Stat243: Problem Set 1, Due Wednesday Sep. 8

August 30, 2021

Comments:

- This covers material in Units 2 and 4 as well as practice with some of the tools we'll use in the course (knitr/R Markdown).
- It's due at 10 am (Pacific) on September 8, **both submitted as a PDF to Gradescope as well as committed to your GitHub repository** as documented in the *howtos/submitting-electronically.txt* file and discussed in Section 1 on Friday September 3.
- 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.**

Formatting requirements

1. Your electronic solution should be in the form of an R markdown file named *ps1.Rmd* 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 or Section 1 (Sep. 3) materials for more information on how to do this.
2. Your PDF submission should be the PDF produced from your Rmd/Rtex. Your GitHub submission should include the Rtex/Rmd file, any R code files containing chunks that you read into your Rtex/Rmd file, and the final PDF, all named according to the guidelines in *howtos/submitting-electronically.txt*.
3. Your solution should not just be code - you should have text describing how you approached the problem and what the various steps were.
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 (and should not) show exhaustive output, but in general you should show short examples of what your code does to demonstrate its functionality.

Problems

1. As preparation for the next few units, please read the sections titled "Memory hierarchy" and "Cache in depth" in the following brief piece that talks about the difference between the CPU cache, main memory (RAM), and disk. You don't need to follow all the technical details in the "Cache in depth"

section - just try to get the big picture of what the cache is and a bit about how it works. You don't need to turn anything in for this problem.

(Side note: the reference to floppy disks and CDROM indicates that webpage was written a while ago, but of the various documents online that talk about this topic, I thought this was the best quick summary I could find.)

2. This problem makes use of the ideas and tools in Unit 2, Sections 2-3 to explore approaches to reading data from disk in an "online" manner (i.e., in chunks rather than the entire file all at once).

   (a) Generate a CSV file of random real-valued numbers with 10 columns and as many rows as needed that the file is about 100 Mb in size. As part of your answer, show what calculations you did to determine the number of rows.

   (b) Read the file into R using *read.csv()* or *read.table()* and time how long it takes to just read the first 100,000 rows. Repeat the reading a few times. In some cases you might find that the first time is slower; if so this has to do with the operating system caching the file in memory (we'll discuss this further in Unit 8). See if using *colClasses* reduces the time to read the data.

   (c) Now experiment with the *skip* and *nrows* arguments to see if you can read in a large chunk of data from the middle of the file as quickly as the same size chunk from the start of the file. What does this indicate regarding whether R has to read in all the data up to the point where the chunk in the middle starts or can skip over it in some fashion? Is there any savings relative to reading all the initial rows and the chunk in the middle all at once?

   (d) Now read the data sequentially in equal-sized chunks using a connection and determine if reading in the large chunk in the middle (after having read the earlier chunks) takes the same amount of time using a connection as it did in part (c). Comment on what you've learned.

   (e) With your matrix of values in R, save it to a .Rda file using *save()* with the *compress* argument set to FALSE. Explain the size of the resulting file in comparison to the size of the CSV file.

   (f) Now write out a matrix of the same size, but containing all the same value. Using *compress=TRUE* (the default), compare the size of the file with random values to the size with a single repeated value. What does this tell you (roughly) about compression? (You don't need to do any detailed investigation.)

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. We'll experiment with webscraping and manipulating HTML by getting song lyrics from the web. Go to MLDb and (in the search bar in the middle, not at the left) enter the name of a song and choose to search by title and 'all words'. In some cases the search goes directly to the lyrics of the song (presumably when there is no ambiguity) and in others it goes to a table of potential songs with that or similar name. (For example, compare 'Dance in the Dark' (or 'Dancing in the Dark') to 'Leaving Las Vegas'.)

   (a) Based on the GET request being sent to the MLDb server (in the cases like 'Dance in the Dark' where you get a table back rather than a single song's lyrics), determine how to programmatically search for a song by title and 'all words' *using read_html()*. (Side question: what does the *si* parameter control?)

IMPORTANT: it's possible that if you repeatedly query the site too quickly, it 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 working to download the HTML, use the downloaded HTML to develop the remainder of your code that manipulates the HTML and don't keep re-downloading the HTML as you work on the remainder of the code.

(b) Write an overall R function (and modular helper functions to do particular pieces of the work needed) that takes as input a title and artist, searches by the title, and then (based on an exact match to the title and artist in the resulting set of song results) finds the URL of the page for the lyrics for that particular song. Then use that URL and return the lyrics, the artist, and the album(s). You can assume that the song you want is on the first page of results. If no exact match is found, just return NULL.

Hint 1: you will need to use some string processing functions to do basic manipulations, combined with the various *rvest* and *xml2* functions. I recommend functions from the *stringr* package (we'll see these in Unit 5 and you can find information in Section 2.1.2 of 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).

Hint 2: For reasons I don't understand (if you figure out why, please post on Piazza!), if you want to get nested HTML elements, doing things like

```
table_elements <- html_elements(html, "table")
all_a <- html_elements(table_elements, "a")
```

gives back all the 'a' elements, **not** just those within the tables. Instead, syntax like

```
a_in_tables <- html_elements(html, "//table//a")
```

seems to work to get only the "a" elements within tables.

(c) Modify your function so it works either when the lyrics are returned directly from the initial search or when multiple songs are returned. Include checks in your code so that it fails gracefully if the user provides invalid input or MLDb doesn't return a result. You don't have to use the *assertthat* package as we won't cover that until Section on Sep. 10, but you can if you want.

(d) (Extra credit) Modify your code to handle cases (e.g., searching for "Dance with me") that return more than one page of results.

5. Look at the *robots.txt* file for MLDb and 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 MLDb? What about Google Scholar?

6. This problem is actually just the formatting of this document.

(a) Have the document be correctly formatted according to the requirements above. You don't need to explicitly answer this sub-problem.

(b) (extra credit) The *reticulate* package and R Markdown allow you now 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). There's a blog post here to get you started: http://feedproxy.google.com/~r/RBloggers/~3/76l-uQEOoiU/?utm_source=feedburner&utm_medium=email .