CM 2062 - Statistical Computing with R
Lab Sheet 8

for-Loop Over Data Frame Columns

Example 1
Recall the "marks" data frame in R. Write a R programme to get the total marks of each student.

```
for (i in 1:nrow(marks))
{
  total<- sub1[i] + sub2[i] + sub3[i]
  print(total)
}
```

```
[1] 190
[1] 188
[1] 211
```

Example 2
Let's consider the "iris" data set in R.

```
data(iris) # Loading iris flower data set
head(iris)
```

Our example data frame contains five columns consisting of information on iris flowers. Let's also replicate our data in a new data frame object called iris_new.

```
iris_new <- iris
```

We can loop over the columns of our data frame using the **ncol** function within the head of the for-statement. Within the for-loop, we are also using a logical if-condition.

```
for(i in 1:ncol(iris_new)) {

  if(grepl("Width", colnames(iris_new)[i])) {      # grepl use for matching

    iris_new[ , i] <- iris_new[ , i] + 1000
  }
}
```

Let's have a look at the updated data frame.

```
> head(iris_new)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1      1003.5          1.4       1000.2  setosa
2          4.9      1003.0          1.4       1000.2  setosa
3          4.7      1003.2          1.3       1000.2  setosa
4          4.6      1003.1          1.5       1000.2  setosa
5          5.0      1003.6          1.4       1000.2  setosa
6          5.4      1003.9          1.7       1000.4  setosa
```

Create Variable Names Using for-Loop

```
iris_new1 <- iris
```

Now, we can apply the **colnames** and **paste0** functions to create new column names for each of our data frame columns.

```
iris_new1 <- iris

for(i in 1:ncol(iris_new1)) {

  colnames(iris_new1)[i] <- paste0("new_", i)
}
```

```
> head(iris_new1)
  new_1 new_2 new_3 new_4  new_5
1   5.1   3.5   1.4   0.2 setosa
2   4.9   3.0   1.4   0.2 setosa
3   4.7   3.2   1.3   0.2 setosa
4   4.6   3.1   1.5   0.2 setosa
5   5.0   3.6   1.4   0.2 setosa
6   5.4   3.9   1.7   0.4 setosa
```

for-Loop Through List Object

Let's create a list first.

```
my_list <- list(1:5,letters[3:1],"XXX")
```

```
> my_list
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "c" "b" "a"

[[3]]
[1] "XXX"
```

Our list consists of three different list elements. Now, we can use the length function to loop over our list.

```
for(i in 1:length(my_list)) {
  my_list[[i]] <- rep(my_list[[i]], 3)
}
my_list

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

[[2]]
[1] "c" "b" "a" "c" "b" "a" "c" "b" "a"

[[3]]
[1] "XXX" "XXX" "XXX"
```

Creating Multiple Plots within for-Loop

for-loops can be very handy when you want to draw multiple plots efficiently within a few lines of code. Let's assume that we want to draw a plot of each numeric column of the iris data frame. Then, we can use the following R code.

Example 1

```
for(i in 1:(ncol(iris) - 1)) {
  plot(1:nrow(iris), iris[ , i])
  Sys.sleep(1)       # Pause code execution
}
```

You can see the four plots for four variables within 1 second intervals.

Example 2

Plotting ggplot2 plots within for Loop. If we want to draw a plot within a for-loop using ggplot2, we need to wrap the **print** function around the R code creating the plot.

```
data <- data.frame(x = 1:100,
                   y1 = rnorm(100),
                   y2 = rnorm(100),
                   y3 = rnorm(100))
library("ggplot2")
for(i in 2:ncol(data)) { # Printing ggplot within for-loop
  print(ggplot(data, aes(x = x, y = data[ , i])) +
          geom_point())
  Sys.sleep(2)
}
```

if else statement in R

The syntax of an if...else statement is,

```
if (test_expression) {
  # body of if statement
} else {
  # body of else statement
}
```

Example 1

```
dice <- 1:6

for(x in dice) {
  if (x == 6) {
    print(paste("The dice number is", x, "Yahtzee!"))
  } else {
    print(paste("The dice number is", x, "Not Yahtzee"))
  }
}
```

```
[1] "The dice number is 1 Not Yahtzee"
[1] "The dice number is 2 Not Yahtzee"
[1] "The dice number is 3 Not Yahtzee"
[1] "The dice number is 4 Not Yahtzee"
[1] "The dice number is 5 Not Yahtzee"
[1] "The dice number is 6 Yahtzee!"
```

Example 2

```
team_A <- 1 # Number of goals scored by Team A
team_B <- 3 # Number of goals scored by Team B
if (team_A > team_B){
  print ("Team A will make the playoffs")
} else {
  print ("Team B will make the playoffs")
}
```

Exercise 1

Let x=12. Write a R function to check whether x is a positive or negative number and display the result.

