Working with R

William H. Knapp III

March 25, 2016

1 Tips Before You Get Started

Like Git, R uses a command line interface. In other words, instead of clicking buttons to do things, you type in commands at the command prompt. Unlike Git which used a dollar sign as it's prompt, R has two types of prompts: the right arrow (i.e. ">") and the plus sign (i.e. "+"). > indicates that R is done doing whatever it has been asked to do and is waiting for input. + indicates that R is waiting for more information before it.

The biggest problem that students have with RStudio has to do with conscientiousness. R is case sensitive, so "M" and "m" are two different things. When I give you commands to enter, make sure you enter them exactly as I give them to you. The one thing that R isn't picky about is spacing. Thus 1+2 will equal the same thing as 1 + 2. So you can feel free to introduce spaces if it helps keep things more organized for you.

Throughout this document I'll show you two things. Generic code that serves as a template for the code you'll need to run. I will format that code as follows:

```
t.test(group1,group2)
```

Remember the template code is just a guide that you'll need to edit to have it do what you want it to. Speaking of editing, many students express frustration at having to learn a new way of interacting with programs (e.g. using the command prompt). Don't get frustrated. Even if we were using a program with buttons to press, you'd still have to learn which ones to press in which order, so keep an open mind as you get experience.

The second thing that I'll show you is actual R code that you should attempt to run. Each line of the code I want you to run will start with either the > or + command prompts. When you try to run the code, do not include the > or + prompts or your code might not work. If you're copying and pasting my commands into an .Rmd file, make sure you get rid of the > and + symbols at the beginning of the lines. This is easily accomplished by going to the first line with the command prompt and deleting the prompt and any intervening spaces between your commands. Then press the down arrow, if you have multiple lines copied and do the same thing until the code in your .Rmd files have been stripped of the command prompt symbols.

I mentioned that the biggest problem is conscientiousness and instructed you to make sure that you're entering the commands exactly as I have them. Along with this has to do with closing brackets, parentheses, and quotes in the right order. If you enter a command and then are surprised to see the + command prompt, most likely you forgot to close a bracket (i.e. "]"), a parenthese (e.g. ")"), a curly bracket (i.e. "}"), or a quote (i.e. """). If you are returned to the + command prompt unexpectedly review the code that came previously and make sure that each open bracket, curly bracket, parenthese, and quote are associated with an end bracket, curly bracket, parenthese, or quote.

2 Introduction To Vectors

A vector is just a simple collection of values that are all of the same type. Let's use a simple example. We'll create a vector of digits from 1-10.

When I give you these commands I recommend that you first enter them into an .Rmd file in the editing panel and run them from there. There are a couple ways to run the commands I give you. If you only have a command that takes up a single line, you can click any where in that line and click "Run" in the editing window to run the code. If you have multiple lines to run at once, highlight everything you want to run and click run. I like using keyboard shortcuts, so when I want to run a command, I place my cursor in the line I want to run or highlight the lines I want to run and press Ctrl+Enter (Windows, I think Mac is Cmd+Enter, but you'll have to check). Ok, let's create a vector of numbers from one to ten.

There are often multiple ways to arrive at the same answer, so I'll start with the easiest example and we'll see other ways that could work too later.

Unlike the template code, the R code I'd like you to run will be formatted as follows. If there is output associated with the commands you're entering, the output will follow the commands. You can differentiate between the codes and the output because the codes will start on lines with either the > or + command prompts, wheras the output won't start with those prompts.

Ok, let's generate a vector from 1-10.

```
> 1:10

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

In the output above, you can see two lines. The first starts with the command prompt > (remember don't copy that or erase it). The second line starts with [1] and then lists the numbers 1-10. The [1] tells you the index number of the first item preceding it. Thus, the first number in our vector is 10.

Now let's do this a little differently and we'll create a vector of numbers from 1-50. We could use the 1:50 form, but I want to show you another way of creating vectors. For this example, we'll use the seq() function to create a sequence. The template for simple sequences is as follows

```
seq(from, to, by)
```

From represents the starting number, to represents the ending number, and by represents the size of the steps you want to take to get from the beginning number to the ending number.

```
> seq(1,50,1
+     )

 [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 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

Notice again, we see the commands following the command prompt followed by the list of numbers. Also notice how there's now a second command line that starts with a +. This is because I didn't include the ending parenthese at the end of the sequence command.

Notice how each line in the output starts with an index number in brackets that indicates which item in the list the following value is associated with. A common source of frustration for students is that the index numbers you'll see on your screens might differ from what you see in these documents. The reason for this is because R will wrap it's output depending on the size of the panel. The important thing is that your output matches my output.

