Intermediate R

Emily Parsons

January 14, 2020

Collaborator: Anna Radchenko

```
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
```

1.Describe the values stored in the object “output”. In other words, what did these loops create?

The values stored in ‘output’ are the means of the columns of data in the ‘iris’ dataset - sepal lenght, sepal width, petal length, and petal width, subdivided by species.

2. Describe using pseudo-code how output was calculated.

```
sp_ids = unique(iris$Species)
#unique tells you all the unique entries of the number. This is making a unique vector of specie
s names and removes repeat values.


output = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
#Creating an empty object called 'output', assigning the species id to the row and and establish
ing sp_ids as the length of the vector. It is also assigning the number of columns as 1 less. In
other words, making an empty matirx in which to put values outputed by the loop. The 0 fills the
empty matrix with zeros, and you ned up with a 3x4 matrix filled with 0.


rownames(output) = sp_ids
colnames(output) = names(iris[ , -ncol(iris)])
#Assigning species ids as the row names and col names as the colnames from the iris data set



for(i in seq_along(sp_ids)) {
  #Creating the for loop. i is some object/index for the operation we want to repeat. seq_along
 is a function that is being assigned in a loop to sp_ids.


    iris_sp = subset(iris, subset=Species == sp_ids[i], select=-Species)
# Telling the loop to pull out all data for the first time it runs thru setosa, to make it work
 with one row at a time. The step to basically parse out the groups of species This is how to su
bset the rows in a particular way, and how to select the rows


    for(j in 1:(ncol(iris_sp))) {
      #initiating a nested for loop from one to the length of the iris species ids
        x = 0
        y = 0
      #establishing x and y as vectors equal to zero
        if (nrow(iris_sp) > 0) {
      #An if statement saying that if the value in the row in the iris_sp vector is greater than
zero.... continue with these values for the rest of the code... Basically excluding all zero val
ues for the math it will do later. A logical step that would be useful for a data set with no da
ta in a row


            for(k in 1:nrow(iris_sp)) {
             #initiating another nested loop that will run the length of the rows in iris_sp. Sa
me thing as before but now run through each row at a time.

                x = x + iris_sp[k, j]
                #this is adding the values of each row and column together for a sum of the valu
es

                y = y + 1
                #this is adding 1 more to the value of y to eventually give out the total number
of observations
            }
            output[i, j] = x / y
            #Dividing the sum of the observations in each row and column by the number of the ob
vervations and assigning them to the empty vextor "output"
        }
    }
}

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
```

```
#Displaying the output of the code
```

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 have been called means, or sp_means to make it more clear what we are doing. x is the sum of the observations in the rowsand columns, so it could be called 'sum'. Y is the total number of observations, so it could be called 'total_obvs'.

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.

You can take out the if (nrow(iris_sp) > 0) statement, because it is simply a logical check looking for rows with no data in them to exclude. This would be useful in a dataset with rows of no data, but here is unneccesary because there is data for all iris species.

```
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
            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
```

OR you could forgo loops and use a function like tapply . . . although this doesn’t really seem to shorten the code by too much.

```
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)])

output[,1] <- tapply(iris$Sepal.Length, iris$Species, mean)
output[,2] <- tapply(iris$Sepal.Width, iris$Species, mean)
output[,4] <- tapply(iris$Petal.Width, iris$Species, mean)
output[,3] <- tapply(iris$Petal.Length, iris$Species, mean)

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
```

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 <- 1:10
y = NULL

 for(i in x) {
    y[i] <- sum(x[1:i])
 }

print(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 <- 1:10
y = NULL

 for(i in x) {
     y[i] <- sum(x[1:i])
     if(y[i] >10)
      y[i]=NA

      else{
      (y[i] <- sum(x[1:i]))
    }

}

print(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.

```
sum_seq2 <- function(x) {



 for(i in x) {
    y[i] <- sum(x[1:i])

    if(y[i] >10)
         y[i]=NA

    else{
        (y[i] <- sum(x[1:i]))
    }

}

print(y)

}
z<-1:40
sum_seq2(z)
```

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