Problem Set 2

Due Friday Sep. 19, 10 am

Comments

- This covers material in Units 3 and 4.
- It's due at 10 am (Pacific) on September 19, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 for formatting and attribution requirements.
- Note that using chunks of bash code in Qmd may be troublesome.
  - You will need to add `engine: knitr` to the YAML preface of your qmd document (as seen in `unit2-dataTech.qmd` in the class repository). The default `jupyter` engine won't run both bash and Python chunks in the same document because Jupyter notebooks are associated with a single 'kernel' (i.e., a single language for the code chunks).
  - For the `knitr` engine, you'll need to have R installed on your computer, including the `knitr` package. Quarto will then process the code chunks through `knitr` (which will use R's `reticulate` package to handle Python chunks).
  - If you have trouble on your own computer, you can always render your solution for this problem set on an SCF machine (we won't generally use bash chunks in future problem sets). (In particular I'm not quite sure what will happen if you render on Windows.)
  - You can control which installed Python is used by the `knitr` engine using the syntax shown in the first code chunk of `unit2-dataTech.qmd`. This will work on the SCF as well as on your laptop (for the latter, you'll of course need to point to a `python` (or `python3`) on your machine.
  - We can help troubleshoot and feel free to post on Ed.
  - **Test things out well before the due date with a dummy qmd file with both a bash chunk and a Python chunk and make sure things work.**
- Using `sed` in a basic way as shown in the bash tutorial might be useful. You should not need to use more advanced functionality nor should you need to use `awk`, but you may if you want to.

Problems

1. A friend of mine is planning to get married in Death Valley National Park in March (this problem is based on real events…). She wants to hold it as late in March as possible but without having a high chance of a very hot day. This problem will automate the task of generating information about what day of March to hold the wedding using data from the Global Historical Climatology Network. All of your operations should be done using the bash shell except part (c). Also, ALL of your work should be done using shell commands that 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 provide the bash code that someone else 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.

a. Download yearly climate data from the location above for a set of years of interest into a temporary directory. Do not download all the years and feel free to focus on a small number of years to reduce the amount of data you need to download. Note that data for Death Valley is only present in the last few decades. As you are processing the files, report the number of observations in each year by printing the information to the screen (i.e., `stdout`), including if there are no observations for that year. Try to avoid printing anything else out (e.g., the progress of the downloading).

b. Subset to the station corresponding to Death Valley, to the TMAX (maximum daily temperature) variable, and to March, and put all the data into a single file. In subsetting to Death Valley, get the information programmatically from the `ghcnd-stations.txt` file one level up on the website. Do NOT type in the station ID code when you retrieve the Death Valley data from the yearly files.

c. Create a Python chunk (or R would be fine too) that takes as input your single file from (b) and makes a single plot showing side-by-side boxplots containing the maximum daily temperatures on each calendar day in March. (If you somehow really have trouble mixing Python and bash chunks, it's ok to insert this figure manually, after running the Python code separately. In this case you could use the `jupyter` engine, provided that a bash kernel is available for Jupyter.)

d. Now generalize your code from parts (a) and (b). Write a shell function that takes as arguments (1) a string for identifying the location, (2) the weather variable of interest, and (3) the time period (i.e., the years of interest and the month of interest), and returns the results (and in this case don't print out the information about the number of rows). Your function should detect if the user provides the wrong number of arguments or a string that doesn't allow one to identify a single weather station and return a useful error message. It should also give useful help information if the user invokes the function as: `get_weather -h`. Finally the function should remove the raw downloaded data files (alternatively, you should download into your operating system's temporary file location).

   Hint: to check for equality in a bash `if` statement, you generally need syntax like the following. The quotations ensure that strings with spaces or characters that the shell might otherwise try to interpret as shell syntax are treated as text.

```
if [ "${var}" == "some string" ]
if [ "${var}" != "some string" ]
```

2. Add documentation, error-trapping (i.e., "exception handling") and testing for your module from Problem 4d 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 (to be distributed in class) or your discussions with other students. These topics will be covered in Lab 2 (Sep. 12) and are also discussed in Unit 4.

   - Add informative doc strings (with example(s)) to your user-facing functions and basic doc-

strings to your internal functions.

- Add exceptions for handling run-time errors. 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 the server refuses the request or if one is not online?). In some cases you will want to raise an error, but in others you may want to catch an error with `try-except` and return `None`. Think about how to have your code be robust to the possibility that the key-value pairs or structure of the json result for some of the commits might be different than what you expect. Consider outputting warnings if there is something unexpected to tell the user about, but it makes sense to continue execution.
- Use the `pytest` package to set up a thoughtful set of unit tests of your functions.