**Run all the codes given on session 3 (Saturday 17 Feb 2024) slide deck in R console and provide interpretation of all the outputs. You will not receive any marks if outputs and its interpretations are not provided.**

* **Basic of R**

> 4\*6+5

**Output**: 29

**Interpret**: Basic Arithmetic Operation

> (4\*6)+5

**Output**: 29

> 4\*(6+5)

**Output**: 44

> (4+6)^2\*5/10+9-1

**Output**: 58

**Interpret:** follows PEMDAS rule in order, Parenthesis, Exponents, Multiplication, Division, Addition and Subtraction. (4+6)=10^2=100\*5=500/10=50+9=59-1=58

* **Variable in R**

Assigning variable

> x<-2

> x

**Output:** 2

> X=3

> X

**Output:** 3

> 4->y

> y

**Output:** 4

> assign("a",11)

> a

**Output:** 11

> age.group<-"Male"

> age.group

**Output:** "Male"

> age\_group<-"Female"

> age\_group

**Output:** "Female"

> \_age<-11

**Output: Error: unexpected symbol in "\_age"**

> 5age<-11

**Output: Error: unexpected symbol in "5age"**

**Interpret:** A variable must not start with underscore and number. ‘x’ and ‘X’ are different variables because R is case-sensitive language.

* **Data types**

1. **Numeric**

> x<-c(1,2,3,4,5,6,7,8,9)

> class(x)

**Output:** "numeric"

> is.numeric(x)

**Output:** TRUE

1. **Integer**

> x<-c(1:9)

> class(x)

**Output:** "integer"

> is.numeric(x)

**Output:** TRUE

Promote “integers” to “numeric”

> class(4L)

**Output:** "integer"

> class(2.8)

**Output:** "numeric"

> 4L\*2.8

**Output:** 11.2

> class(4L\*2.8)

**Output:** "numeric"

**Interpret:** Multiplying integers by numeric promotes to numeric

> class(5L)

**Output:** "integer"

> class(2L)

**Output:** "integer"

> 5L/2L

**Output:** 2.5

> class(5L/2L)

**Output:** "numeric"

**Interpret:** Dividing integer by integer promotes to numeric

> class(4L)

**Output:** "integer"

> class(5L)

**Output:** "integer"

> 4L\*5L

**Output:** 20

> class(4L\*5L)

**Output:** "integer"

**Interpret:** Adding and Multiplying integers does not promote to numeric

1. **Character**

> x<-"data"

> x

**Output:** "data"

> class(x)

**Output:** "character"

> nchar(x)

**Output:** 4

**Interpret: “**nchar” function to count the character in the string

1. **Factor**

Factor is used to categorize data in multiple levels like how gender categorized into levels (male and female), how marital status categorized into levels (unmarried, married).

> gender<-factor(c("male","female","female","male"))

> typeof(gender)

**Output:** "integer"

> attributes(gender)

**Output:**

$levels

"female" "male"

$class

"factor"

> unclass(gender)

**Output:**

2 1 1 2

attr(,"levels")

"female" "male"

**Interpret:** Vector is converted into factor with two different level (male and female) using factor function. Then male is transformed into an integer 2 and female as 1 using unclass function.

1. **Date**

> date1<-as.Date("2024-02-21")

> date1

**Output:** "2024-02-21"

> class(date1)

**Output:** "Date"

> as.numeric(date1)

**Output:** 19774

> date2<-as.POSIXct("2024-02-21 06:00")

> date2

**Output:** "2024-02-21 06:00:00 +0545"

> class(date2)

**Output:** "POSIXct" "POSIXt"

> as.numeric(date2)

**Output:** 1708474500

**Interpret:** as.Date() function is used to store date whereas as.POSIXctl() is used to convert date in specific format with time and time zone

1. **Logical**

> 2==3 FALSE

> 2!=3 TRUE

> 2<3 TRUE

> 2<=3 TRUE

> 2>3 FALSE

> 2>=3 FALSE

> "data"=="stats" FALSE, number of characters in “data” and “stats” are not equal.

> "data"<"stats" TRUE, number of characters in “stats” is greater than “data”

1. **Vector**

Vector are the collection of data with the same data types.

Eg: x<-c(1,2,3,4,5); where c stands for “combine”

1. **Vector of equal length**

> x<-1:10

> y<- -5:4

> x+y

**Output:** -4 -2 0 2 4 6 8 10 12 14

> x-y

**Output:** 6 6 6 6 6 6 6 6 6 6

> x\*y

**Output:** -5 -8 -9 -8 -5 0 7 16 27 40

> x/y

**Output:** -0.2 -0.5 -1.0 -2.0 -5.0 Inf 7.0 4.0 3.0 2.5

> x^y

**Output:** 1.000000e+00 6.250000e-02 3.703704e-02 6.250000e-02 2.000000e-01 1.000000e+00 7.000000e+00 6.400000e+01 7.290000e+02 1.000000e+04

1. **Vector of unequal length**

> x<-1:10

> z<-c(1,2)

> x+z

**Output:** 2 4 4 6 6 8 8 10 10 12

**Interpret:** Here, shorter vector is repeated to match every element in longer vector.And longer vector is multiple of shorter one.

