Stat243: Problem Set 2, Due Friday Sep. 17

September 8, 2021

Comments:

- This covers material in Units 3 and 4.
- It's due at 10 am on September 17, **both submitted as a PDF to Gradescope as well as committed to your GitHub repository** as documented in the *howtos/submitting-electronically.txt* file.
- 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 *ps2.Rmd* or a LaTeX+knitr file named *ps2.Rtex*, with bash and R code chunks included in the file (or read in from a separate code file).
   Please see the *dynamic documents* tutorial or Lab 1 materials for more information on how to do this, including dealing with bash code chunks and line breaks.
2. Your PDF submission should be the PDF produced from your Rmd/Rtex. Your GitHub submission should include the Rtex/Rmd file, any 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. Note that using chunks of bash code in Rmd/Rtex/Rnw can sometimes be troublesome, particularly on Windows machine. Some things to try if you are having trouble: (a) set the terminal to be the Ubuntu subsystem if you are on Windows; (b) if using RStudio, you may only be able to run bash chunks if you have the notebook mode on; (c) using single versus double quotes in your Rmd/Rtex document may make a difference. If you can't produce a PDF that includes the bash chunk output, feel free to not run those chunks and just paste in the output you get manually. Please post on Piazza if you have trouble.
4. You will probably need to use *sed* in a basic way as we have used it so far in class and in the tutorial on bash. You should not need to use more advanced functionality nor should you need to use *awk*, but you may if you want to.
5. Your solution should not just be code - you should have text describing how you approached the problem and what the various steps were.

6. 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.

Problems

1. Add assertions and testing for your code from Problem 4 of PS1. You may use a modified version of your PS1 solution, perhaps because you found errors in what you did or wanted to make changes based on Chris' solutions or your discussions with other students.

   (a) Add formal assertions using the *assertthat* package. You should try to catch the various incorrect inputs a user could provide and anything else that could go wrong (e.g., what happens if one is not online?).

   (b) Use the *testthat* package to set up a small but thoughtful set of tests of your functions. In deciding on your tests, try to think about tricky cases that might cause problems.

2. This problem provides practice using shell scripting. We'll use United Nations Food and Agriculture Organization (FAO) data on agricultural production that we saw in class. If you go to http://data.un.org/Explorer.aspx?d=FAO and click on "Crops", you'll see a bunch of agricultural products with "View data" links. Click on "apricots" as an example and you'll see a "Download" button that allows you to download a CSV of the data. In class we inspected the HTTP requests that the site handles (using the Network information in the browser Developer Tools) and found out that you can download a file directly via a URL that passes a GET request in the following format:
   `http://data.un.org/Handlers/DownloadHandler.ashx?DataFilter=itemCode:526&DataMartId=FAO&Format=csv&c=2,3,4,5,6,7&s=countryName:asc,elementCode:asc,year:desc`
   That downloads the data for Item 526 (apricots). Note that you may need to put the http address inside double quotes when using a UNIX shell command to download it. Also make sure there are no carriage returns in the http address (I had to break the address above to fit on the page). You can see the *itemCode* for other products by hovering over "View Data" link for the relevant product.

   (a) Using the shell, download the data for apricots. Extract the data, excluding the metadata and the global total, into another file. Then subset the data to the year 2005. Based on the "area harvested" determine the ten regions/countries using the most land to produce apricots. Now automate your analysis and examine the top five regions/countries for 1965, 1975, 1985, 1995, and 2005. (Do not just copy and paste your code one time for each of the years!). Everything you do should be done using shell commands you save in your solution file. So you can't say "I downloaded the data from such-and-such website" or "I unzipped the file"; you need to give us the bash code that we could run to repeat what you did. This is partly for practice in writing shell code and partly to enforce the idea that your work should be reproducible and documented.

   Note: dealing with the comma separation in the file can be a problem as there are commas used inside character strings as well. You'll probably want to convert the file to be delimited by something other than a comma - you can do this from the command line or if you would like to do it via R, you are allowed to use Rscript from the command line to run a little bit of R code to do the conversion. (Or if you know Python, feel free to do it from the command line via Python.)

(b) Write a bash function that takes as input a single item code (e.g., 526 for apricots, 572 for avocados), downloads the data, and prints out to the screen (i.e., to *stdout*) the data stored in the CSV file, such that the information could be piped to another UNIX command. Your function should detect if the user provides the wrong number of arguments and return a useful error message. It should also give useful help information if the user invokes the function as: "myfun -h".

3. On Friday, September 17, Section will consist of a discussion of good practices in reproducible research and computing. In preparation, please do the following:

   (a) Read **one** of these five items (you can read more if you want of course!):

      i. Chapter 11 of Christensen et al. Transparent and Reproducible Social Science Research (chapter 11 is here in bCourses) (You can also see the entire book online via UC Library search: https://california.degruyter.com/view/title/568658?tab_body=toc) - this one is written from the perspective of social scientists.
      ii. Gentzkow and Shapiro (https://www.brown.edu/Research/Shapiro/pdfs/CodeAndData.pdf) – also written from the perspective of social scientists.
      iii. Wilson et.al. (https://arxiv.org/pdf/1210.0530v3.pdf)
      iv. Millman and Perez (https://github.com/berkeley-stat243/stat243-fall-2014/blob/master/section/millman-perez.pdf)
      v. The Preface, the Basic Reproducible Workflow Template chapter and Lessons Learned chapters of this Berkeley-produced book on reproducible research, found online via UC Library search: The Practice of Reproducible Research.

      When reading, please think about the following questions:

      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?

      As your answer to this problem, please write a paragraph or two where you discuss one or more of these questions. I'm not looking for more than 10-15 sentences, just evidence that you've read one of the items and considered it thoughtfully.

   (b) Please skim through the paper *ps/clm.pdf* (in the class GitHub repo), focusing on the Method section, but note that the idea is just to get the main idea of the analysis steps, not to understand the context fully or see all the details. Then look at the code the authors provide at https://github.com/andykrause/hhLocation and think about whether that code makes it easy to reproduce what they did in the paper. Based on the Unit 4 PDF and the item you read above in (a), make a short list of strengths and weaknesses of the reproducibility of the authors' materials and provide that as your solution to this problem. Your task in Section on Friday, September 17 will be to discuss with your group and in Section as a whole and come up with a consensus answer.
      Some things to think about when you look at their materials:

i. From the information above and the documentation provided, can you quickly identify where in the code (if present at all) the authors:
   A. collected their data,
   B. cleaned/processed their data,
   C. calculated various statistics, and
   D. produced the plots in their paper.

ii. In terms of coding what elements of their project do you like? Consider: Documentation, comments, organization, naming, workflow, data provenance, etc.

iii. Similarly, what do you think could be improved?

iv. Without examining the code itself, can you quickly discern the purpose of each file?

v. Without examining the code in detail, can you quickly tell what each block of code does?

vi. Are there exceptions to your answers above?

vii. In what way do the authors document their workflow? Do you think this method is effective?