When working with numbers, it's easy to create vectors using the : function or the seq(function). If you want to create a vector of text, it's often easier to use the concatenate function: c(). Concatenate means to join two or more things together. So let's create a list of genders.

```
> c("female", "male", "other")
```

```
[1] "female" "male"   "other"
```

Now we have a vector of size three that contains our genders. Perhaps we wanted to create a list in which each gender appears a particular number of times. To accomplish this we'll use the replicate function rep() in conjunction with the function c(). The replicate function takes two primary forms.

```
rep(vectortoreplicate, times=numberofreplications)
rep(vectortoreplicate, each=numberofreplications)
```

Let's try each of these forms to see what they do. In the times variation we'll repeat the whole vector the number of times we want.

```
> rep(c("female","male"),times=2)
```

```
[1] "female" "male"   "female" "male"
```

Notice how the vector is female male female male. If instead we use the "each" argument our vector will be female female male male.

```
> rep(c("female","male"),each=2)
```

```
[1] "female" "female" "male"   "male"
```

3 Assigning Variables

Often we'll want to save the output of the commands we use to a variable that we can access later. We assign the output of our commands to a variable by using the assignment operator (i.e. "<-"). The template for assignment is as follows:

```
variablename<-Rcommands
```

Let's assign our previous operation to a variable.

```
> gender<-rep(c("female","male"),each=2)
```

In this example, we assigned the variable gender (you can name variables as you see fit), but I recommend using meaningful names to make it easier for you to remember and for others to follow.

Notice how in the above statement, we didn't see any output. If you ran the line in the R console, you'll be returned to the > command prompt without any indication of success. In R, being returned to the > command prompt without any errors means that everything worked well.

There are two different ways to check that R did the right thing. In the Environment/History panel in the Environment tab, you should see that gender is a vector of 4 characters (i.e. chr [1:4]) and the first couple values in the vector. The other way is to ask R to display the contents of gender. We can do this by entering the variable name into the R console.

```
> gender
```

```
[1] "female" "female" "male"   "male"
```

4 Checking A Variable's Structure

Often when you're working with data, especially with data with which you're unfamiliar, it's a good idea to check the structure of the data. We do this using the str() function.

```
> str(gender)
```

```
 chr [1:4] "female" "female" "male" "male"
```

The above tells us that we have a vector of characters (i.e. chr). That there are four elements in the vector (i.e. [1:4]). We also see the first few elements of the variable. Because our vector is so small, you'll probably see all the values, but with larger datasets you'll only see the first few values.

I recommend checking the structure as you're progressing through the assignments regularly to make sure that the variables you're using are structured in the way you think they are.

5 Logic in R

R allows us to compare values to see if they're the same or not using logic. We evaluate equality using double equals signs (i.e. "=="). As an example, let's see which of our observations in the variable gender are female. Let's run the code and then discuss it.

```
> gender=="female"
```

```
[1]  TRUE  TRUE FALSE FALSE
```

This code says go through the variable gender and determine whether each observation is female or not. If it's female, the result will be true. If it's not, the results will be false.

We can find particular observations of some variable by using index numbers or logic. For example, let's get the first two observations in gender.

```
> gender[1:2]
```

```
[1] "female" "female"
```

Notice how I included the 1:2 in brackets. The brackets represent the indices. Thus we're asking for the first two values from our data set. Alternatively, we could ask for all the values that are true according to logic. For example:

```
> gender[gender=="female"]
```

```
[1] "female" "female"
```

Again we see the same output. But here, we're saying for the gender variable, find all of the observations that are female. This might not seem like a big deal right now, but it will be crucial when working with data frames.

6 Dataframes

A dataframe is a collection of vectors that are all of the same length. The data that you collect and enter into a .csv file will be turned into a data frame. Like the vectors, dataframes can be saved as variables. There's a little bit of special notation when dealing with the variables inside the data frames, but we'll get to that in a moment.

We already have a vector of gender. Let's create a vector for some data associated with the participants we observed and collected genders for.

```
> dat<-c(10,20,15,25)
```

Remember assigning values to a variable won't produce any visible output in the console other than returning you to the command prompt.

Now let's combine our two vectors gender and dat into a single dataframe using the data.frame() command.

```
> df<-data.frame(gender,dat)
```

If you check the structure of df. You'll see that df is a data.frame with 4 observations of two variables. It also indicates what the variables are (notice the lines that begin with $) that are contained in the dataframe and the structure of those variables. Notice how gender is now being treated as a factor variable (i.e. a variable that can only take on a particular range of values). Factor variables will play heavily in some of the analyses that you'll be performing.

