Stat243: Problem Set 3, Due Friday Sept. 29

September 17, 2017

This covers the first part of Unit 4 and Unit 5 (including the R debugging tutorial).

It's due **on paper** and submitted via Github at the start of class, with the additional requirement that Problem 1 is due by the end of the day Friday September 22 as discussed below.

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. As usual, your solution should mix textual description of your solution, code, and example output. And your code should be commented.
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.
4. Use functions as much as possible, in particular for any repeated tasks. We will grade in part based on the modularity of your code and your use of functions.
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.

Problems

1. On Monday, September 25, section will consist of a discussion of good practices in reproducible research and computing, led by Andrew and Chris. In preparation, please do the following:
   (a) Read Unit 5 and the *R debugging* tutorial (Andrew will have covered much of the material in the debugging tutorial in section September 18).
   (b) Read one of these four items:
      i. Read the Preface, the Basic Reproducible Workflow Template chapter and Lessons Learned chapters of the new (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)

(c) Please write down a substantive comment or question you have that is raised by this material. Write your comment/question in a plain text file called *bestpractices.txt*, put it in the *ps3* directory of your Git repository, and push to Github as you do for your problem set solutions. **Please do this by the end of the day, Friday Sep. 22** as we'll be collating them over the weekend. In addition, include your comment/question here in your problem set solution.

(d) 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?

2. DRAMATIS PERSONAE: INSTRUCTOR, STUDENTS, GSI
ACT 1. SCENE 1
UC Berkeley
INSTRUCTOR. [To STUDENTS] I set before thee the task of processing the plays of Shakespeare, to mine the text as it were, in the manner of picking coal from rock.
STUDENTS. [With gnashing of teeth] Nay, nay, thou shall not set us to this mighty task!
...
ACT Post-Berkeley
STUDENTS. [To INSTRUCTOR] We thank thee - you have set us upon a golden path, where beautiful data lie open to us in every nook of this World Wide Web.

Translation:
Your job is to extract data from the complete works of William Shakespeare, available as plain text at **http://www.gutenberg.org/cache/epub/100/pg100.txt**.

The goal of Problem 2 is two-fold: first to give you practice with regular expressions and text manipulation and the second 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. Comment your code as needed so it is clear what each piece of code does. Think carefully about how to structure your objects to store the information about the plays. If you prefer you can use an object-oriented approach, thereby combining problems 2 and 3 of this problem set.

(a) Extract the plays into a character vector or a list, with one element for each play. Skip the information at the start of the file, the first piece (the sonnets), and the last piece (Lover's Complaint).

(b) Extract meta data about each play and extract the body of the play. The result should be in the form of an R object, with one element for each play. By meta data I mean the year of the play, the title, the number of acts, and the number of scenes.

(c) Extract the actual text spoken by the characters into your object. You should store each chunk of spoken text and store the speaker of each chunk. Discard stage directions, Dramatis personae

information, and scene information. Note that the stage directions come in many forms, with inconsistent formatting, so we're not expecting perfection here – you'll likely accidentally include a small number of stage directions as part of your spoken text.

(d) Now use the constructed data object to calculate summary statistics about each play. These should include the following:

i. The number of unique speakers.
ii. The number of spoken chunks.
iii. The number of sentences and words spoken and average number of words per chunk.
iv. The number of unique words.

When extracting words, make sure you remove punctuation such as commas and periods but not apostrophes.

(e) Plot some of your summary statistics as a function of time to see if there are trends in Shakespeare's plays over the course of his writing career. Also in your solution, please report the number of acts and scenes, number of unique speakers, and number of chunks for each play; we'll look at this to see if your processing made any large-scale errors.

(f) Extra credit: particularly clever or thorough treatments of the problem may earn extra credit, such as avoiding excluding any plays or sophisticated treatment of songs and other items that are hard to handle. If you'd like your solution to be considered for extra credit, describe what you've done that you think does a good job of handling tricky aspects of the text.

HINTS:

Note that for part (c), I relied in part on indentation to indicate the beginnings and continuations of spoken chunks. The fourth play (The Comedy of Errors) has different indentation than the others, so in my solution I excluded it. You are allowed to exclude it or alternatively to exclude a small number of other plays that make your processing particularly difficult.

In my processing, I did a bit of preprocessing in UNIX (including a lot of exploratory grepping just to get a sense for formatting) and also did some preprocessing on a single character string containing the whole file in R. This allowed me to clean up some of the inconsistent formatting (e.g., "ACT 1" vs. "Act I."). It also allowed me to insert some special symbols of my own that later allowed me to split the chunks of spoken text more easily. (In particular I inserted a special character just before the name of each speaker of a chunk.)

For one time steps such as stripping off the lines that don't contain plays, you are allowed to hard-code in appropriate constants, such as what lines to extract.

I haven't had time to investigate this in full but it appears the the regex ".*" does NOT catch newlines (\n) when used with *stringr* functions. I had to work around this by making use of the :space: character class. Extra credit for anyone who tracks down why this is the case for stringr functions and if there is a solution.

3. This problem asks you to design an object-oriented programming (OOP) approach to the Shakespeare analysis of problem 2. You don't need to code anything up, but rather to decide what the fields and methods would be for a class that represents a Shakespeare play. You can think of this in the context of S4 or reference classes, or in the context of classes in an object-oriented language like Python or C++.

(a) What are the fields (i.e., member data, slots, etc.) for the class? Describe the kind of variable of each field (e.g., it might be a named numeric vector, or an unnamed list). You might decide to have a field contain an object of yet another class that you should describe.

(b) What are the methods for the class? Some of the methods should relate to processing the text of the plays to produce the fields and other methods should relate to providing information to a user who wants to know something about a play or see the text of the play. Indicate very briefly what the method does, and any input arguments to each method, fields that are created or modified by the method, and any output that is produced.