R Users Guide to Stat 201: Chapter 1

Michael Shyne, 2017

Chapter 1: Introduction to Statistics

Congratulations on choosing R as your statistics application. While StatCrunch is a fine program for learning basic statistics, R is rapidly becoming an essential skill for doing statistics whether in further education or in professional environments. Beginning with R now, while taking Stat 201, will give you a good head start.

These guides will give you the basics of for completing the tasks of this course. They are not intended to be comprehensive. The key to learning a new programming language or, well, just about anything is to experiment. Take what you learn from these guides and play around. Try different combinations and see what happens. And remember, Google is your friend. There are many sites with a lot of helpful information on R that are just a few search terms away.

Since there is not much to do statistics-wise in Chapter 1, this first guide will serve as an introduction to the R language and some helpful tools.

Note to programmers

If you have worked with other programming languages, much of that experience will be a benefit when working with R. However, there are a few differences to be aware of.

First, and the difference that has taken me the longest to get used to, is the period. Whereas in many languages, the period is used as an operator to access data or functions within an object or other data structure, in R the period is an acceptable character to use in variable and function names, like the underscore. Thus, the command to conduct a t-test is "`t.test(...)`" and "`my.variable`" is a perfectly good variable name.

Second, the standard assignment operator is "`<-`" (that is less-than and dash). The more typical "`=`" will also work in most situations, but there is a difference between them in how assignments are performed. There is a chance you will get unexpected results using the equal sign. It is a good habit to develop to just use the arrow for assignment.

R and R Studio

First you need to install R. I won't go into details here, since they will vary by operating system. Start by going to https://www.r-project.org/. Immediately after R is installed, download and install R Studio (https://www.rstudio.com/). While you could use the base R environment, R Studio offers many features that make writing code easier. Also, it is integrated with a number of technologies that will be helpful in your future statistics endeavors, such knitr with R markdown for producing reports integrated with R code and output which can be exported to PDFs or the web (which I am using to create these guides), and Shiny for creating interactive web apps. And it is also free.

Getting started in R

When you open R Studio, you will see a window divided into several panels. One panel will be the console. This is where you can directly type R commands, one at a time, and observe their results. In these guides, unless stated otherwise, the commands presented should be typed at the "`>`" prompt in the console.

However, if you want to be able to save your code for future reference or when you start tackling more complicated tasks, you can write your code in a script file (File > New File > R Script). From within a script file, you can execute a single command or group of commands with the "Run" button or execute the whole file with "Source".

In the console, a question mark followed by any command will open its documentation page in a separate panel. Try `?mean` (and "Enter"). You will use this feature often.

Functions and parameters

The documentation page for `mean` gives the "usage" as follows:

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

The first line gives the most generic version of the function. You must give it something as a parameter (`x`). The ellipse (`...`) means that you can also pass additional parameters which the function might or might not use. The second version (default S3 method) is more interesting. Parameters listed without an equal sign (`=`) are required parameters. In this case, there is just one (`x`), but many functions will have several required parameters. Parameter with equal signs are option parameters. The value after the equal sign is the default value that will be used by the function if no other value is provided.

When calling a function, you can just pass in values, in order and separated by commas, for each parameter. Alternatively, you can "name" each value you pass to the function. In that case, you can enter them in any order you like. Or you provide the first few values, unnamed and in order, and then add any named parameters you need. This is the most common approach. Call the function with the required parameters unnamed, and then name any of the optional parameters you need. See the examples below. Don't worry about the details of the function calls yet.

```
# Find the mean of the varaible 'my.data',
#   parameters trim and na.rm keep their default values
mean(my.data)

# Find the mean, with trim of 0.1 and na.rm TRUE
mean(my.data, 0.1, TRUE)

# Same as above
mean(trim=0.1, na.rm=TRUE, x=my.data)

# Find the mean of my.data, na.rm TRUE,
#   with trim keeping its default value of 0
mean(my.data, na.rm=TRUE)
```

Important

R is case-sensitive. This means that the functions `sum()` and `Sum()` are different functions, one of which doesn't exist, and that `x` and `X` are different variables. This can lead to great confusion if you are not careful or aware. By convention, most R functions and variables are all lowercase with "words" separated by periods or underscores.

