Acadgild – Data Analytics – Batch 4 Assignment

SESSION: 1 To 5

Task 1:

1. How many ways are there to call a function in R?

Soluion:

a. call(name, ...)

where, name: is a a non-empty character string naming the function to be called ... stands for arguments to be part of the call

call returns an unevaluated function call, that is, an unevaluated expression which consists of the named function applied to the given. Although the call is unevaluated, the arguments ... are evaluated.

b. do.call

do.call constructs and executes a function call from a name or a function and a list of arguments to be passed to it.

c. Recall

Recall is used as a placeholder for the name of the function in which it is called. It allows the definition of recursive functions which still work after being renamed

2. What is the Recycling of elements in a vector?

Solution:

Recycling occurs when vector arithmetic is performed on multiple vectors of different sizes. R takes the shorter vector and repeats them until it becomes long enough to match the longer one.

For example:-

CASE I: When the length of shorter vector divides evenly into the length of longer vector

The output of the R-Script (from Console window) is given as follows:

```
> a <- c(10,2,23,4)+c(2,10)
> a
[1] 12 12 25 14
```

The output is obtained by recycling the shorter vector c(2,10) until its length is same as the longer vector c(10,2,23,4). The vector c(2,10) vector repeated itself to form c(2,10, 2,10) so that it could successfully match the previous term.

So vector a is obtained as: a <- (10+2, 2+10, 23+2, 4+10)

CASE II: When the length of shorter vector does not divide evenly into the length of longer vector, R will still apply the recycling method, but will throw a warning.

The output of the R-Script (from Console window) is given as follows:

```
> b<- c(1,2,3,4,5,6,7) + c(1,3)
Warning message:
In c(1, 2, 3, 4, 5, 6, 7) + c(1, 3) :
  longer object length is not a multiple of shorter object length
> b
[1] 2 5 4 7 6 9 8
```

So vector b is obtained as: b <- (1+1, 2+3, 3+1, 4+3, 5+1, 6+3, 7+1)

3. Give an example of recycling of elements.

Solution:

The output of the R-Script (from Console window) is given as follows:

```
> a <- c(10,2,23,4)+c(2,10)
> print(a)
[1] 12 12 25 14
> b<- c(1,2,3,4,5,6,7) + c(1,3)
Warning message:
In c(1, 2, 3, 4, 5, 6, 7) + c(1, 3) :
  longer object length is not a multiple of shorter object length
> print(b)
[1] 2 5 4 7 6 9 8
> x <- c(1,2,3,4,5,6)+c(2,10)
> print(x)
[1]  3 12  5 14  7 16
> y<- c(1,2,3,4,5,6,7) + c(10,30)
Warning message:
In c(1, 2, 3, 4, 5, 6, 7) + c(10, 30) :
  longer object length is not a multiple of shorter object length
> print(y)
[1] 11 32 13 34 15 36 17
```

Task 2:

1. What should be the output of the following Script?

```
v <- c( 2,5.5,6)

t <- c(8, 3, 4)

print(v%/%t)
```

The output of the R-Script (from Console window) is given as follows:

```
> v <- c( 2,5.5,6)
> t <- c(8, 3, 4)
> print(v%/%t)
[1] 0 1 1
```

2. You have 25 excel files with names as xx_1.xlsx, xx_2.xlsx,.....xx_25.xlsx in a dir.

Write a program to extract the contents of each excel sheet and make it one df.

Solution:

setwd("c:/R/mergeme") 0r specific file path name files=list.files(pattern=".xlsx") for(i in 1:length(files)) {filename=files[i] data=read.xlsx(file = filename,header = T) assign(x = filename,value = data)} #Suppose the columns are the same for each file, #you can bind them together in one dataframe with bind_rows from dplyr: library(dplyr) #one more option is as follows df<-lapply(files, read.xlsx) %>% bind_rows()

Task 3:

1. Create an m x n matrix with replicate(m, rnorm(n)) with m=10 column vectors of n=10 elements each, constructed with rnorm(n), which creates random normal numbers.

