Solutions to Assignment 2: R intermediate

Dan McGlinn

January 15, 2016

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?

These values are averages of the traits of each species.

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

```
#loop through species ids
#    subset iris down to only rows associated with a particular species
#   loop through columns (i.e., species traits)
#       if their are records associated with that column then
#           loop through each observation
#               sum across the observations
#               count the number of observations
#           compute the mean across the observations
```

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.

```
# the simplest change here it to rename the vauge objects
sp_mean = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(sp_mean) = sp_ids
colnames(sp_mean) = 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))) {
        trait_sum = 0
        num_records = 0
        if (nrow(iris_sp) > 0) {
            for(k in 1:nrow(iris_sp)) {
                trait_sum = trait_sum + iris_sp[k, j]
                num_records = num_records + 1
            }
            sp_mean[i, j] = trait_sum / num_records
        }
    }
}
```

The loop above is much easier to read and understand by not using vauge names.

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.

```
# R has a function to compute means so we can drop quite a few lines using that
sp_mean = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(sp_mean) = sp_ids
colnames(sp_mean) = 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))) {
        sp_mean[i, j] = mean(iris_sp[ , j])
    }
}
```

By using a function here such as mean we've made our code more readable and less prone to error

```
# here are two other ways to simplify the code even further that both
# accomplish the exact same task

# approach 1
apply(iris[ , -5], 2, function(trait) tapply(trait, iris$Species, mean))
```

```
##            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
```

```
# approach 2
t(sapply(as.character(sp_ids), function(sp)
       apply(iris[iris$Species == sp, -5], 2, mean)))
```

```
##            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
```

Approach 1 seems like the better choice for readability; however, this very condensed code is more difficult to understand. Most beginners would consider this code to be very tearse. Using the `apply` family of functions can produce large code speed ups in certain situtations. The exception is `sapply` which is essentially just simple for loop.

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 = 1:10
y = NULL
for(i in 1:length(x)) {
    y[i] = sum(x[1:i])
}
y
```

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

```
# altneratively we could use an sapply function
y = sapply(1:length(x), function(i) sum(x[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

```r
y = NULL
for(i in 1:length(x)) {
    y[i] = sum(x[1:i])
    if (y[i] > 10) {
        y[i] = NA
    }
}
y
```

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

```r
# altneratively although much more difficult to understand we could use
y = sapply(1:length(x), function(i) ifelse(sum(x[1:i]) > 10, NA, sum(x[1:i])))

# I definately prefer the loop approach in this context because of readability
# the sapply approach will be no faster
```

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

```r
cum_sum_cutoff = function(x, cutoff=10) {
    # this function computes a cumulative sum along a numeric vector up to a cutoff
    # arguments
    # x: a numeric vector
    # cutoff: a number above which sums are set to NA, defaults to 10.
    if (!is.vector(x))
        stop('x must be a vector')
    if (!is.numeric(x))
        stop('x must be numeric')
    y = NULL
    for(i in 1:length(x)) {
        y[i] = sum(x[1:i])
        if (y[i] > cutoff) {
            y[i] = NA
        }
    }
    return(y)
}

cum_sum_cutoff(1:10)
```

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

```r
cum_sum_cutoff(runif(30))
```

```
##  [1] 0.5214130 0.9956462 1.4311108 1.8560051 2.6463848 3.4275354 4.2616269
##  [8] 4.4141576 4.8166875 5.0748785 5.4714840 5.7773639 6.4046439 6.7199857
## [15] 7.3092145 8.0798216 8.4298758 8.4894843 9.4291174 9.8284047        NA
## [22]        NA        NA        NA        NA        NA        NA        NA
## [29]        NA        NA
```

Notice above I have defined a new variable called `cutoff` which I use to vary what the cutoff of the cumulative sum is. Also notice that I used a fairly informative function name. This name is pushing the upper limits of how long an object name to shoot for. Text completion helps to deal with cumbersome names.