Vectors

The fundamental data type in R is the vector, basically an ordered list of values. To create a vector, you can use the `c()` function (short for "concatenate", I imagine). The following sets a vector of the first four integers to the variable `x`:

```
x <- c(1,2,3,4)
```

We can also create a list of consecutive integers with a colon (:) between the lower and upper bounds of our desired list. The following command is equivalent to the first:

```
x <- 1:4
```

To see the contents of a variable, simply type its name.

```
x
```

```
## [1] 1 2 3 4
```

Notice the 1 in brackets on the left. That is the index of the first item displayed for a row. If we have a long enough vector to print on a second row, we will see that value change.

```
x <- 1:25
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25
```

Even a single value is stored as a vector. Notice the index still displays on the output.

```
y <- 2+3
y
```

```
## [1] 5
```

Working with vectors

Vectors can be used (almost) anywhere you would normally expect a single value to be used. In most cases operations or functions which expect a single value or pair of values, when given vectors, will do entry-wise calculations. For example, adding two vectors will add the first entries in the vectors, then the second entries, etc.

```
x <- 1:4
y <- 11:14

# We don't have to store results in a variable. An operation
#   without assignment will just output results.
x + y
```

```
## [1] 12 14 16 18
```

If presented with a vector and a single value, R will apply the single value to each entry of the vector.

```
# Square a vector (x still represents the same vector as above)
x ^ 2
```

```
## [1]  1  4  9 16
```

Actually, what is going on here is that when R encounters an operation on two vectors and one vector is shorter than the other, it will repeat copies of the shorter vector until it is as long as the longer vector. This can produce some confusing results if you are not expecting it, but it can be utilized to your advantage in some situations.

```
# Make alternating entries negative
x <- 1:10
x * c(1,-1)
```

```
##  [1]   1  -2   3  -4   5  -6   7  -8   9 -10
```

Of course, vectors can be used in more expected ways, in functions that operate on lists of values.

```
x.mean <- mean(x)
x.mean
```

```
## [1] 5.5
```

If we need to access individual or multiple items in a vector, we can use square brackets (`[]`) with an index number or numbers. To access multiple indexes, use a vector of values inside the brackets.

```
# Get the fifth integer (i.e. 5)
x[5]
```

```
## [1] 5
```

```
# Get the third, fourth and sixth squares
(x^2)[c(3,4,6)]
```

```
## [1]  9 16 36
```

Data frames

We can think of a vector as one factor or variable (in the stats sense), or as one column in a data table. To work with a data table as a whole, we use data frames. We can create a data frame from already defined vectors.

```
x <- 1:4
x.sqr <- x^2

# Create columns by setting column names = vectors of equal length
squares.df <- data.frame(num=x, square=x.sqr)
squares.df
```

```
##   num square
## 1   1      1
## 2   2      4
## 3   3      9
## 4   4     16
```

The left-most numbers in the output are row numbers. Across the top are the column names we chose, with the values of the column below. While just typing the variable name is fine for our little data frame, most data sets will have hundreds, if not thousands, of rows and perhaps dozens of columns. It is impracticable to examine such data frames this way (though R will not print all of thousands of rows, it will stop printing after the first one thousand).

The first command when working with a data set will almost always be `str()` (for structure, not string as in many other languages). This can be used with any variable, but it is particularly useful with data frames.

```
str(squares.df)
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ num   : int  1 2 3 4
##  $ square: num  1 4 9 16
```

As you can see `str` gives us the number of rows (`obs.`) and columns (`variables`), as well as the names of the column and the first few values in each column. If we want get a sense of how the data looks, without printing hundreds of rows, the `head{}` function will output the first 6 rows by default, or a specified number of rows with the optional `n=` parameter.

```
head(squares.df, n=2)
```

```
##   num square
## 1   1      1
## 2   2      4
```

Additionally, to view the data in a spreadsheet like format, try the `View(my.dataframe)` command (remember R is case-sensitive).

