Contents

Quiz for Week 2

R Programming
Week 2 Videos

www.Stats-Lab.com

Quiz Question 1

Suppose we define the following function in R

```
cube <- function(x, n) {
        x^3
}
```

What is the result of running cube(3) in R after defining this function?

Options

1. The number 27 is returned
2. An error is returned because 'n' is not specified in the call to 'cube'
3. A warning is given with no value returned.
4. The users is prompted to specify the value of 'n'.

Remarks

- As we can see by running the code - we get the answer we expect.
- While n is specified as a possible argument for the function, it is clear that it is not used in the body of the function, and that an answer can be computed without it.

Counter-Example

```
pow <- function(x, n=3) {
        x^n
}
```

Quiz Question 2

Suppose I define the following function in R

```
pow <- function(x = 4, n = 3) {
        x^n
}
```

What is the result of running `pow()` in `R` after defining this function?

1. A warning is given and the function returns 64.
2. An error is given the function does not finish execution.
3. The number 64 is returned.
4. The number 81 is returned.

As we can see by running the code - we get the answer we expect, without any error warnings.

If no arguments are specified by the user, the function will use the **default** settings.

This function can be used for other values of `x` and `n`.

Quiz Question 3

The following code will produce a warning in `R`.Why is this?

```
x <- 1:10
if(x > 5) {
        x <- 0
}
```

Options

1. '`x`' is a vector of length 10 and '`if`' can only test a single logical statement.
2. There are no elements in 'x' that are greater than 5 (**obviously false**)
3. The expression uses curly braces.
4. You cannot set '`x`' to be 0 because '`x`' is a vector and 0 is a scalar.
   (**true**)
5. The syntax of this `R` expression is incorrect.
   (**No obvious Syntax errors**)

The `aggregate()` function

Firstly let us construct a data frame called ***df1***, using the code below.

```
set.seed(1)  #generating random numbers

#Three Variables
col1 <- c(rep('happy',9), rep('sad', 9))
col2 <- rep(c(rep('alpha', 3),
   rep('beta', 3), rep('gamma', 3)),2)
score=rnorm(18, 10, 3)

#Combine the 3 variables as a data frame
df1<-data.frame(col1=col1, col2=col2, score=score)
df1
```

The aggregate() function

There are two categorical variables. The first (i.e. `col1`) has two levels, the second (i.e. `col2`) has three.
We can use the `aggregate()` function to apply a specified command for groups.

```
aggregate(variable, by=list(group1,group2,..),function)
```

```
aves1 = aggregate(df1$score,
    by=list(col2=df1$col2), mean)

aves1



aves2 = aggregate(df1$score,
    by=list(col1=df1$col1), mean)

aves2

aves3 = aggregate(df1$score,
   by=list(col1=df1$col1, col2=df1$col2), mean)

aves3
```

We can use the `merge` command to combine the group-wise results with the original data frame.

```
results = merge(df1, aves2)
results
```

```
> results = merge(df1, aves2)
> results
    col1  col2     score        x
1  happy alpha  8.120639 10.54247
2  happy alpha 10.550930 10.54247
3  happy alpha  7.493114 10.54247
4  happy  beta 14.785842 10.54247
5  happy  beta 10.988523 10.54247
..............
```

Quiz Question 4

Take a look at the 'iris' dataset that comes with `R`.

The data can be loaded with the code:

```
library(datasets)
data(iris)
```

Questions

- A description of the dataset can be found by running `?iris`.
- There will be an object called '***iris***' in your workspace.
- In this dataset, what is the mean of 'Sepal.Length' for the species virginica?
- `(Please only enter the numeric result and nothing else.)`

Remarks

- One approach is to use the `aggregate()` command mentioned in the previous section.
- A second approach is to create a subset (let's call it ***iris.vir***) and then use the `summary()` command.

```
    iris.vir=iris[iris$Species=="virginica",]

    #Alternatively
    iris.vir=subset(iris,iris$Species=="virginica")
```

```
summary(iris.vir)
```

The summary of ***iris.vir*** should look like this:

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.900   Min.   :2.200   Min.   :4.500   Min.   :1.40
 1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100   1st Qu.:1.80
 Median :6.500   Median :3.000   Median :5.550   Median :2.00
 Mean   :6.588   Mean   :2.974   Mean   :5.552   Mean   :2.02
 3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875   3rd Qu.:2.30
 Max.   :7.900   Max.   :3.800   Max.   :6.900   Max.   :2.50
       Species
 setosa    : 0
 versicolor: 0
 virginica :50
```

0.1 The `apply()` function

The ***Apply*** family of functions keep you from having to write loops to perform some operation on every row or every column of a matrix or data frame, or on every element in a list.

The `apply()` function

The `apply()` function is a powerful device that operates on arrays and, in particular, matrices.

The `apply()` function returns a vector (or array or list of values) obtained by applying a specified function to either the row or columns of an array or matrix.

To specify use for rows or columns, use the additional argument of

- 1 for rows,
- 2 for columns.