To access the variables within a data frame use the following notation.

```
dataframename$variablename
```

Thus, if we wanted to get the dat variable from the df dataframe, we'd do the following.

```
> df$dat
```

```
[1] 10 20 15 25
```

Now here's where the logic comes in handy. Imagine that you had thousands of observations, instead of just four and you wanted to calculate the mean score for females. You could do this manually by finding the index numbers for all the females and concatenating them (e.g. df$dat[c(1,2,...)]). This is a huge pain in the rear, so I'd encourage you to use logic instead. Check out the following.

```
> mean(df$dat[gender=="female"])

[1] 15
```

In plain English what this says is to take the mean of the dat variable only when their associated gender is female.

You can also create more complicated logical statements by using logical ANDs (i.e. "&") and/or logical ORs (i.e. "|"). The | symbol can be entered using Shift+\. Check out the following examples.

```
> 1==1 & 2==2

[1] TRUE

> 1==1 & 2==1

[1] FALSE

> 1==1 | 2==1

[1] TRUE
```

Thus, if our dataframe was much larger and also included things like political affiliation, we could get the mean for female independents using mean(df$dat[df$gender == "female" & df$party == "independent"]).

If we wanted everyone but independents we could use the logical not (i.e. "!="). Literally, != reads as does not equal. Continuing with the same example, we could do something like mean(df$dat[df$gender == "female" & df$party != "independent"]).

7 Reading in .csv Files

7.1 Setting the Working Directory

When working with data files, we need to tell R where the data files we want to work with are located. When I created this project, I included example and homework files for each assignment except the first. When you're examining example.Rmd or working on homework.Rmd. Go up to the upper left hand corner of the RStudio window. Then click "Session," then click "Set Working Directory," and finally click "To Source File Location."

The .Rmd document is the source file you're working with and by setting the working directory to the location of the source file you'll have access to the data files in that directory.

7.2 Reading in the .csv File

Once we're in the appropriate working directory we can read in the .csv file by using the read.csv function and assigning the output to a variable.

```
> dat<-read.csv("example2.csv")
```

Above, I'm reading the .csv file into a variable named dat that will be a dataframe. You can check this by using the str() command.

```
> str(dat)
```

```
'data.frame':        100 obs. of  3 variables:
 $ gender : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 1 ...
 $ party  : Factor w/ 2 levels "democrat","republican": 2 2 2 2 2 2 2 2 2 2 ...
 $ support: num  73.9 61.5 68.2 68.7 66.6 ...
```

Once you have the data loaded up, you're ready to start analyzing or plotting your data. Let's find the standard deviation of the female democrats support scores for some initiative. We'll do this using the standard deviation function sd().

```
> sd(dat$support[dat$gender=="female" & dat$party=="democrat"])
```

```
[1] 7.45434
```

8 Working with .Rmd files

R Markdown (i.e. .Rmd) files are very similar to plain Markdown (i.e. .md) files. You can use the formatting commands you learned in the last reading and you can also embed chunks of R code. A chunk of R code looks like this.

~~~
```{r}
R commands
go in
here.
```
~~~

‘‘‘{r} on the line before the R code indicates the start of an R code chunk and ‘‘‘ on the line following the R code indicates the end of an R code chunk. You can insert chunks manually by typing in the beginning and endingindicators or you can go to the upper right hand of the editing panel and click ”Chunks” and then ”Insert Chunk.” I recommend using the latter method to avoid spelling errors (e.g. missing one ‘ symbol).

If you want to run the code inside of the chunk, just highlight what’s between the starting and ending indicators and run it the way I mentioned in the last reading.

So use what you learned from the last reading and assignment to organize your .Rmd files and use chunks to embed R code. When you knit the document the R code and any output will be displayed in the resulting document.

9 Getting Help

If a function isn’t working the way you expected it to work, you can pull up R’s documentation on the function by typing a ? before the function name. For example:

```
> ?rep
```

This will open the help file in the lower right panel of RStudio.

Unfortunately, the help files can be a bit dense to read. If you can’t figure out the help file, do a search on the internet. There’s a ton of good information out there. If that doesn’t work, contact me. For simple questions email often works well. Just copy and paste the codes you're using and the error messages you're receiving so I have enough context to help. For more complicated problems, Skype is often best so I can see your screen and help guide you along.

10 Completing This Week's Homework

To complete this weeks homework, you'll need to take what you learned from the last assignment and from this reading to edit homework2.Rmd. When you're finished editing homework2.Rmd (make sure it knits so you don't lose points) you should save your work. Then open up Terminal or Git Bash to sync your changes with what's on the internet. Remember, there are three steps to syncing. First add, then commit, then push.