Alternative to the if else statement in R

The **ifelse()** function is a shorthand function to the traditional if...else statement.

Syntax of ifelse() function:

```
ifelse(test_expression, x, y)
```

Here, test_expression must be a logical vector (or an object that can be coerced to logical). The return value is a vector with the same length as test_expression. This returned vector has element from x if the corresponding value of test_expression is TRUE or from y if the corresponding value of test_expression is FALSE. This is to say, the i-th element of result will be x[i] if test_expression[i] is TRUE else it will take the value of y[i]. The vectors x and y are recycled whenever necessary.

Example

```
a <- c(5,7,2,9)
ifelse(a %% 2 == 0,"even","odd")
```

```
[1] "odd"  "odd"  "even" "odd"
```

if...else if...else Statement in R

if...else if...else statement allows you execute a block of code among more than 2 alternatives.

The syntax of if...else if ... else statement is:

```
if ( test_expression1) {
statement1
} else if ( test_expression2) {
statement2
} else {
statement3
}
```

Only one statement will get executed depending upon the test_expressions.

Example

```
x <- 0
if (x < 0) {
print("Negative number")
} else if (x > 0) {
print("Positive number")
} else
print("Zero")
```

```
[1] "Zero"
```

for loop with if else statement

Example 1

```
matches <- list(c(2,1),c(5,2),c(6,3))

for (match in matches){
  if (match[1] > match[2]){
    print("Win")
  } else {
    print ("Lose")
  }
}
```

```
[1] "Win"
[1] "Win"
[1] "Win"
```

Example 2

```
matches <- list(c(2,1),c(5,2),c(6,3))

for (match in matches){
    if (match[1] > match[2]){
        print("Win")
        break
    } else {
        print("Lose")
        }
}
```

```
[1] "Win"
```

Example 3

```
x <- 1:5
for(i in 1:length(x)) {        # Using ifelse function in for loop
  cat(ifelse( x[i] == 1,
              yes = "If condition was TRUE",
              no = "If condition was FALSE"),
      "\n")
}
```

```
If condition was TRUE
If condition was FALSE
If condition was FALSE
If condition was FALSE
If condition was FALSE
```

Alternatives to the for Loop

Sometimes, using loops, the R programme can be very slow when applied to large data sets or in complex settings such as nested for-loops. For that reason, it might make sense for you to avoid for-loops and to use functions such as the **family of apply functions** instead. This might speed up the R syntax and can save a lot of computational power. The functions in family of apply functions are **apply(), lapply(), sapply(), vapply(), tapply() and mapply()**.

apply() function in R

The apply function takes data frames as input and can be applied by the rows or by the columns of a data frame.

Example
Recall the "marks" data frame in R. Use apply() function to get the total marks of each student.

```
apply(marks, 1, sum)
```

```
 A1  B1  C1
190 188 211
```

As you can see based on the above R code, we specified three arguments within the apply function,
The name of our data frame (i.e. marks).
Whether we want to use the apply function by rows or by columns. The value 1 indicates that we are using apply by row.
The function we want to apply to each row (i.e. the sum function).

lapply() Function in R

The l in front of apply stands for "list".

Within the lapply function, we simply need to specify the name of our list (i.e. my_list) and the function we want to apply to each list element.

Example
Let's take the length of each list element in my_list.

```
my_list <- list(1:5,letters[3:1],"XXX")
lapply(my_list, length)
```

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

[[2]]
[1] 3
```

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

sapply() Function in R

The sapply function (s stands for simple) therefore provides a simpler output than lapply.

```
sapply(my_list, length)
```

```
[1] 5 3 1
```

vapply() Function in R

The vapply function is very similar compared to the sapply function, but when using vapply you need to specify the output type explicitly. In this example, we'll return an integer.

```
vapply(my_list, length, integer(1))
```

```
[1] 5 3 1
```

tapply() Function in R

The tapply function is another command of the apply family, which is used for vector inputs.

Example

Let's consider the iris_new data frame and we need to find the mean of Sepal.Length for each Species.

```
tapply(iris_new$Sepal.Length, iris_new$Species, mean)
```

```
setosa versicolor  virginica
5.006      5.936      6.588
```

mapply() Function in R

Another function that is used for vectors is mapply.

Example 1

Create a Matrix.

```
mapply(rep, 1:3, times=5)
```

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

Example 2

```
mapply(rep, times = 1:5, letters[1:5])
```

```
[[1]]
[1] "a"

[[2]]
[1] "b" "b"

[[3]]
[1] "c" "c" "c"

[[4]]
[1] "d" "d" "d" "d"

[[5]]
[1] "e" "e" "e" "e" "e"
```

Home Works

1. Find more examples of replacing for loop using family of apply functions.

2. Try to do the examples that we discussed under the for loop using family of apply functions.

3. Find more alternatives to the for Loop.