Then we transform it into a dataframe (thus 10 observations of 10 variables) and perform an algebraic operation on each element using a nested for loop: at each iteration, every element referred by the two indexes is incremented by a sinusoidal function, compare the vectorized and non-vectorized form of creating the solution and report the system time differences.

Solution:

```
#Vectorized form
set.seed(42)


#create matrix
mat_1<- replicate(10,rnorm(10))


#transform into data frame
df_1=data.frame(mat_1)
df_1<- df_1 +10*sin(0.75*pi)


#non-vectorized form
set.seed(42)
#create matrix
mat_1<- replicate(10,rnorm(10))
#transform into data frame
df_1=data.frame(mat_1)


for(iin1:10){
for(j in1:10){
   df_1[i,j]<- df_1[i,j] +10*sin(0.75*pi)
   print(df_1)
   }
}


#time difference
system.time(
     df_1[i,j]<- df_1[i,j] +10*sin(0.75*pi)
)
system.time(
for(iin1:10){
for(j in1:10){
   df_1[i,j]<- df_1[i,j] +10*sin(0.75*pi)
  }
 }
 )
```

Explanation:

Here, Vectorized form and non- Vectorized form is created and converted into data frames respectively. Hence, the time difference is calculated using system.time()

Task 4:

1. Define matrix mymat by replicating the sequence 1:5 for 4 times and transforming into a matrix, sum over rows and columns.

Solution:

The R-script for the given problem is as follows:

```
rep(1:5, 4)              # replicating the sequence 1 to 5

mymat <- matrix(rep(1:5 ,4), nrow = 4 , ncol = 5, byrow = TRUE )

mymat

# sum over rows and columns.

apply(mymat, 1, sum)                    # sum of rows

apply(mymat, 2, sum)                    # sum of columns
```

Explanation:

- Matrix mymat is created by replicating the sequence of 1 to 5 (1,2,3,4,5) for 4 times by using rep(1:5 ,4).
- The matrix mymat is of order 4X5 (4 rows and 5 columns)
- The sum over rows and columns is found by apply() function using the r-commands as follows:

1. apply(mymat, 1, sum) # sum of rows
2. apply(mymat, 2, sum) # sum of columns

Here,1 is used for rows and 2 is used for columns.

The output of the R-Script (from Console window) is given as follows:

```
> rep(1:5, 4)
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> mymat <- matrix(rep(1:5 ,4), nrow = 4 , ncol = 5, byrow = TRUE )
> mymat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    2    3    4    5
[3,]    1    2    3    4    5
```

```
[4,]    1    2    3    4    5
> # sum over rows and columns.
> apply(mymat, 1, sum)    # sum of rows
[1] 15 15 15 15
> apply(mymat, 2, sum)    # sum of columns
[1]  4  8 12 16 20
```

Task 5:

1. States = rownames(US Arrests)

Get states names with ‘w’.

Get states names with ‘W’.

Solution:

The R-script for the given problem is as follows:

```
# Get states names with 'w'.

States[grep("w", States)]

#Get states names with 'W'.

States[grep("W", States)]
```

Explanation:

grep( ) function searches for matches to argument pattern within each element of a character vector.

To get the states names with ‘w’ , grep("w", States) is used.

To get states names with ‘W’ , grep("W", States) is used.

