Problem Set 3

Due Friday Oct. 3, 10 am

Comments

- This covers material in Unit 5.
- It's due at 10 am (Pacific) on October 3, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 for formatting requirements.

Problems

1. Recall the regex problem on detecting any integer or real number, positive or negative.

   a. In asking for AI assistance for this problem, I got both of the following answers. What is the main difference between them in terms of what they would detect? Is one better than the other?

      - `-?[0-9]+(\.[0-9]+)?`
      - `-?(\d+(\.\d+)?|\.\d+)`

   b. Let's take the problem and make it more concrete. Suppose you are writing a CSV reader in Python. One sub-task is to determine if a sequence of characters (found between two commas) is a number. In addition to numbers like 0.35, -.35, 17.35,and +72, you should detect scientific notation of the form 1.6e-8, where the mantissa (1.6 here) is integer or real and the exponent is integer. You should also detect numbers like "12,345.72" (which could occur in a CSV file like this: `,"12,345.72",`. Everything else should not be detected. Write a Python function that takes in the value (i.e., the characters between the commas separating the fields and assuming any quotations have been stripped out) and returns either the detected string (e.g., `"-.35"` or `"12,345.72"`) or None. (Side note: if you prefer to consider European notation like 12.345,73 and 0,73 in this problem, you're welcome to do so. And it would be straightforward to extend the problem to handle both styles of notation, but you don't have to.)

   c. Include a test suite that covers a wide variety of cases. To save a bit of time, this does not need to involve full formal use of pytest; you can just check the cases informally with your code, such as looping over a list of test strings.

2. Let's investigate the structure of the `pandas` package to get some experience with the structure of a large Python package and with how `import` and the `__init__.py` file(s) are used. You'll

need to go into the Pandas source code (see Unit 5). Note that the main `__init__.py` and the `__init__.py` files in the subpackages/submodules are complicated, and I'm not expecting you to understand everything about them. Also note that the following cases involve functions, classes, and class methods. Be sure to be clear to say which of those it is and for the class methods cases, make sure you're clear on what class the method is part of and any class inheritance structure. Run `import pandas` and then consider the following questions:

a. Consider `pandas.core.config_init.is_terminal`. What namespace is it (or its class) in? What file/module is `is_terminal` in? Is it a function, class, or a class method (and if a class method what is the class)? Describe how it is imported by discussing the relevant statement(s) in the relevant `__init__.py` file(s).
b. Consider `pandas.read_csv`. Answer the same questions as for (a).
c. Consider `pandas.arrays.BooleanArray`. Answer the same questions as for (a).
d. Consider `pandas.DataFrame.to_csv`. Answer the same questions as for (a).

Hints: (1) `grep -R <pattern> <directory>` will search all files within a directory recursively. (2) As you work on this, you may want to be able to modify one or more of the `__init__.py` files to better understand what is happening (e.g., by commenting out a line of code or adding a print statement). A good way to do this is to create a Conda environment in which pandas is installed, so you isolate any changes you make, e.g., `conda create -n test_env python=3.13 pandas`. Then you can edit code files in the environment and when you start Python and import pandas, you should see the effects of your changes. Alternatively, you could use the debugger to set breakpoint(s) in an `__init__.py` file. (3) Or you might create your own small toy package to experiment and see how things work with nested `__init__.py` files and various ways to use `import`.

3. This problem explores doing string processing in Python using AI assistance. I frame the problem as leaning heavily into AI and assessing/improving/iterating/checking what the AI provides, with the main points being (1) to get experience with AI assistance and (2) to get more experience with writing good Python code, critically assessing what AI produces. However, if you feel that what you get from the AI assistance is not that helpful or if you prefer to simply practice writing Python code without AI assistance (or with more limited assistance at a syntax level or line-by-line level) you can do that instead. Also, I haven't previously assigned a question like this that leans so heavily into AI, so we'll see how it goes. Please ask questions or post thoughts on Ed as you have them (noting that some of you likely have more experience with AI-assisted coding than I do).

   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. With the recent political upheaval in the US, this pattern of a regular series of debates has become more irregular, breaking down entirely in 2024.) Your task is to process the information and produce data on the debates. Note that while I present the problem below as subparts, your solution does not need to be divided into subparts in the same way. For the purposes of this problem, please work on the the debates I've selected (see code below) for the years 2000, 2004, 2008, 2012, 2016, and 2020. (I've tried to select debates that cover domestic policy in whole or in part to control one source of variation, namely the topic

of the debate.) 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.

In order to focus on the string processing and given you've already gotten some experience with http requests, I'm giving you the code (in the file `ps/ps3prob3.py` in the class repository) to download the HTML and do some initial processing, so you can dive right into processing the actual debate text.

Use an AI coding assistant to help solve the problem (which I have divided into subtasks below). You might use Agent mode to try to have it solve the entire problem at once or you might ask it to help with each subpart one by one. You can iterate with it to modify what it provides if that seems helpful, or take what it provides and modify it directly yourself. You should work through the code it produces line by line. In your final submitted code, there should not be any AI-produced code that you don't understand **in detail**. This is both to avoid errors and as a way for you to learn from the AI-generated code.

You're welcome to use whatever AI tool you want, either a ChatBot or an AI-assisted IDE, but getting experience with an IDE seems to me to be a good thing to practice with. GitHub Copilot in VS Code is probably the easiest to set up and can make use of GitHub for Education to use GitHub Copilot Pro, providing more free queries and use of premium models. You're welcome to try different back-end models; I've heard that folks think Claude Sonnet and Claude Opus are particularly good for coding, but I suspect various ones will do a pretty good job.

In interacting with the AI, you'll want to provide the Python code in `ps3prob3.py` and some examples of the content (from running `get_content`) as context. The context could be provided as files to an AI-assisted IDE or as part of the prompt/conversation with a ChatBot.

a. Convert the text so that for each debate, the spoken text is 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, combine them into a single chunk (in my AI-generated code, it did this incorrectly…). Make sure that any formatting and non-spoken text (e.g., the tags for 'Laughter' and 'Applause') is stripped out. Report the number of chunks per speaker.

b. Extract the individual words. To do the extraction I suggest you use existing Python natural language processing packages that can "tokenize" text. Doing this with 100% accuracy may be hard (particularly for spoken text). Try to handle various issues, but don't try to be perfect.

c. For each candidate, for each debate, count the number of words and characters and compute the average word length for each candidate, as well as the number of chunks. Compare these between candidates and over time in some basic fashion.

d. Set up assertions to check that the code is behaving as you expect. You'll want to run these assertions with the code you get from the AI and as you iterate on the code. Including assertions with shorter test input text where it is easier to know the correct result may be helpful. For the sake of time, you don't need to set up formal testing.

e. Please provide the prompt you used to initially request AI assistance.

f. Briefly describe some errors (minor or major) made by the AI assistant.

g. Briefly describe some thing(s) you learned about Python from examining the AI-generated code.

h. Briefly report on your overall experience with the AI assistant. Did it produce code that worked well? How much did you need to modify it, either yourself or by iterating with the AI. Was there syntax that you didn't understand that you needed to investigate or replace with your own syntax? Overall does this style of coding seem more efficient than if you wrote all the code yourself (perhaps with Google searches or AI assistance at a line-by-line/syntax level)?

i. (Extra credit) For each candidate, count the following words or word stems (feel free to replace some or all of these with words or expressions of your choice) and store in a data structure: I, we, America{,n}, democra{cy,tic}, republic, Democrat{,ic}, Republican, free{,dom}, terror{,ism}, safe{,r,st,ty}, {Jesus, Christ, Christian}. Make a plot or two and comment briefly on the results.