Eg:- z ← c(5,2,-3,8)
w ← z(z*z > 8)
> w

Note: The filtering in R takes defaultly TRUE values in the given vectors based on conditions. Gen.

Generating filtering indicators:

```
z<-c(5,2,-3,8)
w<- z (z*z>8)
>w
```

o/p: z<-z [TRUE, FALSE, TRUE, TRUE] (5, 2, -3, 8)

also apply ">" (z*z,8) Vectorization resulting boolean values

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

z<-[c [TRUE, FALSE, TRUE, TRUE]) o/p: 5 -3 8

Filtering with the subset function

filtering can be also be with the subset function when applied to vectors the difference b/w using this function and ordinary filtering lies in the manner in which na values are handled

```
Eg:- x<-c(6,1:3,NA,12)
     x
     [1] 6 1 2 3 NA 12
     > x[x>5]   o/p: 6 NA 12
     > subset(x, x>5)  o/p: 6 12
```

When we which to exclude NA values using subset() saves you the trouble of removing the NA values yourself

Which() function: (prints the position) of array)

filtering consist of extracting elements of a vector z that satisfy a certain condition if we wish to find the positions in z we use the which()

```
Eg:- z<-c(5,2,-3,8)
     which (c(z*z>8))  o/p: 1 3 4
```

Vectorized if_than_else

If_else(): R also indicates a vectorized version. ifelse() form is syntax- ifelse(b,u,v)

where b= boolean vector, u and v are vectors.

```
Eg:- > x<-1:10                    o/p: 12 12 12 12 12 12 12 12 12 12
     > y<-NULL
     y<-ifelse(x%%2==0,5,12) >y
```

```
eg: > x <- c(5, 2, 9, 12)
    > y <- ifelse (x>6, 2*x, 3*x)
    > y          o/p: 15  6  18  24
```

diff():

Which does lag() function for vectors compare each element with the element 3 three of behind it and termed as lag(3)

```
Eg:- > v <- c(1, 6, 7, 2, 3, 5)
     > diff(v)   o/p: 6-1=5   1  -5  1  2
```

sign():

which converts the numbers in its argument vector to 1, 0 or -1 depending on weather they are positive, zero or negative.

```
Eg:- > v <- c(1, 6, 7, 2, 3, 5)
     > diff(v)
     > sign(diff(v))
     o/p: 1  1  -1  1  1
```

all() and any():- 28/04/2022

The any and all functions are short cuts this report whether any or all of there arguments are TRUE. TRUE

```
> x <- 1:10
> all (x>0)
o/p: TRUE
> any (x>8)
o/p: TRUE
> any (x>88)
o/p: FALSE
> all (x>0)
o/p: FALSE
```

```
> x <- (1:10)
> y <- any (x>8)
> y   o/p: 9
```

- any() then reports whether any of these values is TRUE
- all() works similarly and reports if all of the values TRUE