Chapter 8: Your Own Functions!

Tyson S. Barrett

Summer 2017

Utah State University

Introduction

Introduction

User-defined functions — functions you create yourself — are an important tool.

We will show how you can make your own functions.

What is a Function Anyway?

It is a special `R` object that performs something on another object or the environment.

For example,

- `ggplot()` takes a `data.frame` object and variable names and produces a plot.
- `lm()` takes a formula, a `data.frame` and performs a statistical model.
- `gather()` takes a `data.frame` and produces another `data.frame` in another form

Creating a Function

Creating a Function

It all starts with the function `function()`.

```
myfunction <- function(arguments){
  stuff = that(you)
  want = it + to + do + 4
  you
}
```

After `function()` we use `{` and `}`. Everything in between is what you want the function to do.

All you need to do is run the function and you can use it in the `R` session.

Named Functions

Named Functions

These are functions that you assign a name to.

```
mean2 <- function(x){
  n <- length(x)
  m <- (1/n) * sum(x)
  return(m)
}
```

Now, `mean2()` is a function you can use.

What does this function do?

Named Functions

```
v1 <- c(1,3,2,4,2,1,2,1,1,1)   ## vector to try
```

Let's give it a try using the vector `v1`

```
mean2(v1)                      ## our function
```

```
## [1] 1.8
```

```
mean(v1)                       ## the base R function
```

```
## [1] 1.8
```

Named Functions

For practice, we will write one more function. Let's make a function that takes a vector and gives us the N, the mean, and the standard deviation.

```
important_statistics <- function(x, na.rm=FALSE){
  N  <- length(x)
  M  <- mean(x, na.rm=na.rm)
  SD <- sd(x, na.rm=na.rm)

  final <- c(N, M, SD)
  return(final)
}
```

Named functions

One of the first things you should note is that we included a second argument in the function seen as `na.rm=FALSE` (you can have as many arguments as you want within reason).

- This argument has a default that we provide as `FALSE`
- We take what is provided in the `na.rm` and give that to both the `mean()` and `sd()` functions.
- Finally, you should notice that we took several pieces of information and combined them into the `final` object and returned that.

Named Functions

Let's try it out with the vector we created earlier.

```
important_statistics(v1)
```

```
## [1] 10.000000  1.800000  1.032796
```

Named Functions

Looks good but we may want to change a few aesthetics. In the following code, we adjust it so we have each one labeled.

```
important_statistics2 <- function(x, na.rm=FALSE){
  N  <- length(x)
  M  <- mean(x, na.rm=na.rm)
  SD <- sd(x, na.rm=na.rm)

  final <- data.frame(N, "Mean"=M, "SD"=SD)
  return(final)
}
important_statistics2(v1)
```

```
##    N Mean       SD
## 1 10  1.8 1.032796
```

Anonymous Functions

Anonymous Functions

Sometimes it is not worth saving a function but want to use it, generally within loops.

```
function(x) thing(that, you, want, it, to, do, with, x)
```

We will show a few of these in the looping section (although they are identical in nature to named functions, they just aren't named)

Why Write Your Own?

Why Write Your Own?

Several reasons exist.

- Looping
- Adjusting output
- Performing a special function
- Other customization

We are going to talk about looping in depth.

Looping

Writing your own functions for looping is very common and practical.

Loops are things that are repeated.

For example:

- We may want a certain statistic (like a mean) for every continuous variable in the data set.
- We may want to remove 999 from every variable in a data set.
- We may want to change variable types of certain variables across the whole data set.

Looping

Examples in `R` include:

- `for` loops
- the `apply` family of functions

For Loops

```
for (i in 1:10){
  mean(data[, i])
}
```

For Loops

Another example:

```
library(tidyverse)
data = read.csv("~/Dropbox/Teaching/R for Social Sciences/D
  select(Prod1, MentalApt, PhysApt, Income, Children, SubsU
thing = list()
for (i in 1:8){
  thing[[i]] = cbind(mean(data[, i]), sd(data[, i]))
}
```

For Loops

```
thing
```

```
## [[1]]
##      [,1]     [,2]
## [1,]  3.2 1.146423
##
## [[2]]
##      [,1]     [,2]
## [1,]  5.2 2.305273
##
## [[3]]
##          [,1]     [,2]
## [1,] 5.733333 1.869556
##
## [[4]]
```

For loops

I like for loops. They are easy to understand and fiddle with, after some practice.

However, they used to be slow in `R` and so they have a bad reputation.

The apply family

There are several `apply` functions in `R` that do loops for you.

- `apply()`
- `sapply()`
- `lapply()`
- `tapply()`

sapply

Produces a vector based on the function that it is repeating.

Both do the same thing here.

```
for (i in 1:10){
  mean(data[, i])
}
sapply(1:10, function(i) mean(data[, i]))
```

sapply

Can also just provide the `data.frame` and it assumes you want the function (in this case `mean()`) applied to each variable.

```
sapply(data, mean)
```

```
##       Prod1  MentalApt     PhysApt      Income    Children
##  3.2000000  5.2000000  5.7333333 53.3333333  0.4000000
##       Ment1       Ment2
## 13.8666667  6.0666667
```

lapply

Produces a list. Just like `sapply()` it takes a `data.frame` and a function and applies it across the variables.

```
lapply(data, mean)
```

```
## $Prod1
## [1] 3.2
##
## $MentalApt
## [1] 5.2
##
## $PhysApt
## [1] 5.733333
##
## $Income
## [1] 53.33333
```

tapply

Is a bit different than the rest. It doesn't do much in terms of looping necessarily (although you can have it do that). Instead, it applies a function based on a grouping variable. With the `tidyverse`, however, this is not often used any more.

```
tapply(data$Income, data$Children, mean)
```

```
##        0        1        2
## 53.18182 55.00000 52.50000
```

How could you do this in the `tidyverse` framework?

Loops with User-Defined Functions

Loops with User-Defined Functions

Going back to our `important_statistics2()` function:

```
important_statistics2 <- function(x, na.rm=FALSE){
  N  <- length(x)
  M  <- mean(x, na.rm=na.rm)
  SD <- sd(x, na.rm=na.rm)

  final <- data.frame(N, "Mean"=M, "SD"=SD)
  return(final)
}
```

Let's put it in a loop.

Loops with User-Defined Functions

```
lapply(data, important_statistics2)
```

```
## $Prod1
##    N Mean       SD
## 1 15  3.2 1.146423
##
## $MentalApt
##    N Mean       SD
## 1 15  5.2 2.305273
##
## $PhysApt
##    N     Mean       SD
## 1 15 5.733333 1.869556
##
## $Income
```

Loops with User-Defined Functions

```
sapply(data, important_statistics2)
```

```
##       Prod1     MentalApt PhysApt   Income    Children  Subs
## N     15        15        15        15        15        15
## Mean  3.2       5.2       5.733333  53.33333  0.4       0.26
## SD    1.146423  2.305273  1.869556  12.63027  0.7367884 0.45
##       Ment2
## N     15
## Mean  6.066667
## SD    2.65832
```

Loops with User-Defined Functions

It applied our function across the variables in the `data.frame`. So we can easily get information we want.

This was a very simplified version of how I created the `table1()` function in `furniture`.

Conclusions

Conclusions

Writing your own functions takes time and practice but it can be a worthwhile tool in using R.

I recommend you start simple and start soon.

Ultimately, you can make your own group of functions you use often and create a package for it so others can use them too :)