The output of the R-Script (from Console window) is given as follows:

```
> USArrests
            Murder Assault UrbanPop Rape
Alabama       13.2     236       58 21.2
Alaska        10.0     263       48 44.5
Arizona        8.1     294       80 31.0
Arkansas       8.8     190       50 19.5
California     9.0     276       91 40.6
Colorado       7.9     204       78 38.7
Connecticut    3.3     110       77 11.1
Delaware       5.9     238       72 15.8
Florida       15.4     335       80 31.9
Georgia       17.4     211       60 25.8
Hawaii         5.3      46       83 20.2
Idaho          2.6     120       54 14.2
```

```
Illinois          10.4     249       83 24.0
Indiana            7.2     113       65 21.0
Iowa               2.2      56       57 11.3
Kansas             6.0     115       66 18.0
Kentucky           9.7     109       52 16.3
Louisiana         15.4     249       66 22.2
Maine              2.1      83       51  7.8
Maryland          11.3     300       67 27.8
Massachusetts      4.4     149       85 16.3
Michigan          12.1     255       74 35.1
Minnesota          2.7      72       66 14.9
Mississippi       16.1     259       44 17.1
Missouri           9.0     178       70 28.2
Montana            6.0     109       53 16.4
Nebraska           4.3     102       62 16.5
Nevada            12.2     252       81 46.0
New Hampshire      2.1      57       56  9.5
New Jersey         7.4     159       89 18.8
New Mexico        11.4     285       70 32.1
New York          11.1     254       86 26.1
North Carolina    13.0     337       45 16.1
North Dakota       0.8      45       44  7.3
Ohio               7.3     120       75 21.4
Oklahoma           6.6     151       68 20.0
Oregon             4.9     159       67 29.3
Pennsylvania       6.3     106       72 14.9
Rhode Island       3.4     174       87  8.3
South Carolina    14.4     279       48 22.5
South Dakota       3.8      86       45 12.8
Tennessee         13.2     188       59 26.9
Texas             12.7     201       80 25.5
Utah               3.2     120       80 22.9
Vermont            2.2      48       32 11.2
Virginia           8.5     156       63 20.7
Washington         4.0     145       73 26.2
West Virginia      5.7      81       39  9.3
Wisconsin          2.6      53       66 10.8
Wyoming            6.8     161       60 15.6
> States
 [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"       "California"
 [6] "Colorado"       "Connecticut"    "Delaware"       "Florida"        "Georgia"
[11] "Hawaii"         "Idaho"          "Illinois"       "Indiana"        "Iowa"
[16] "Kansas"         "Kentucky"       "Louisiana"      "Maine"          "Maryland"
[21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"    "Missouri"
[26] "Montana"        "Nebraska"       "Nevada"         "New Hampshire"  "New Jersey"
[31] "New Mexico"     "New York"       "North Carolina" "North Dakota"   "Ohio"
[36] "Oklahoma"       "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
[41] "South Dakota"   "Tennessee"      "Texas"          "Utah"           "Vermont"
[46] "Virginia"       "Washington"     "West Virginia"  "Wisconsin"      "Wyoming"
> # Get states names with 'w'.
>
> States[grep("w", States)]
[1] "Delaware"      "Hawaii"        "Iowa"          "New Hampshire" "New Jersey"    "New Mexico"
[7] "New York"
> #Get states names with 'W'.
>
```

```
> States[grep("W", States)]
[1] "Washington"    "West Virginia" "Wisconsin"     "Wyoming"
```

2. Prepare a Histogram of the number of characters in each US state.

Solution:

The R-script for the given problem is as follows:

```
df <- nchar(States)

df

hist(df)
```

Explanation:

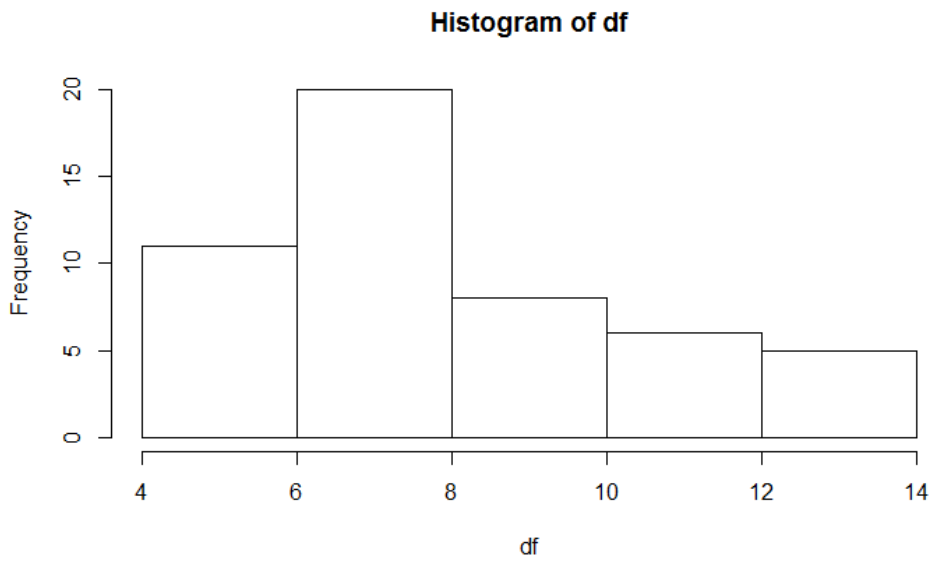
nchar( ) takes a character vector as an argument and returns a vector whose elements contain the sizes of the corresponding elements

