Welling Intermediate R Assignment

> Premature optimization is the root of all evil – Donald Knuth

The humble for loop is often considered distasteful by seasoned programmers because it is inefficient; however, the for loop is one of the most useful and generalizable programming structures in R. If you can learn how to construct and understand for loops then you can code almost any iterative task. Once your loop works you can always work to optimize your code and increase its efficiency.

Before attempting these exercises you should review the lesson R intermediate (../lessons/R_intermediate) in which loops were covered.

Examine the following for loop, and then complete the exercises

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 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
```

```
sp_ids = unique(iris$Species)

output = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(output) = sp_ids
colnames(output) = names(iris[ , -ncol(iris)])

for(i in seq_along(sp_ids)) {
    iris_sp = subset(iris, subset=Species == sp_ids[i], select=-Species)
    for(j in 1:(ncol(iris_sp))) {
        x = 0
        y = 0
        if (nrow(iris_sp) > 0) {
            for(k in 1:nrow(iris_sp)) {
                x = x + iris_sp[k, j]
                y = y + 1
            }
            output[i, j] = x / y
        }
    }
}
output
```

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

Excercises

Iris loops

1. Describe the values stored in the object `output`. In other words what did the loops create?
   The loops created a matrix of means for the leaf parameters for each species.
2. Describe using pseudo-code how `output` was calculated, for example, (explain it in english in the same format as the code)

```
sp_ids = unique(iris$Species)
# Make a new vector with no repitition of species names

output = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
#Make a new object ("output") that is a matrix with no data (right now), with the same n
umber of rows as the sp_id vector, and the one fewer column than the iris data frame

rownames(output) = sp_ids
#Make the row names of the matrix the names in the vector sp_id

colnames(output) = names(iris[ , -ncol(iris)])
#Make the column names of the matrix the same names from the iris data frame except the
 last column in iris

for(i in seq_along(sp_ids)) {
  #for every species listed in the vector sp_ids
    iris_sp = subset(iris, subset=Species == sp_ids[i], select=-Species)
    #make iris_sp using the iris data, grouped by species, take out the species column
    for(j in 1:(ncol(iris_sp))) {
      #for every item from the first through the number of columns in sp_ids
        x = 0
        y = 0
        #beginning with x=0 and y=0??
        if (nrow(iris_sp) > 0) {
          #if the number of rows in iris_sp is greater than zero
            for(k in 1:nrow(iris_sp)) {
              #for every item from the first to the number of rows in iris_sp
                x = x + iris_sp[k, j]
                #make x= x + all of the rows, and do it for every column
                y = y + 1
                #make y equal to y + 1
            }
            output[i, j] = x / y
            #fill the matrix output with the value to x divided by y
        }
    }
}
output
```

3. The variables in the loop were named so as to be vague. How can the objects `output`, `x`, and `y` could be renamed such that it is clearer what is occurring in the loop. ‘Output’ could be renamed as ‘mean_parameter’ or ‘iris_means’, ‘x’ could be named ‘parameter_sums’, and ‘y’ could be ‘length’.

4. It is possible to accomplish the same task using fewer lines of code? Please suggest one other way to calculate `output` that decreases the number of loops by 1. Instead of calculating the mean by hand, you can use the ‘mean()’ function in R, and this modifications allows you to take out the ‘k’ loop.

```
data(iris)

sp_ids = unique(iris$Species)

output = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(output) = sp_ids
colnames(output) = names(iris[ , -ncol(iris)])

for(i in seq_along(sp_ids)) {
    iris_sp = subset(iris, subset=Species == sp_ids[i], select=-Species)
    for(j in 1:(ncol(iris_sp))) {
        if (nrow(iris_sp) > 0) {
           x = mean(iris_sp[,j])
            }
            output[i, j] = x
        }
    }
output
```

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

Sum of a sequence

5. You have a vector `x` with the numbers 1:10. Write a for loop that will produce a vector `y` that contains the sum of `x` up to that index of `x` . So for example the elements of `x` are 1, 2, 3, and so on and the elements of `y` would be 1, 3, 6, and so on.

```
x<- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

y <- vector("numeric", 10)
for (i in x) {
  y[i] <- sum(1:i)
}
y
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

6. Modify your for loop so that if the sum is greater than 10 the value of `y` is set to NA

```
x<- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

y <- vector("numeric", 10)
for (i in x) {
  y[i] <- sum(1:i)
  if (y[i] <= 10) {
    print(y[i])
  }
  else {
    print("NA")
  }
}
```

```
## [1] 1
## [1] 3
## [1] 6
## [1] 10
## [1] "NA"
## [1] "NA"
## [1] "NA"
## [1] "NA"
## [1] "NA"
## [1] "NA"
```

7. Place your for loop into a function that accepts as its argument any vector of arbitrary length and it will return `y`.

```
x<- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

sums_x <- function (x) {
  y <- NULL
  for (i in x) {
    y[i] <- sum(1:i)
    if (y[i] <= 10) {
      print(y[i])
    }
  else {
    print("NA")
  }
  }
}

sums_x(x)
```

```
## [1] 1
## [1] 3
## [1] 6
## [1] 10
## [1] "NA"
## [1] "NA"
## [1] "NA"
## [1] "NA"
## [1] "NA"
## [1] "NA"
```