Accessing data in data frames

Now that we have our data in a data frame, we want to get it out so we can work with it. There are several ways to do this.

To retrieve a column of the data frame (a vector), we can use the `$` operator. The format of the operator is `data.frame$column.name`.

```
squares.df$square
```

```
## [1]  1  4  9 16
```

Like with vectors, we can also use square brackets, but now we are working with two dimensions. Both need to be included within the brackets, rows then columns separated with a comma.

```
# Get the third square. The squares are the second column.
#   So we want the 3rd row and 2nd column.
squares.df[3,2]
```

```
## [1] 9
```

With square brackets, we can use column names instead of numbers if we put the names within quotes (single or double). Note, we didn't use quotes around the column name when we used the dollar sign.

```
squares.df[3,'square']
```

```
## [1] 9
```

We can use multiple indices, like with vectors.

```
# Get the third and fourth squares
squares.df[c(3,4),'square']
```

```
## [1]  9 16
```

If we leave either the row or the column indicator blank, then all rows or all columns, respectively, will be included. We must still use the comma.

```
# The following are all equivalent
squares.df$square         # The square column
```

```
## [1]  1  4  9 16
```

```
squares.df[ , 2]          # all rows of the 2nd column
```

```
## [1]  1  4  9 16
```

```
squares.df[ , 'square']    # all rows of the square column
```

```
## [1]  1  4  9 16
```

Getting data

R has a number of built in data sets for use in testing code and demonstrations. I will be using these data sets in the following guides so you can use the example code without needing an external data file. You can see the list of data sets with the following command (I suppressed the output because the list is quite long):

```
data()
```

To use a data set, say the `chickwts` set:

```
# Make the chickwts data set available
data('chickwts')

# Examine the structure of chickwts
str(chickwts)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

```
# Find the mean chick weight
mean(chickwts$weight)
```

```
## [1] 261.3099
```

You can find more information about a built-in data set with the question mark (i.e. `?chickwts`).

Of course, eventually we want to work with our own data. R can import data from may kinds of files (Excel, SPSS, SAS, etc.), but the most common and easiest to work with are csv file (comma separated values). Because I don't want to bother remembering all the different commands for handling different file types, if I have to work with a non-csv file, I will import into Excel and save it as a csv file.

```
my.data <- read.csv('...filepath...', header=TRUE)
```

The value of the first parameter will be the path to the location of the file, including the file name. If you are using R Studio (and this is reason enough to do so, in my opinion), after you type `read.csv` and the first parentheses and the first quote, hit the TAB key. A mini file browser will pop-up that will allow you to select the location of the file and insert the file path in the command. This can be a huge time saver.

The `read.csv` function has many optional parameters (see `?read,csv`) most of which you will never have to bother with. However, the `header` parameter is important. A `TRUE` value for `header` indicates that the first row of the file contains the names of the columns, rather than data. A false value means the file only contains data. If you are not sure which is the case with your file, you can open it in a text editor, or call `read.csv` with `header=FALSE` and then call `head(my.data)`. If the first row of values are names, then read in your data again with `header=T`. (Note: The values `TRUE` and `FALSE` can be abbreviated to `T` and `F`.)

Getting data from MyStatLab

One of the primary advantages to using StatCrunch for Stat 201 is the ease which data from homework problems is imported It is literally as easy as pushing a button (or rather, selecting a item from a menu). Using R will require a few extra steps. I believe the simplest method is to import the data into Excel (from the same menu that allows importing into StatCrunch) and saving the data as a csv file. Then, it can be imported into R as described above.

Conclusion

That's enough to get started with, I believe. If you have further questions, there are many resources on the internet, including several sites dedicated to learning R, YouTube videos and online courses (check out Khan's Academy, Coursera, Udacity, etc).

License