Stat243: Problem Set 3, Due Friday Sept. 28

September 17, 2018

This covers the first part of Unit 4 and Unit 5.
It's due **as PDF submitted to bCourses** and submitted via Github at 2 pm on Sep. 28.
Note that problem 1 is due by the end of the day September 24.
Some general guidelines on how to present your problem set solutions:

1. Please use Rmd/Rtex as in previous problem sets.
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 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*.
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. In particular, **please give the names of any other students that you worked with on the problem set and indicate in comments any ideas or code you borrowed from another student.**

Problems

1. On Tuesday, September 25, section will consist of a discussion of good practices in reproducible research and computing, led by Omid and Chris. In preparation, please do the following:

   (a) Read one of these four items:

      i. Read the Preface, the Basic Reproducible Workflow Template chapter and Lessons Learned chapters of the new-ish (Berkeley-produced) book: The Practice of Reproducible Research.
      ii. Gentzkow and Shapiro (http://www.brown.edu/Research/Shapiro/pdfs/CodeAndData.pdf)
      iii. Wilson et.al. (http://arxiv.org/pdf/1210.0530v3.pdf)
      iv. Millman and Perez (https://github.com/berkeley-stat243/stat243-fall-2014/blob/master/section/millman-perez.pdf)

   Based on what you read, please write down a substantive comment or question you have that is raised by this material and submit to this Google form (https://goo.gl/forms/Q9TG16Ml1Ig7WPJR2): **by the end of the day, Monday Sep. 24** as we'll be collating them before section. In addition, include your comment/question here in your problem set solution.

(b) (Nothing to turn in) When reading, please think about the following questions, which we will be discussing in section:

i. Are there practices suggested that seem particularly compelling to you? What about ones that don't seem compelling to you?

ii. Do you currently use any of the practices described? Which ones, and why? Which ones do you not use and why (apart from just not being aware of them)?

iii. Why don't researchers consistently utilize these principles/tools? Which ones might be the most/least used? Which ones might be the easiest/most difficult to implement?

iv. What principles and practices described apply more to analyses of data, and which apply more to software engineering? Which principles and practices apply to both?

(c) (Nothing to turn in) Please skim through the paper *ps/clm.pdf* before section, in particular looking at the Method section, as a good part of our discussion in Section will be about whether one can reproduce what they did in the paper.

(d) Based on our discussion in section, please list a few strengths and a few weaknesses of the reproducibility materials provided for the analysis in *clm.pdf*.

2. The goal of Problem 2 is two-fold: first to give you practice with regular expressions and string processing and the second (more important) to have you thinking about writing well-structured, readable code. Regarding the latter, please focus your attention on writing short, modular functions that operate in a vectorized manner and also making use of *apply()/lapply()/sapply()* to apply functions to your data structures. Think carefully about how to structure your objects to store the debate information. You might have each candidate's response to a question be an element in a character vector or in a list.

   The website Commission on Presidential Debates has the text from recent debates between the candidates for President of the United States. (As a bit of background for those of you not familiar with the US political system, there are usually three debates between the Republican and Democratic candidates at which they are asked questions so that US voters can determine which candidate they would like to vote for.) Your task is to process the information and produce data on the debates. Note that while I present the problem below as subparts (a)-(g), your solution does not need to be divided into subparts in the same way, but you do need to make clear in your solution where and how you are doing what. Your solution should do all of the downloading and processing from within R so that your operations are self-contained and reproducible. For the purposes of this problem, please work on the first of the three debates from each of the years 1996, 2000, 2004, 2008, 2012, 2016. I'll call each individual response by a candidate to a question a "chunk". A chunk might just be a few words or might be multiple paragraphs. The result of all of this activity in parts (a)-(d) should be well-structured data object(s) containing the information about the debates and candidates.

   Given that in PS2, you already worked on downloading and processing HTML, I'm giving you the code (in the file *ps/ps3prob2.R*) to download the HTML and do some initial processing, so you can dive right into processing the actual debate text.

   (a) Convert the text so that for each debate, the spoken words are split up into individual chunks of text spoken by each speaker (including the moderator). If there are two chunks in a row spoken by a candidate, it's best to combine them into a single chunk. Make sure that any formatting and non-spoken text (e.g., the tags for 'Laughter' and 'Applause') is stripped out. There should be some sort of metadata or attributes so that you can easily extract only the chunks for one

candidate. For the Laughter and Applause tags, retain information about the number of times it occurred in the debate for each candidate. You may need to do some looping as you manipulate the text to get the chunks, but try to do as much as possible in a vectorized way. Please print out or plot the number of chunks for the candidates.

(b) Use regular expression processing to extract the sentences and individual words as character vectors, one element per sentence and one element per word.

(c) For each candidate, for each debate, count the number of words and characters and compute the average word length for each candidate. Store this information in an R data structure and make of the word length for the candidates. Comment briefly on the results.

(d) For each candidate, count the following words or word stems and store in an R data structure: I, we, America{,n}, democra{cy,tic}, republic, Democrat{,ic}, Republican, free{,dom}, war, God [not including God bless], God Bless, {Jesus, Christ, Christian}. Make a plot or two and comment briefly on the results.

(e) Please include unit tests for your various functions. For the sake of time, you can keep this to a test or two for each function.

(f) (Extra credit) We may give extra credit for particularly nice solutions.

Hint: Depending on how your process the text, you may end up with lists for which the name of a list element is very long. Syntax such as `names(myObj) <- NULL` may be helpful.

3. This problem asks you to design an object-oriented programming (OOP) approach to the debate text analysis of problem 2. You don't need to code anything up, but rather to decide what the fields and methods you would create for a class that represents a debate, with additional class(es) as needed to represent the spoken chunks and metadata on the chunks for the candidates. The methods should include methods that take input text and create the fields in the classes. You can think of this in the context of R6 classes, or in the context of classes in an object-oriented language like Python or C++. To be clear, you do **not** have to write any of the code for the methods nor even formal code for the class structure; the idea is to design the formats of the classes. Basically if you were to redesign your functional programming code from problem 2 to use OOP, how would you design it? As your reponse for each class, please provide a bulleted list of methods and bulleted list of fields and for each item briefly comment what the purpose is. Also note how each method uses other fields or methods.