1+1=**2**, 2+2=**4**, 3+1=**4**, 4+2=**6**, 5+1=**6**,6+2=**8**,7+1=**8**,8+2=**10,** 9+1=**10**,10+2=**12**

> x<-1:10

> w<-c(1,2,3)

> x+w

**Output:** 2 4 6 5 7 9 8 10 12 11

Warning message:

In x + w: longer object length is not a multiple of shorter object length

**Interpret:** shorter vector has 3 elements which cannot be matched the multiple to 10 elements of longer vector. So, a warning is shown.

1. **Giving names to a vector**

> c(One="a",Two="y",Last="r")

**Output:** One Two Last

"a" "y" "r"

> w<- 1:3

> names(w) <- c("a","b","c")

> w

**Output:** a b c

1 2 3

* **Missing data in R**

**NA** and **NULL** are used to represent missing data in R.

> z<-c(1,2,NA,8,3,NA,3)

> mean(z)

**Output:** NA

> z<-c(1,2,NULL,8,3,NULL,3)

> mean(z)

**Output:** 3.41

**Interpret: NA** is for “missingness”, and **NULL** is for “nothingness”. So, we cannot get mean value because NA is used for missing value instead NULL value can be used.

> mean(z,na.rm=TRUE)

**Output:** 3.4

> var(z,na.rm=TRUE)

**Output:** 7.3

> sd(z,na.rm=TRUE)

**Output:** 2.701851

**Interpret: “**na.rm” function with TRUE argument will remove missing value.

* **Pipes in R**

Pipes are alternative way to call the function. “Magrittr” package are required. Pipes are effective while using with multiple function call.

> x<-1:10

> library(magrittr)

> x%>%mean

**Output:** 5.5

> z<-c(1,2,NA,8,3,NA,3)

> z%>%mean(na.rm=TRUE)

**Output:** 3.4

* **Dataframe**

Dataframe is the data structure which can store data in tabular format.

> x<-10:1

> y<- -4:5

> q<-c("Hockey","Football","Baseball","Kabaddi","Rugby","Pingpong","Basketball","Tennis","Cricket","Volleyball")

> theDF <-data.frame(x, y, q)

> theDF

**Output:** x y q

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

#Add column names

> theDF<- data.frame(First=x,Second=y,Sport=q)

> theDF

**Output:** First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

> names(theDF)

**Output:** "First" "Second" "Sport"

> rownames(theDF)

**Output:** "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"

#Add row names

> rownames(theDF)<-c("One","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten")

> theDF

**Output:** First Second Sport

One 10 -4 Hockey

Two 9 -3 Football

Three 8 -2 Baseball

Four 7 -1 Kabaddi

Five 6 0 Rugby

Six 5 1 Pingpong

Seven 4 2 Basketball

Eight 3 3 Tennis

Nine 2 4 Cricket

Ten 1 5 Volleyball

> head(theDF) #View few top rows

**Output:**

First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

> head(theDF,n=7) #View 7 rows

**Output:**

First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

> tail(theDF) #View few bottom rows

**Output:**

First Second Sport

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

> theDF[3,2] #View element of 3rd row, 2nd Column.

**Output:** -2

> theDF[3,2:3] #View elements of 3rd row, 2nd and 3rd column.

**Output:** Second Sport

3 -2 Baseball

> theDF[,3] #View 3rd Column only

**Output:** "Hockey" "Football" "Baseball" "Kabaddi" "Rugby" "Pingpong" "Basketball" "Tennis" "Cricket" "Volleyball"

> theDF[3,] #View 3rd row only

**Output:** First Second Sport

3 8 -2 Baseball

> theDF[,c("First","Sport")] #View columns with name “First” and “Sport”

**Output:** First Sport

1 10 Hockey

2 9 Football

3 8 Baseball

4 7 Kabaddi

5 6 Rugby

6 5 Pingpong

7 4 Basketball

8 3 Tennis

9 2 Cricket

10 1 Volleyball

* **List**

A list is the collection of objects with the same or different data types. Argument on list function becomes an element of the list.

> list1<-list(1,2,3) #Three argument inside list function indicate elements.