hist ( ) computes a histogram of the given data values

The output of the R-Script (from Console window) is given as follows:

```
> df <- nchar(States)
> df
 [1]  7  6  7  8 10  8 11  8  7  7  6  5  8  7  4  6  8  9  5  8 13  8  9 11  8  7  8  6 13 10 10  8
[33] 14 12  4  8  6 12 12 14 12  9  5  4  7  8 10 13  9  7
> hist(df)
```

From plot window:



Task 6:

1. Test whether two vectors are exactly equal (element by element).

vec1 = c(rownames(mtcars[1:15,]))

vec2 = c(rownames(mtcars[11:25,]))

Solution:

The R-script for the given problem is as follows:

```
vec1 = c(rownames(mtcars[1:15,]))

vec2 = c(rownames(mtcars[11:25,]))

 isTRUE(all.equal(vec1,vec2))          # returns true/false

identical(vec1,vec2)                   # returns true/false

all.equal(vec1,vec2)                   # returns number of differences
```

Explanation:

- isTRUE(all.equal(vec1,vec2)) returns TRUE if vec1 is equal to vec2;else it returns FALSE.
- identical(vec1,vec2) returns TRUE if vec1 is identical/same to vec2;else it returns FALSE.
- all.equal(vec1,vec2) returns number of differences between vec1 and vec2.

The output of the R-Script (from Console window) is given as follows:

```
> vec1 = c(rownames(mtcars[1:15,]))
> vec2 = c(rownames(mtcars[11:25,]))
> isTRUE(all.equal(vec1,vec2))
[1] FALSE
> identical(vec1,vec2)
[1] FALSE
> all.equal(vec1,vec2)
[1] "15 string mismatches"
```

2. Sort the character vector in ascending order and descending order.

vec1 = c(rownames(mtcars[1:15,]))

vec2 = c(rownames(mtcars[11:25,]))

Solution:

The R-script for the given problem is as follows:

```
vec1 = c(rownames(mtcars[1:15,]))

vec2 = c(rownames(mtcars[11:25,]))

sort(vec1)                          # vec1 in ascending order

sort(vec1, decreasing = TRUE)       # vec1 in descending order

sort(vec2)                          # vec2 in ascending order

sort(vec2, decreasing = TRUE)       # vec2 in descending order
```

Explanation:

sort(vec1) function arranges the character vector vec1 in ascending order. For descending order "decreasing" parameter is set as "TRUE"

sort(vec2) function arranges the character vector vec2 in ascending order. For descending order "decreasing" parameter is set as "TRUE"

The output of the R-Script (from Console window) is given as follows:

```
> vec1 = c(rownames(mtcars[1:15,]))
> vec2 = c(rownames(mtcars[11:25,]))
> sort(vec1)
 [1] "Cadillac Fleetwood" "Datsun 710"         "Duster 360"         "Hornet 4 Drive"
 [5] "Hornet Sportabout"  "Mazda RX4"          "Mazda RX4 Wag"      "Merc 230"
 [9] "Merc 240D"          "Merc 280"           "Merc 280C"          "Merc 450SE"
[13] "Merc 450SL"         "Merc 450SLC"        "Valiant"
> sort(vec1,decreasing = TRUE)
 [1] "Valiant"            "Merc 450SLC"        "Merc 450SL"         "Merc 450SE"
 [5] "Merc 280C"          "Merc 280"           "Merc 240D"          "Merc 230"
 [9] "Mazda RX4 Wag"      "Mazda RX4"          "Hornet Sportabout"  "Hornet 4 Drive"
[13] "Duster 360"         "Datsun 710"         "Cadillac Fleetwood"
> sort(vec2)
 [1] "AMC Javelin"         "Cadillac Fleetwood"  "Camaro Z28"          "Chrysler Imperial"
 [5] "Dodge Challenger"    "Fiat 128"            "Honda Civic"         "Lincoln Continental"
 [9] "Merc 280C"           "Merc 450SE"          "Merc 450SL"          "Merc 450SLC"
[13] "Pontiac Firebird"    "Toyota Corolla"      "Toyota Corona"
> sort(vec2,decreasing = TRUE)
 [1] "Toyota Corona"       "Toyota Corolla"      "Pontiac Firebird"    "Merc 450SLC"
 [5] "Merc 450SL"          "Merc 450SE"          "Merc 280C"           "Lincoln Continental"
 [9] "Honda Civic"         "Fiat 128"            "Dodge Challenger"    "Chrysler Imperial"
[13] "Camaro Z28"          "Cadillac Fleetwood"  "AMC Javelin"
```

