R Tips: Looping

PSCI 8357, Spring 2016
January 22, 2016

In this week's assignment—and, most likely, future ones—you will want to perform the same operation on many different values. We typically do this in R (and any other programming language) with a *loop*. This brief note will introduce you to the concept of loops and illustrate some especially convenient ways to write loops in R.

`for` Loops

Use a loop when you have:

1. A list of values
2. An operation to perform on each of those values

For example, suppose you want to:

- Draw $N = 5$ values from a standard normal distribution
- Calculate their maximum and save it
- Repeat for $N = 10, 15, 20, 25$

So you have the list of values $N = 5, \ldots, 25$ and the operation "draw $X_1, \ldots, X_N$ from $N(0, 1)$ and calculate their mean". One way you could do this is the copy-and-paste way:

```
max_5 <- max(rnorm(5))
max_10 <- max(rnorm(10))
max_15 <- max(rnorm(15))
max_20 <- max(rnorm(20))
max_25 <- max(rnorm(25))

c(max_5, max_10, max_15, max_20, max_25)
```

```
## [1] 0.224 1.948 2.189 1.150 1.959
```

Don't do this. First, it doesn't scale well. Imagine how much time you'd waste if you had to copy and paste this a hundred, or a thousand, or 300 million times. Second, it's error prone. At some point you *will* mess up and enter the wrong thing. Third, it's a nightmare if the operation changes—you'll have to go back and manually fix every line of code.

A better way would be to write a loop, like in the following example.

```
## Define the sequence of values
n_seq <- seq(5, 25, by = 5)

## Make a vector to store the results
max_by_n <- rep(NA, length(n_seq))

for (i in 1:length(n_seq)) {
    ## Retrieve the corresponding value
    n <- n_seq[i]

    ## Perform the operation and store the results
    x <- rnorm(n)
    max_by_n[i] <- max(x)
}

max_by_n
```

```
## [1] 1.402 0.877 1.649 1.366 2.469
```

When we write `for (i in vector) { operation }`, we are telling R to take each individual value of `vector`, call it `i`, and perform `operation` on it. So here, we take each value of `i` between 1 and 5, find the corresponding value of N, draw N observations from a standard normal distribution, and save their maximum.

`foreach` Loops

The code above is a bit clunky. Before we write our loop, we have to set up storage for the results. Instead of looping directly over N, we loop over the number of different values of N we'll be examining. We can write loops more elegantly using the **foreach** package.

```
library("foreach")
```

The basic syntax of a foreach loop is `foreach (i = vector) %do% { operation }`. Notice that we've replaced `for` with `foreach`, replaced `in` with `=`, and added `%do%` between the loop setup and the operation. But there is an importance difference between `for` and `foreach`.

- `for` just performs the operation you ask it to. If you don't tell it to save the results of the operation, it'll perform it and then forget.
- `foreach` saves the result of each operation it performs—specifically, the value of the last line of the operation—and returns the results as a list.

So we can rewrite our earlier loop much more elegantly using `foreach`. The exact values of the output will be different than before since we're drawing random numbers, but this code performs the same task:

```
n_seq <- seq(5, 25, by = 5)

max_by_n <- foreach (n = n_seq) %do% {
    max(rnorm(n))
}

max_by_n
```

```
## [[1]]
## [1] 0.666
##
## [[2]]
## [1] 0.175
##
## [[3]]
## [1] 1.79
##
## [[4]]
## [1] 1.76
##
## [[5]]
## [1] 1.85
```

Notice that we don't have to explicitly set up a place to store the results, and we can loop over N instead of $1, \ldots, 5$. The drawback is that the results are

stored as a list, and we'd like a vector. We can accomplish that by using the `.combine` argument to `foreach`:

```
n_seq <- seq(5, 25, by = 5)

max_by_n <- foreach (n = n_seq, .combine = "c") %do% {
    max(rnorm(n))
}

max_by_n
```

```
## [1] 1.12 1.63 1.41 1.59 1.24
```

For operations that spit out a vector of numbers, you can stack them using `.combine = "rbind"` or place them in columns with `.combine = "cbind"`. Both of these assume that each vector of output will be the same length—if not, then you'll have to take the output as a list.

When to Avoid Looping

For many common operations, you don't have to write a loop—R does it for you. For example, imagine you want to take the square root of each of a vector of numbers. The `sqrt()` function automatically loops over them.

```
y <- 1:10
sqrt(y)
```

```
##  [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

We call functions like `sqrt()` *vectorized*, because they operate on each element of a vector. Before using a `for` loop, always check for a vectorized function that does what you want.