Stat243: Problem Set 2, Due Friday Sep. 15

September 11, 2017

This covers material in Unit 3.
It's due **on paper** and submitted via Github at the start of class on Sep. 18.
Some general guidelines on how to present your problem set solutions:

1. Please use your Rtex/Rnw/Rmd solution from PS1, problem 4 as your template for how to format your solutions (only non-Statistics students are allowed to use R Markdown).

2. Your solution should not just be code - you should have text describing how you approached the problem and what the various steps were.

3. Your paper submission should be the printout of the PDF produced from your Rtex/Rnw/Rmd file. Your Github submission should include the Rtex/Rnw/Rmd file, any R or bash code files containing chunks that you read into your Rtex/Rnw/Rmd file, and the final PDF, all named according to the guidelines in *howtos/submitting-electronically.txt*.

4. Your code should have comments indicating what each function or block of code does, and for any lines of code or code constructs that may be hard to understand, a comment indicating what that code does. You do not need to show exhaustive output but in general you should show short examples of what your code does to demonstrate its functionality.

5. Please note my comments in the syllabus about when to ask for help and about working together.

6. **Please give the names of any other students that you worked with on the problem set.**

Problem

1. This problem explores file sizes and their relationship to the file format.

   (a) Based on understanding storage in ASCII plain text versus binary formats and on the fact that numbers are stored as 8 bytes per number in binary format, explain the results below. If there are any that seem inconsistent with the class notes and our discussion in class, note that.

```
## save letters in text format
chars <- sample(letters, 1e6, replace = TRUE)
write.table(chars, file = 'tmp1.csv', row.names = FALSE, quote = FALSE,
            col.names = FALSE)
system('ls -l tmp1.csv', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 2000000 Sep 11 07:38 tmp1.csv"
```

```
chars <- paste(chars, collapse = '')
write.table(chars, file = 'tmp2.csv', row.names = FALSE, quote = FALSE,
          col.names = FALSE)
system('ls -l tmp2.csv', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 1000001 Sep 11 07:38 tmp2.csv"

## save in binary format
nums <- rnorm(1e6)
save(nums, file = 'tmp3.Rda')
system('ls -l tmp3.Rda', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 7678421 Sep 11 07:38 tmp3.Rda"

## save in text format
write.table(nums, file = 'tmp4.csv', row.names = FALSE, quote = FALSE,
          col.names = FALSE, sep = ',')
system('ls -l tmp4.csv', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 18158912 Sep 11 07:38 tmp4.csv"

write.table(round(nums, 2), file = 'tmp5.csv', row.names = FALSE,
            quote = FALSE, col.names = FALSE, sep = ',')
system('ls -l tmp5.csv', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 5379375 Sep 11 07:38 tmp5.csv"
```

(b) Now consider the additional results below. Can you explain what is going on and understand what R is doing when it saves an object using *save()*? You might experiment with options to the *save()* function to determine if your explanations are correct.

```
chars <- sample(letters, 1e6, replace = TRUE)
chars <- paste(chars, collapse = '')
save(chars, file = 'tmp6.Rda')
system('ls -l tmp6.Rda', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 635237 Sep 11 07:38 tmp6.Rda"

chars <- rep('a', 1e6)
chars <- paste(chars, collapse = '')
save(chars, file = 'tmp7.Rda')
system('ls -l tmp7.Rda', intern = TRUE)

## [1] "-rw-r--r-- 1 paciorek scfstaff 1056 Sep 11 07:38 tmp7.Rda"
```

2. Go to https://scholar.google.com and enter the name (including first name to help with disambiguation) for a researcher whose work interests you. (If you want to do the one that will match the problem set solutions, you can use “Geoffrey Hinton”, who is a famous machine learning researcher.) If you’ve entered the name of a researcher that Google Scholar recognizes as having a Google Scholar profile,

you should see that the first item returned is a “User profile”. Next, if you click on the hyperlink for the researcher under “User profiles for <researcher name>”, you’ll see that brings you to a page that provides the citations for all of the researcher’s papers.
Note: if you repeatedly query the Google Scholar site too quickly, Google will start returning “503” errors because it detects automated usage. So, if you are going to run code from a script such that multiple queries would get done in quick succession, I’d suggest putting `Sys.sleep(2)` in between the calls that do the HTTP requests. Also when developing your code, once you have the code in part (a) working to download the HTML, use the downloaded HTML to develop the remainder of your code and don’t keep re-downloading the HTML as you work on the remainder of the code.

(a) Now, based on the information returned by your work above, including the various URLs that your searching and clicking generated, write R code that will programmatically return the citation page for your researcher. Specifically, write a function whose input is the character string of the name of the researcher and whose output is the html text corresponding to the researcher’s citation page as well as the researcher’s Google Scholar ID.

(b) Create a second function to process the resulting HTML to create an R data frame that contains the article title, authors, journal information, year of publication, and number of citations as five columns of information. Try your function on a second researcher to provide more confidence that your function is working properly.

(c) Based on the discussion in lab on Monday Sept. 11, include checks in your code so that it fails gracefully if the user provides invalid input or Google Scholar doesn’t return a result. Also write some test code that uses the *testthat* package to carry out a small number of tests of your function.

(d) (Extra credit) Fix your function so that you get all of the results for a researcher and not just the first 20. Hint: figure out what query is made when you click on “Show More” at the bottom of the page.

Hint: as you are trying to understand the structure of the HTML, note that printing the object returned by *htmlParse()* produces nicely formatted text.

Note: For simplicity you can either assume only one User Profile will be returned by your initial search or that you can just use the first of the profiles.