3. What is the major difference between str() and paste() show an example?

Explanation:

str() gives the class of variable, number of values and the elements whereas paste() printsor displays the actual elements .

For example:

str(mtcars$mpg) gives the class of mtcars$mpg as num, number of values as 32(1:32) and the elements as 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...

whereas **paste(mtcars$mpg)** prints the actual elements present in mtcars$mpg.

The output of the R-Script (from Console window) is given as follows:

```
> str(mtcars$mpg)
 num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
> paste(mtcars$mpg)
 [1] "21"   "21"   "22.8" "21.4" "18.7" "18.1" "14.3" "24.4" "22.8" "19.2" "17.8" "16.4" "17.3"
[14] "15.2" "10.4" "10.4" "14.7" "32.4" "30.4" "33.9" "21.5" "15.5" "15.2" "13.3" "19.2" "27.3"
[27] "26"   "30.4" "15.8" "19.7" "15"   "21.4"
```

4. Introduce a separator when concatenating the strings.

The R-script for the given problem is as follows:

```
paste(rownames(mtcars[1,]), rownames(mtcars[2,]), sep = " ")

paste(rownames(mtcars[1,]), rownames(mtcars[4,]), sep = ",")

paste(rownames(mtcars[2,]), rownames(mtcars[1,]), sep = "--")

paste(rownames(mtcars[3,]), rownames(mtcars[10,]), sep = "$")

paste("hello","world",sep=" @ ")

paste("Assignment","5","3",sep="_")
```

Explanation:

paste(rownames(mtcars[1,]), rownames(mtcars[2,]), sep = " ") introduces a separator ,a single blank " " between the strings rownames(mtcars[1,]) and rownames(mtcars[2,].

paste(rownames(mtcars[1,]), rownames(mtcars[4,]), sep = ",") introduces a separator comma ", " between the strings rownames(mtcars[1,]) and rownames(mtcars[4,].

paste(rownames(mtcars[2,]), rownames(mtcars[1,]), sep = "--") introduces a separator "-- " between the strings rownames(mtcars[2,]) and rownames(mtcars[1,].

paste(rownames(mtcars[3,]), rownames(mtcars[10,]), sep = "$") introduces a separator dollar "$ " between the strings rownames(mtcars[3,]) and rownames(mtcars[10,].

paste("hello","world",sep="@") introduces a separator "@" between the strings “hello” and “world”

paste("Assignment","5","3",sep="_") introduces a separator underscore "_ " between the strings "Assignment","5" and "3".

The output of the R-Script (from Console window) is given as follows:

```
> paste(rownames(mtcars[1,]), rownames(mtcars[2,]), sep = " ")
[1] "Mazda RX4 Mazda RX4 Wag"
> paste(rownames(mtcars[1,]), rownames(mtcars[4,]), sep = ",")
[1] "Mazda RX4,Hornet 4 Drive"
> paste(rownames(mtcars[2,]), rownames(mtcars[1,]), sep = "--")
[1] "Mazda RX4 Wag--Mazda RX4"
> paste(rownames(mtcars[3,]), rownames(mtcars[10,]), sep = "$")
[1] "Datsun 710$Merc 280"
> paste("hello","world",sep=" @ ")
[1] "hello @ world"
> paste("Assignment","5","3",sep="_")
[1] "Assignment_5_3"
```