**Introduction to R Language**

Basic R shell commands 🡪

* **help(object)** --> Provides a manual page for the object. Eg. help(seq)
* **help.start()** --> Open Help Menu on Browser ( Graphical help 0 ).
* **source(‘file.r’)** --> It will run file.r (from cwd) file inside your shell.
* **sink(‘output\_data’)** --> It will save all your standard output in output\_data file in current working directory.
* **sink()** --> It will reset your output stream default
* **objects()** --> It will show all the objects (data types) that are present in current session.
* **rm(object1,object2,object3)** --> It will remove object1, object2, object3 from current shell session.
* **q()** --> It is used to quit your R shell. Every time you quit R shell it creates two files on is as .RData and .Rhistory
* **.RData** --> This will save all your data objects present in current session in encrypted form and will be loaded each time when you run shell again.
* **.Rhistory** --> it will save all your commands that you fired in your current session and will load your command history each time you enter in new session.

**Vector and Assignment in R**

* **x = 5** --> It will assign value 5 inside variable x.
* **data <- c( 10.4, 5.6, 3.1, 6.4, 21.7 ) -->** it will create a number vector and assign into data variable. <- is same as = operator.
* **5 -> x -->** It is same as x = 5 or x -> 5 all are known as assignment.
* **c() function -->** it is constructor in R language to create objects every time we will use it to create data objects. Also known as concatenation adding argument end to end.
* **assign( “data”, c( 10.4, 5.6, 3.1, 6.4, 21.7 ) ) -->** it is also used to assign values to variable, same as data = c( 10.4, 5.6, 3.1, 6.4, 21.7 )
* **Examples :-**

**>>** c(10,20,30,35) -> x

**>>** d <- c(x,0,x)

**>>** z = c(x,d)

**>>** print(x); print(y); print(z)#multiple statement in one line

**Some Common Built-in Functions of Python**

>> **sum(obj)** 🡪 will return summation of all number present in object obj.

>> **max(obj) 🡪** will return maximum value from all other values present in object obj.

>> **min(obj) 🡪** will return minimum value from all other values present in data object obj.

>> **mean(obj) 🡪** will return mean value of data object obj.

>> **length(obj) 🡪** will return number of items present inside object obj.

>> **sqrt(obj) 🡪** return a vector object of same length as obj with squre root of each value present in object obj.

>> **prod(obj) 🡪** will give you cross production ( multiplication ) of object obj.

>>**sort(obj) 🡪** will return a copy of sorted object obj in ascending order. For descending order use **sort(obj,decreasing=TRUE)**

>>**order(obj) 🡪** will give you index vector according to ascending order sorting. For descending order index vector use **order(-obj)**, where **–** sign represent the reverse order sort.

>>**median(obj) 🡪** will give you a median of object obj

**Examples :-**

>> x <- c(10,50,20,15,22,33)

>> print(x)

[1]10 50 20 15 22 23

>> sum(x)

[1] 150

>> max(x)

[1] 50

>> min(x)

[1] 10

>> mean(x)

[1] 25

>> length(x)

[1] 6

>> sqrt(x)

[1] 3.162278 7.071068 4.472136 3.872983 4.690416 5.744563

>> prod(x)

[1] 108900000

>> sort(x)

[1] 10 15 20 22 33 50

>> sort(x,decreasing=TRUE)

[1] 50 33 22 20 15 10

>> order(x)

[1] 1 4 3 5 6 2

>> order(-x)

[1] 2 6 5 3 4 1

>> x[order(x)] #same as sort

[1] 10 15 20 22 33 50

**Generating regular sequences**

>> **1:10** will generate a vector similar to c( 1, 2, 3, 4, …, 9, 10 )

Note -> colon operator has higher priority as

>> **2\*1:10** willgive you result similar to c( 2, 4, 6, 8 …, 18, 20 )

>> **1:n-1** will give you python like sequence generation like **1:10-1** will give you vector similar to c( 0, 1, 2, 3, …, 8, 9 )

>> **30:1** may be used to generate a sequence backward

>> **seq()** function is more used to generate sequence objects. Syntax is **seq( from, to, by, length )** where **from** is **starting point**, **to** is **end point**, and **by** works as **jump** or **step** variable, **length** is **no of values** to be generate.

>>**rep()** function is used to repeat (replicate) an object. Syntax is **rep( obj, times=5 )** or **rep(obj,each=5)**

**Examples :-**

>> **seq(1:10)**

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

>> **seq(from=1,to=10,by=2)**

[1] 1 3 5 7 9

>> **seq(from=1,to=19,length=10)**

[1]  1 3 5 7 9 11 13 15 17 19

>> **x <- c( 1, 2, 3, 4, 5)**

**>> rep(x,3) or rep(x,times=3)**

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

**>> rep(x,each=3)**

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

>> **rep(x,1:5)**

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

>> **rep(x,c(1,3,2,1,2)**

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

>> **seq(1,9,by=pi)**

[1] 1.0000000 4.141593 7.283185

>> **seq(15) #same as seq(1:15) #or use seq\_len(15)**

**Random Numbers in R Language**

**runif, rbinom, rnorm**

**>> runif(no\_of\_values,start,end) 🡪** The function uses the continuous uniform distribution meaning that every value between the two end points has an equal probability of being sampled. Here is the code to produce 100 values between 1 and 100. eg **runif(100,1,100)**.

**>> rbinom(iterations,trials,probability) 🡪** will generate binomial distribution random number as **rbinom(100,6,0.5)** will provide you a sample data of 100 events of a dice rolling with probability of 0.5.

**>> rnorm(value,mean,deviation) 🡪** will generate normal distribution data eg. **rnorm(100,50,12.5)**

>> sample function used to generate a random sequence from given data. eg. **sample(99,100,replace=TRUE)** , first argument take 1-99 no, second argument is length , and replace TRUE allow repetition. eg. **data = c(101,110); sort(sample(data,30,replace=FALSE))**