R PROGRAMMING ASSIGNMENT-2

1.Can you write a program that converts a matrix into a one-dimensional array?

ANS:

PROGRAM THAT CONVERTS A MATRIX TO A ONE DIMENSIONAL ARRAY:

```
1. row_names = c("row1", "row2", "row3", "row4")
2. col_names = c("col1", "col2", "col3", "col4")
3. M = matrix(c(1:16), nrow = 4, byrow = TRUE, dimnames =
   list(row_names, col_names))
4. print("Original Matrix:")
5. print(M)
6. result = as.vector(M)
7. print("1 dimensional array (column wise):")
8. print(result)
9. result = as.vector(t(M))
10.      print("1 dimensional array (row wise):")
11.      print(result)
```

INPUT:

[1] “Original Matrix: ”

	Co11	co12	co13	co14
row1	1	2	3	4
row2	5	6	7	8
row3	9	10	11	12
row4	13	14	15	16

Output:

[1] “1 dimensional array(column wise); ”

[1] 1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 16

```
[1]  “1 dimensional array(row wise): ”

 [1]  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16
```

2. What are the functions provided by the R program?

ANS:

R Functions Types

There are two types of functions in R:

- In-built functions
- User-defined functions

In-built Functions

These functions in R programming are provided by the R environment for direct execution to make your work easier to accomplish your needed data.

Some examples for the frequently used in-built functions are as follows:

```
#seq() To create a sequence of numbers
print(seq(1,9))
```

Output:

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

```
#sum() To find the sum of numbers
```

```
print(sum(25,50))
```

Output:

```
 [1] 75
```

```
#mean() To find the mean of numbers
print(mean(41:68))
```

Output: [1] 54.5

```
#paste() To combine vectors after converting them to characters
paste(1,"sam",2,"rob",3,"max")
```

Output:

```
[1] "1 sam 2 rob 3 max"
```

```
paste(1:3, c("sam","rob","max"), sep = '-', collapse = " and ")
#collapse is used to separate the values when a vector is passed
```

Output:

```
 [1] "1-sam and 2-rob and 3-max"
```

```
#paste0() Has default sep="" argument
paste0(1,"sam",2,"rob",3,"max")
```

Output:

```
 [1] "1sam2rob3max"
```

```
paste0(1:3, c("sam","rob","max"), collapse = " and ")
```

Output:

```
 [1] "1sam and 2rob and 3max"
```

```
head(x,n) #To retrieve the first n rows of a matrix, data frame, or a vector
```

```
x = data frame, matrix, or vector
n = number of rows to be retrieved
empid <- c(1:4)
empname <- c("Sam","Rob","Max","John")
empdept <- c("Sales","Marketing","HR","R & D")
emp.data <- data.frame(empid,empname,empdept)
print(head(emp.data,3))
```

Output:

```
      empid   empname     empdept
1         1       Sam       Sales
2         2       Rob   Marketing
3         3       Max          HR
```

```
tail(x,n) #To retrieve the last n rows of a matrix, data frame, or a vector
x = data frame, matrix, or vector
n = number of rows to be retrieved
empid <- c(1:4)
empname <- c("Sam","Rob","Max","John")
empdept <- c("Sales","Marketing","HR","R & D")
emp.data <- data.frame(empid,empname,empdept)
print(tail(emp.data,2))
```

Output:

```
      empid    empname     empdept
3         3        Max          HR
4         4       John       R & D
```

```
min() To return the minimum value from a vector.
x <- c(3,45,6,7,89,9)print(min(x))
#max() To return the maximum value from a vector
x <- c(3,45,6,7,89,9)
```

```
print(max(x))
```

Output:

```
[1] 3
[1] 89
```

```
#range() To return the minimum and maximum values from a vector
x <- c(3,45,6,7,89,9)
range(x)
x <- c(3,45,-6,7,89,Inf,9)
range(x)
x <- c(3,45,-6,7,NA,89,Inf,9)
range(x)
x<- c(3,45,-6,7,NA,89,Inf,9)
range(x, na.rm = TRUE)          To remove NA from the result (na.rm) is used
```

Output:

```
[1]  3 89
[1]  -6 Inf
[1] NA NA
[1]  -6 Inf
```

```
#which.min() To return the index of the minimum value from a vector
x <- c(3,45,6,7,89,9)
print(which.min(x))
#which.max() To return the index of the maximum value from a vector
x <- c(3,45,6,7,89,9)
print(which.max(x))
```

Output:

```
[1] 1
[1] 5
```

User-defined Functions

These functions in R programming language are declared, and defined by a user according to the requirements, to perform a specific task.

For example:

```
#Create a function to print the sum of squares of numbers in sequence
sum = 0
Function1 <- function(x) {
for(i in 1:x) {
a <- i^2
 sum = sum + a
print(sum)
}
}
#Calling a function
Function1(5)
```

Output:

```
[1] 1
[1] 5
[1] 14
[1] 30
[1] 55
```

```
#Function without an Argument
sum = 0
Function1 <- function() {
for(i in 1:5) {
```

```
a <- i^2
sum = sum + a
print(sum)
}
}
#Calling a function
Function1()
```

Output:

```
[1] 1
[1] 5
[1] 14
[1] 30
[1] 55
```

```
#Create a function to print the sum of squares of three numbers
Function1 <- function(x,y,z) {
sum = x^2 + y^2 + z^2
print(sum)
}
}
#Calling a function
Function1(2,4,6)                    #Call by position of arguments
Function1(x=3,y=5,z=7)              #Call by names of arguments
```

Output:

```
[1] 56
[1] 83
```

```
Function1 <- function(x=4,y=5,z=6) {
    sum = x^2 + y^2 + z^2
```

```
    print(sum)
}
#Calling a function
Function1()              #Calling a function with default arguments
Function1(6,7,8)          #Passing new values to replace default arguments
```

Output:

```
[1] 77
[1] 149
```

4.What are the R language's limitations?

