Fourth_Exercise (scripts and indexing)

Joe Thompson

2018-07-25

Using scripts

I have found that many students like to play around with R code. Before they check the help, they will execute lines of code that they think ‘might’ solve their problem. What can be problematic is when students do all their work in the R Studio console and consequently lose track of their progress. Their curiosity (which would normally be a good thing) can become a hindrence. Scripts are a much more useful way of keeping track of your work. There are at least three reasons why.

First, while you may lose track of their progress when relying on the console, scripts can be saved and backed up just like text documents. When scripts become very complex (or there are a number of authors making frequent changes to them), you can even turn to industry tools for keeping track of such changes (e.g., Git).

Secondly, instructors and students can use comments in their scripts to better communicate their thinking to the reader. Contrast, for example, this student’s work.

```
sum(seq(1,10))/2
```

```
## [1] 27.5
```

with this student’s work

```
#Goal: Calculate the mean
Data=seq(1,10)
#step 1: Sum the data
MySum=sum(Data)
#step 2: Divide by n
Mean=MySum/2
```

It is much easier to understand (and spot errors in) well commented code. If you are using R Studio for research, then I would strongly recommend that you establish laboratory-wide norms about how code should be commented.

Finally, scripts are a first step to reproducible work. If you can trace an analysis back to the scripts which produced it, then the analysis will OFTEN be reproducible.

When scripts do not produce reproducible research

It is important to emphasize that scripts use the variable workspace or environment in the top-right hand of R Studio. This means that the same script can produce different results depending on the variables that exist in the workspace. For example, suppose someone studying response time first uses the console to define the variable Data

```
#
Data=c(1,2,0.5,0.7,0.6)
```

and then runs the following script.

```
#Goal: Convert Response time into milliseconds
Data=Data*1000

#plot data
summary(Data)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##     500     600     700     960    1000    2000
```

Notice that each time they run the script they are changing the variable Data. This means that the output of the script will change if they run the script again. e.g.,

```
#Goal: Convert Response time into milliseconds
Data=Data*1000

#plot data
summary(Data)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  500000  600000  700000  960000 1000000 2000000
```

Therefore, you should always be aware that these scripts make use of the variable workspace or Global Environment window. One way to avoid this problem is to clear your variable workspace prior to running important analysis scripts. This can be done with the following code.

```
#Clear variables
rm(list = ls())
```

Excercises I

1. One thing I like to do with students is have a data generation tool that plots data and displays descriptives. Students can then play with the code by altering various parameters (e.g., they might alter the sample size or the population mean). Create a well commented script that could be used as a teaching tool. If you need inspiration, see NormalDistributionPlayground.R.
2. [Conceptual] Reflect on the early cognitive scientists who were thinking of the brain as a computer. Why might it be problematic for an organism to rely on a shared variable workspace for all mental functioning? Hint: What sorts of programming ‘bugs’ might this organism be susceptible to?

Indexing operations

Sometimes you need access to only a small portion of your data, such as the first ten values. e.g,.

```
#Goal: Calculate the mean score on the DV by condition
#Step 1: Generate fake data
DV=c(rnorm(n=10,mean=10,sd=1), rnorm(n=10,mean=0,sd=1))
DV
```

```
##  [1] 10.3421926  8.9827602  9.0127411 11.1495708 11.2863182 11.2833936
##  [7] 10.6364168 11.1933877  8.6214671  9.4803529 -1.1001956  0.9320390
## [13] -0.3094105  0.2186191  0.9256484 -0.0777448  2.1525278 -0.8013717
## [19]  1.6713577  0.7635750
```

Suppose we know that the first ten numbers in the vector 'DV' contains scores for the control condition, and the second ten scores reflect the experimental condition.

We can then use square brackets to index into the first ten values of the DV. To get started, see that

```
DV[1]
```

```
## [1] 10.34219
```

returns the first number in 'DV', and see that

```
DV[20]
```

```
## [1] 0.763575
```

returns the last number. Finally, note that

```
DV[c(1,2)]
```

```
## [1] 10.34219  8.98276
```

returns the first two numbers in the DV. To extract the first ten numbers of the DV, we only need to index into the DV using the vector of numbers from 1 to 10.

```
DV[seq(1,10)]
```

```
##  [1] 10.342193  8.982760  9.012741 11.149571 11.286318 11.283394 10.636417
##  [8] 11.193388  8.621467  9.480353
```

Since this gives us the first ten numbers of the DV (which corresponds to the control condition in this example), we can calculate the mean of the control condition by using the following.

```
# One-line indexing
mean(DV[seq(1,10)])
```

```
## [1] 10.19886
```

Indeces can be a source of confuson and bugs, so depending on your purposes you might prefer to create a well commented script that draws out this process.

```
#Verbose indexing
#Goal: Get mean of the first ten numbers in DV
#Step 1: Create an index from one to ten
IDX=seq(1,10)
#Step 2: Grab the DV scores associated with the control condition
Control=DV[IDX]
#Step 3: Calculate mean
mean(Control)
```

```
## [1] 10.19886
```

Indexing and Logicals

Some R functions and operators return logical statements rather than numbers. They can report, for exampe, whether the mean of our control condition is greater than 1.

```
mean(DV[seq(1,10)]) >= 1
```

```
## [1] TRUE
```

Furthermore, these sorts of comparisons can be solicited for entire vectors of numbers

```
DV>1
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

The output will be the same length of the vector DV, and it will return TRUE wherever the DV is greater than 1.

In future work, you will probably use logicals in order to index into variables. For example,

```
#Goal: Find all scores on DV which are greater than 1
#Step 1: Create logical index
IDX=DV>1
#Step 2: Grab scores on DV which are greater than 1
DV[IDX]
```

```
##  [1] 10.342193  8.982760  9.012741 11.149571 11.286318 11.283394 10.636417
##  [8] 11.193388  8.621467  9.480353  2.152528  1.671358
```

will return all the scores on the DV which are greater than 1 or, equivelently, all the scores on the DV such that IDX is TRUE.

Exercises II

1. Generate a vector (of at least 20 rows) of data and index into the fifth element.
2. Generate a vector and index into the last five rows.
3. Generate a vector and index into the first row, the third row, and the fifth row.
4. [Philosophical] Compare the one-line indexing script to the verbose indexing script. What are the advantages and disadvantages of the verbose version of the script?
5. [Difficult] Get the fiftieth percentile of your vector (you might need to use google to find a useful function), and then use the '>' operator to display any numbers which are larger than the fiftieth percentile.