```
m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)
m
apply(m,1,mean)
apply(m,2,mean)
```

```
> # create a matrix of 10 rows x 2 columns
> m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)
>
> # mean of the rows
>
```

```
> apply(m, 1, mean)
[1] 6 7 8 9 10 11 12 13 14 15
>
> # mean of the columns
> apply(m, 2, mean)
[1] 5.5 15.5
```

The `lapply()` and `sapply()` function

The `lapply()` command returns a list of the same length as a list X, each element of which is the result of applying a specified function to the corresponding element of X.

A simpler user-friendly version of `lapply()` is `sapply()` The `sapply()` command is a variant of `lapply()` returning a simple vector instead of a list - again of the same length as a list X, each element of which is the result of applying a specified function to the corresponding element of X.

```
> x <- list(a=1:10, b=exp(-3:3), logic=c(T,F,F,T))
>
> # compute the list mean for each list element
>
> lapply(x,mean)
$a
[1] 5.5
$b
[1] 4.535125
$logic
[1] 0.5
>
> sapply(x,mean)
```

```
a b logic
5.500000 4.535125 0.500000
>
```

Quiz Question 5

Continuing with the '***iris***' dataset from Question 4, what `R` code returns a vector of the means of the variables 'Sepal.Length', 'Sepal.Width', 'Petal.Length', and 'Petal.Width'?

```
colMeans(iris)
apply(iris, 2, mean)
apply(iris[, 1:4], 1, mean)
apply(iris[, 1:4], 2, mean)
apply(iris, 1, mean)
rowMeans(iris[, 1:4])
```

Try out all the code, to see what happens. However, based on your knowledge of the apply family of functions, you should spot that this option would be suitable.

```
> apply(iris[, 1:4], 2, mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

`R` code that would have worked if it had been an option

```
> colMeans(iris[,1:4])
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

Quiz Question 6

Load the '***mtcars***' dataset in `R` with the following code

```
library(datasets)
data(mtcars)
```

- There will be an object names '***mtcars***' in your workspace. You can find some information about the dataset by running `?mtcars`.
- How can one calculate the average miles per gallon (mpg) by number of cylinders in the car (cyl)?

Options

```
tapply(mtcars$cyl, mtcars$mpg, mean)
split(mtcars, mtcars$cyl)
mean(mtcars$mpg, mtcars$cyl)
tapply(mtcars$mpg, mtcars$cyl, mean)
```

- Cars can have either 4, 6 or 8 cylinders.

  ```
  > table(mtcars$cyl)

   4  6  8
  11  7 14
  ```

- using the `tapply()` function

  ```
  > tapply(mtcars$mpg, mtcars$cyl, mean)
         4        6        8
  26.66364 19.74286 15.10000
  ```

Quiz Question 7

Continuing with the ***mtcars*** dataset from Question 6, what is the absolute difference between the average horsepower of 4-cylinder cars and the average horsepower of 8-cylinder cars?

Remarks

This is another question where the `aggregate()` command comes in handy. (We will use `attach()` and `detach()` to avoid unnecessary typing)

```
> attach(mtcars)
> CylMeans <- aggregate(hp,by=list(Cyls=cyl),mean)
> CylMeans
  Cyls         x
1    4  82.63636
2    6 122.28571
3    8 209.21429
> detach(mtcars)
>
> CylMeans$x
[1]  82.63636 122.28571 209.21429
>
> CylMeans$x[3]-CylMeans$x[1]
[1] 126.5779
```

0.2 Quiz Question 8

What is the difference between the '`sapply()`' function and the '`lapply()`' function?

Options

1. There is no difference; 'sapply' and 'lapply' are two names for the same function
2. '`sapply()`' always returns a 2-dimensional matrix while 'lapply' returns a list.
3. '`lapply()`' always returns a list while '`sapply()`' attempts to simplify the result. (**Correct**)
4. '`lapply()`' always returns an atomic vector and 'sapply' always returns a list.

The question can be easily solved by reading the help files for both commands.

```
help(sapply)
```

0.3 Quiz Question 9

Consider the following function

```
f <- function(x) {
        g <- function(y) {
                y + z
        }
        z <- 4
        x + g(x)
}
```

If I then run in `R`

```
z <- 10
f(3)
```

What value is returned by 'f'?

Quiz Question 10

If you run `debug(ls)` what happens when you next call the '`ls`' function?

1. Execution of '`ls`' will suspend at the beginning of the function and you will be in the browser.
2. The '`ls`' function will execute as usual. (**FALSE**)
3. The '`ls`' function will return an error. (**FALSE**)
4. Execution of the '`ls`' function will suspend at the 4th line of the function and you will be in the browser.
5. You will be prompted to specify at which line of the function you would like to suspend execution and enter the browser.