R Intermediate HW_Will

Morgan Will

1/18/2022

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 in which loops were covered.

Examine the following for loop, and then complete the exercises

```
data(iris)
#head(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))) {
        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
```

```
#sp_ids = vector listing unique species names in dataframe "iris"
#output = defines an empty matrix with 3 rows (number of unique species in ir
is,
  #from sp_ids) and 4 columns (number of columns in iris minus 1, since speci
es
  #is what we are filtering for)
```

```
#rownames(output) = sets row names of matrix "output" equal to vector values
  #of "sp_ids"
#colnames(output) = sets column names of matrix "output" equal to the column
  #names or dataframe "iris", minus the last column (-ncol(iris))
#First for loop: iris_sp = creating a subset or the iris dataframe.
      #"subset=Species == sp_ids[i]" tells function to keep only rows that ar
e
        #equal to a specific species ([1] = setosa, [2] = versicolor,
        #[3] = virginica)
      #"select=-Species" drops the "Species" column from the dataframe
#Second for loop: sets "x" and "y" equal to 0 for all columns of "iris_sp"
  #(setting up for next step)
#Third if loop: for each column of "irsi_sp", if number of rows > 0....
#Fourth for loop: adds each value from the specified column to x, adds value
  #of 1 to y for each row iterated (setting up to calculate mean)
#output = for each value [i,j] (row,col of matrix), take the mean of the valu
es
  #(x/y) to put back into the output matrix (which already has column and row
  #names from before)
```

Exercises

Iris loops

1. Describe the values stored in the object `output`. In other words what did the loops create?

   The code above first creates an empty matrix "output". The names of the matrix rows and columns are then specified based on the "iris" dataframe (rows are unique species names, columns are same as those found in "iris", minus the last column). The loops are then used to calculate the average of the values from "iris" for each empty space in the matrix (ex: row 1 col1 is the average of all values from the column "Sepal.Length" in dataframe "iris" that have "setosa" in the species column of "iris").

2. Describe using pseudo-code how `output` was calculated, for example,

```
Loop from 1 to length of species identities
   Take a subset of iris data (iris_sp)
   Loop from 1 to number of columns of the iris data
      If greater than 0 rows exist within sp_ids
        Loop from 1 to number of rows within iris_sp
          Sum the column values (x) and count the number of values sumed (y)
        Take the average within the "output" matrix (x/y)
```

3. The variables in the loop were named so as to be vague. How can the objects `output`, `x`, and `y` be renamed such that it is clearer what is occurring in the loop.

   output -> AvgPerSpecies

   x -> MatSum

   y -> MatCount

4. Is it 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.

```
data(iris)
sp_ids = unique(iris$Species)
AvgPerSpecies = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(AvgPerSpecies) = sp_ids
colnames(AvgPerSpecies) = 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))) {
        MatSum = 0
        MatCount = 0
        #Removed if loop
            for(k in 1:nrow(iris_sp)) {
                MatSum = MatSum + iris_sp[k, j]
                MatCount = MatCount + 1
            AvgPerSpecies[i, j] = MatSum / MatCount
        }
    }
}
AvgPerSpecies
```

```
##            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 <- c()
for(i in x) {
  VecSum = 0
  for(j in x) {
    VecSum <- VecSum +j
    y[j] <- VecSum
  }
}
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 <- c()
for(i in x) {
  VecSum <- 0
```

```
  for(j in x) {
    VecSum <- VecSum + j
    y[j] <- VecSum
      if(y[j] > 10) {
        y[j] <- NA
      }
  }
}
y
```

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

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

```
eval_x <- function(x){
  y <- c()
  for(i in x) {
   VecSum <- 0
    for(j in x) {
     VecSum <- VecSum + j
     y[j] <- VecSum
        if(y[j] > 10) {
          y[j] <- NA
        }
    }
  }
  return(y)
}
eval_x(x)
```

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