ANS:

Limitations of R Programming

R programming is beloved by academics and researchers as well because of the state-of-the-art tools in data science and analysis that it offers. Not only does its open-source nature ensure that contributors to the project are able to come out with packages that enable R programming to sport the latest advancements in its field but also makes it an option that can be used to burn a hole. Can be applied within the pocket of loved ones.

Good for prototyping

However, this disadvantage rarely comes in the case of prototyping or if you are creating models that serve as proof of concept because in such cases the data sets are comparatively small. This is a huge factor when building machine learning systems at the scale or scale of enterprises. Such large corporate organizations use R as a sandbox of their data to

experiment with new models or methods related to machine learning. If one such experiment shows promise or in other words is successful, the company's engineers usually attempt to replicate the particular functionality that originally existed in the R programming language in a language that SAS or other software publishers offer. is more suitable as Analysis and data science field.

Ground-level

We may dare to end this post with the observation that the large-scale enterprises that use SAS and R programming share the same ground and one is as important as the other

4.What is the best way to write commands in R?

ANS:

The Easy Way:

The easiest way is to click on the "**Import Data set**" button at the upper right in the RStudio console and then browse your computer and follow the instructions. If the dialog asks if the data set has a header, say "yes", since this data set and others for this course have headers (i.e., titles) for their columns.

- NOTE #1: In the newest version of R clicking on the "Import Data set" tab produces a short drop down menu from which you should choose the second (readr) selection for CSV file. You will then be prompted to install an additional package into R.
- NOTE #2: If you are using Chrome as your browser, you may have problems importing data sets into R. If so, try using Safari or Firefox.

Once the file has been imported, we usually give it a short nickname to cut down on typing:

```
> fram <- framstudy
```

"fram" is a short name or a nickname that I made up for the data set to reduce typing during programming. The "less than" character followed by a dash (<-) looks a bit like an arrow and functions as an assigner to tell R that we are assigning a name to a data set we are importing.

5.What does the dim() function do?

ANS:

DIM: The dim() is an inbuilt R function that either sets or returns the dimension of the matrix, array, or data frame. The dim() function takes the R object as an argument and returns its dimension, or if you assign the value to the dim() function, then it sets the dimension for that R Object.

Syntax:

```
# To get the dimension value
dim(data)


# To set the dimension value
dim(data) <- value
```

Parameters:

The data is an input R Object whose dimension we have to get

If we want to assign a new dimension, then use the second syntax.

Example:

```
df <- data.frame(c1 = c("a", "b", "c", "d"),
 c2 = c("e", "f", "g", "h"),
 c3 = c("i", "j", "k", "l"),
 c4 = c("m", "n", "o", "p"),
 c5 = c("q", "r", "s", "t"))

df
cat("The dimension of data frame is: ", "\n")
dim(df)
```

Output:

```
  c1 c2 c3 c4 c5
1  a  e  i  m  q
2  b  f  j  n  r
3  c  g  k  o  s
4  d  h  l  p  t
The dimension of data frame is:
[1] 4 5
```

From the output, you can see that our data frame has (4, 5) dimensions. We have four rows and five columns.

For the data frame, the **`dim()`** function returns the number of rows and columns as an integer vector.

The functions **dim()** and **dim<-** are generic.

6.Using the dim() functions, generate a 3-dimensional array of 24 elements in asimple application?

ANS:

R Programming Code :

```
v =  sample(1:5,24,replace = TRUE)

dim(v) = c(3,2,4)

print("3-dimension array:")

print(v)
```

Output:

```
[1] 3-dimension array

, , 1

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

, , 2

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

 , , 3
```

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

     [,1] [,2]
[1,]    3    3
[2,]    1    5
[3,]    1    2
```

7.What is the best way to preserve data in R programming?

ANS: You can save an R object like a data frame as either an RData file or an RDS file. RData files can store multiple R objects at once, but RDS files are the better choice because they foster reproducible code.

To save data as an RData object, use the `save` function. To save data as a RDS object, use the `saveRDS` function. In each case, the first argument should be the name of the R object you wish to save. You should then include a file argument that has the file name or file path you want to save the data set to.

For example, if you have three R objects, `a`, `b`, and `c`, you could save them all in the same RData file and then reload them in another R session:

```
a <- 1
b <- 2
```

```
c <- 3
save(a, b, c, file = "stuff.RData")
load("stuff.RData")
```

However, if you forget the names of your objects or give your file to someone else to use, it will be difficult to determine what was in the file—even after you (or they) load it. The user interface for RDS files is much more clear. You can save only one object per file, and whoever loads it can decide what they want to call their new data. As a bonus, you don't have to worry about `load` overwriting any R objects that happened to have the same name as the objects you are loading:

```
saveRDS(a, file = "stuff.RDS")
a <- readRDS("stuff.RDS")
```

Saving your data as an R file offers some advantages over saving your data as a plain-text file. R automatically compresses the file and will also save any R-related metadata associated with your object. This can be handy if your data contains factors, dates and times, or class attributes. You won't have to reparse this information into R the way you would if you converted everything to a text file.

On the other hand, R files cannot be read by many other programs, which makes them inefficient for sharing. They may also create a problem for long-term storage if you don't think you'll have a copy of R when you reopen the files.