8.5: Implicit Loops

There's one last topic I want to discuss in this chapter. In addition to providing the explicit looping structures via `while` and `for`, R also provides a collection of functions for ***implicit loops***. What I mean by this is that these are functions that carry out operations very similar to those that you'd normally use a loop for. However, instead of typing out the whole loop, the whole thing is done with a single command. The main reason why this can be handy is that – due to the way that R is written – these implicit looping functions are usually about to do the same calculations much faster than the corresponding explicit loops. In most applications that beginners might want to undertake, this probably isn't very important, since most beginners tend to start out working with fairly small data sets and don't usually need to undertake extremely time consuming number crunching. However, because you often see these functions referred to in other contexts, it may be useful to very briefly discuss a few of them.

The first and simplest of these functions is `sapply()`. The two most important arguments to this function are `X`, which specifies a vector containing the data, and `FUN`, which specifies the name of a function that should be applied to each element of the data vector. The following example illustrates the basics of how it works:

```
words <- c("along", "the", "loom", "of", "the", "land")
sapply( X = words, FUN = nchar )
```

```
## along   the  loom    of   the  land
##     5     3     4     2     3     4
```

Notice how similar this is to the second example of a `for` loop in Section 8.2.2. The `sapply()` function has implicitly looped over the elements of `words`, and for each such element applied the `nchar()` function to calculate the number of letters in the corresponding word.

The second of these functions is `tapply()`, which has three key arguments. As before `X` specifies the data, and `FUN` specifies a function. However, there is also an `INDEX` argument which specifies a grouping variable.[140] What the `tapply()` function does is loop over all of the different values that appear in the `INDEX` variable. Each such value defines a group: the `tapply()` function constructs the subset of `X` that corresponds to that group, and then applies the function `FUN` to that subset of the data. This probably sounds a little abstract, so let's consider a specific example, using the `nightgarden.Rdata` file that we used in Chapter 7.

```
gender <- c( "male","male","female","female","male" )
age <- c( 10,12,9,11,13 )
tapply( X = age, INDEX = gender, FUN = mean )
```

```
##   female     male
## 10.00000 11.66667
```

In this extract, what we're doing is using `gender` to define two different groups of people, and using their `ages` as the data. We then calculate the `mean()` of the ages, separately for the males and the females. A closely related function is `by()`. It actually does the same thing as `tapply()`, but the output is formatted a bit differently. This time around the three arguments are called `data`, `INDICES` and `FUN`, but they're pretty much the same thing. An example of how to use the `by()` function is shown in the following extract:

```
by( data = age, INDICES = gender, FUN = mean )
```

```
## gender: female
## [1] 10
## ------------------------------------------------------------
## gender: male
## [1] 11.66667
```

The `tapply()` and `by()` functions are quite handy things to know about, and are pretty widely used. However, although I do make passing reference to the `tapply()` later on, I don't make much use of them in this book.

Before moving on, I should mention that there are several other functions that work along similar lines, and have suspiciously similar names: `lapply`, `mapply`, `apply`, `vapply`, `rapply` and `eapply`. However, none of these come up anywhere else in this book, so all I wanted to do here is draw your attention to the fact that they exist.