> list1

**Output:**

[[1]]

[1] 1

[[2]]

[1] 2

[[3]]

[1] 3

> list2<-list(c(1,2,3)) #indicates as single element

> list2

**Output:**

[[1]]

[1] 1 2 3

> list3<-list(c(1,2,3),3:7) # List with two vector as argument

> list3

**Output:**

[[1]]

[1] 1 2 3

[[2]]

[1] 3 4 5 6 7

> list4<-list(theDF,1:10) #List with data frame and vector

> list4

**Output:**

[[1]]

First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

[[2]]

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

> list5<-list(theDF,1:10,list3) #Three element list with data frame, vector and list itself.

> list5

**Output:**

[[1]]

First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

[[2]]

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

[[3]]

[[3]][[1]]

[1] 1 2 3

[[3]][[2]]

[1] 3 4 5 6 7

> names(list5)<-c("data.frame","vector","list") #Naming the list5

> names(list)

NULL

> names(list5)

**Output:**

[1] "data.frame" "vector" "list"

> list5

**Output:**

$data.frame

First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

$vector

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

$list

$list[[1]]

[1] 1 2 3

$list[[2]]

[1] 3 4 5 6 7

#Naming the elements of list5 with “TheDataFrame”,”The Vector” and “TheList”

> list6<-list(TheDataFrame=theDF,TheVector=1:10,TheList=list3)

> names(list6)

**Output:** [1] "theDataFrame" "TheVector" "TheList"

> list6

**Output:**

$TheDataFrame

First Second Sport

1 10 -4 Hockey

2 9 -3 Football

3 8 -2 Baseball

4 7 -1 Kabaddi

5 6 0 Rugby

6 5 1 Pingpong

7 4 2 Basketball

8 3 3 Tennis

9 2 4 Cricket

10 1 5 Volleyball

$TheVector

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

$TheList

$TheList[[1]]

[1] 1 2 3

$TheList[[2]]

[1] 3 4 5 6 7

* **Matrices**

> A<-matrix(1:10,nrow=5)

> A

**Output:**

[,1] [,2]

[1,] 1 6

[2,] 2 7

[3,] 3 8

[4,] 4 9

[5,] 5 10

> nrow(A)

**Output:** 5

**Interpret:** Matrix A with 5 row 2 column, 5X2 dimension.

> B<-matrix(21:30,nrow=5)

> B

**Output:**

[,1] [,2]

[1,] 21 26

[2,] 22 27

[3,] 23 28

[4,] 24 29

[5,] 25 30

> ncol(B)

**Output:** 2

**Interpret:** Matrix B with 5 row 2 column, 5X2 dimension.

> C<-matrix(21:40,nrow=2)

> C

**Output:** [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]

[1,] 21 23 25 27 29 31 33 35 37 39

[2,] 22 24 26 28 30 32 34 36 38 40

> dim(C)

**Output:** 2 10

**Interpret:** Matrix C with 2 row 10 column, 2X10 dimension

> A+B

**Output:**

[,1] [,2]

[1,] 22 32

[2,] 24 34

[3,] 26 36

[4,] 28 38

[5,] 30 40

> A-B

**Output:**

[,1] [,2]

[1,] -20 -20

[2,] -20 -20

[3,] -20 -20

[4,] -20 -20

[5,] -20 -20

#Matrix multiplication is only possible if number of columns of left-hand matrix is same as rows of right-hand matrix

> A%\*%B

**Output:** Error in A %\*% B: non-conformable arguments

**Interpret:** Dimension of matrix A is 5X2, Dimension of matrix B is 5X2, here number of columns on matrix A and number of rows on matrix B is not same. So, matrix multiplication is not possible.

> A%\*%C

**Output:**

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

[1,] 153 167 181 195 209 223 237 251 265 279

[2,] 196 214 232 250 268 286 304 322 340 358

[3,] 239 261 283 305 327 349 371 393 415 437

[4,] 282 308 334 360 386 412 438 464 490 516

[5,] 325 355 385 415 445 475 505 535 565 595

**Interpret:** Dimension of matrix A is 5X2, Dimension of matrix C is 5X10, here number of columns on matrix A and number of rows on matrix C is same. So, matrix multiplication is possible and resulted matrix will be in 5X10 dimension.

* **Array**

Array is the multidimensional vector with same data types.

> theArray<-array(1:12,dim=c(2,3,2))

> theArray

**Output:**

, , 1

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

[1,] 1 3 5

[2,] 2 4 6

, , 2

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

[1,] 7 9 11

[2,] 8 10 12

**Interpret:** 2 dimensional matrices are created with 2 rows and 3 columns.

**# First row of both dimensional matrices.**

> theArray [1, , ]

**Output:**

[,1] [,2]

[1,] 1 7

[2,] 3 9

[3,] 5 11

**# First row of first dimensional matrix**

> theArray[1,,1]

**Output:**

[1] 1 3 5

**# First column of both dimensional matrices.**

> theArray[,1,]

**Output:**

[,1] [,2]

[1,] 1 7

[2,] 2 8