Problem Set 1

Due Wednesday Sep. 7, 10 am

Comments

- This covers material in Units 2 and 4 as well as practice with some of the tools we'll use in the course (R Markdown/knitr).
- It's due at 10 am (Pacific) on September 7, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please note my comments in the syllabus about when to ask for help and about working together. **In particular, please give the names of any other students that you worked with on the problem set and indicate in the text or in code comments any specific ideas or code you borrowed from another student or any online reference.**

Formatting requirements

1. Your electronic solution should be in the form of an R markdown file named ps1.Rmd, a Quarto markdown document named ps1.qmd, or a LaTeX+knitr file named ps1.Rtex, with R code chunks included in the file (or read in from a separate code file). Please see the dynamic documents tutorial for more information on how to do this.
2. Your PDF submission should be the PDF produced from your Rmd/qmd/Rtex. Your GitHub submission should include the Rmd/qmd/Rtex file, any R code files containing chunks that you read into your Rmd/qmd/Rtex file, and the final PDF, all named according to the submission guidelines.
3. Your solution should not just be code - you should have text describing how you approached the problem and what the various steps were. 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 (and should not) show exhaustive output, but in general you should show short examples of what your code does to demonstrate its functionality. Please see the grading rubric.

Problems

1. Please read these lecture notes about how computers work, used in a class on statistical computing at CMU. In particular, make sure you know the difference between disk and memory. You don't need to turn anything in for this problem.

2. This problem explores file sizes in light of understanding storage in ASCII plain text versus binary formats and the fact that numbers are (generally) stored as 8 bytes per number in binary format.

   a. Explain the sizes of the two files created below. In discussing the CSV text file, how many characters do you expect to be in the file (i.e., you should be able to estimate this very accurately from first principles *without* using `wc` or any explicit program that counts characters). Hint: what do we know about numbers drawn from a standard normal distribution?

```
n <-  1e7
a <- matrix(rnorm(n), ncol = 100)
a <- round(a, 10)

fn_csv <- tempfile(fileext = '.csv')
write.table(a, file = fn_csv, quote=FALSE, row.names=FALSE,
               col.names = FALSE, sep=',')
fn_rda <- tempfile(fileext = '.Rda')
save(a, file = fn_rda, compress = FALSE)

file.size(fn_csv)
```

```
[1] 133887318
```

```
file.size(fn_rda)
```

```
[1] 80000096
```

   b. Now consider saving out the numbers one row per number. Given we no longer have to save all the commas, why is the file size unchanged?

```
b <- a
dim(b) <- c(1e7, 1)  ## change to one column by adjusting attribute
fn_csv_onecol <- tempfile(fileext = '.csv')
write.table(b, file = fn_csv_onecol, quote=FALSE,
     row.names=FALSE, col.names = FALSE, sep=',')
file.size(fn_csv_onecol)
```

```
[1] 133887318
```

   c. Consider the following ways of reading the data into R (though similar results would be obtained in other languages). Explain the difference in speed between the three situations. Side note: in this case `readr::read_csv()` is rather faster than `read.csv()`.

```
system.time(a1 <- read.csv(fn_csv, header = FALSE))
```

```
   user  system elapsed
 24.927   0.222  25.167
```

```
system.time(a2 <- read.csv(fn_csv, header = FALSE,
                    colClasses = 'numeric'))
```

```
   user  system elapsed
  2.414   0.076   2.491
```

```
system.time(a3 <- scan(fn_csv, sep = ','))
```

```
   user  system elapsed
  2.317   0.024   2.341
```

d. Explain why `tmp.Rda` is so much bigger than `tmp2.Rda` given they both contain the same number of numeric values.

```
fn1_rda <- tempfile(fileext = '.Rda')
save(a, file = fn1_rda)
file.size(fn1_rda)
```

```
[1] 76779201
```

```
b <- rep(rnorm(1), 1e7)
fn2_rda <- tempfile(fileext = '.Rda')
save(b, file = fn2_rda)
file.size(fn2_rda)
```

```
[1] 116505
```

3. Please read Unit 4 on good programming/project practices and incorporate what you've learned from that reading into your solution for Problem 4. As your response to this question, briefly (a few sentences) note what you did in your code for Problem 4 that reflects the material in Sections 1.2 and 1.3 of Unit 4. Please also note anything in Unit 4 that you disagree with, if you have a different stylistic perspective.

4. Go to Google Scholar 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 "Jennifer Chayes", who is Berkeley's dean for data science.) 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. **IMPORTANT: if you repeatedly query the Google Scholar site too quickly, Google will start returning "503" errors because it detects automated usage (see problem 5 below). So, if you are going to run code from a script such that multiple queries would get done in quick succession, please put something like `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 (as an object of class `xml_document`) corresponding to the researcher's citation page as well as the researcher's Google Scholar ID.

   Hint: you will need to use some string processing functions to extract the Scholar ID (and possibly for other things) from the output of the various *rvest* functions. I recommend functions from the `stringr` package (we'll see these in Unit 5 and you can find information in the string processing tutorial), in particular: `str_detect()`, `str_extract()`, `str_split()`, and `str_replace()`. You should NOT need to use regular expressions (which we'll cover in Unit 5), but you can if you want/know how to. Finally if you get an error message like this "`Syntax error in regexp pattern. (U_REGEX_RULE_SYNTAX)`", it means the string processing function is interpreting the characters you are looking for as a regular expression. Simply wrap the string you are looking for in the `fixed()` function, e.g., `fixed('sdf?=sd34')` so the characters are simply interpreted as regular characters (i.e., interpreted literally).

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. Include checks in your code so that it fails gracefully if the user provides invalid input or Google Scholar doesn't return a result. You don't have to use the `assertthat` package as we won't cover that until Section on Sep. 9, but you can if you want.

d. (Extra credit) Fix your function so that you get all of the results for a researcher and not just the first 20.

Hint: as you are trying to understand the structure of the HTML, one option is to use `write_xml()` on the result of `read_html()` and then view the file that is produced in a text editor.

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.

5. Look at the `robots.txt` for Google Scholar (scholar.google.com) and the references in Unit 2 on the ethics of webscraping. Does it seem like it's ok to scrape data from Google Scholar?

6. (Extra Credit) The `reticulate` package and R Markdown allow you to have Python and R chunks in a document that interact with each other. Demonstrate the ability to use this functionality, in particular sending data from R to Python and back to R, with some processing done in Python (it doesn't have to be complicated processing).