STAT 29000 Project 3

Topics: python3, functions, importing, modules, packages

Motivation: Even though it is not necessary to understand every detail about importing, writing functions, modules, packages, etc., in order to do data-driven work in Python, grasping the basics can go a long way in preventing future frustration.

Context: The past two projects have been a crash course in solving problems using Python. We've been introduced to a variety of topics like lists and sets, as well as important scientific computing packages like `scipy`, `numpy`, and `pandas`. In this project we learn about how packages are structured and imported, and use these skills to help organize code written to solve tasks dealing with movie data.

Scope: Python's `import` keyword to import various components of a Python package. Basic Python package structure, and writing custom functions.

For the following questions, please copy and paste the following code at the top of your Jupyter notebook. We are simply importing useful Python modules that are required for this project. Be sure to replace "PURDUEALIAS" with your Purdue username. Note that if you get an error after running this chunk of code, you may need to click on Kernel->Restart.

```
# the following two lines tell notebook.scholar.rcac.purdue.edu's default python interpreter
# that it should look in ~/.local/lib/python3.6/site-packages for packages as well
# if you do not include these two lines at the very top of your notebook, you won't
# be able to import and use the packages we installed at earlier times/dates
import sys
sys.path.append("/home/PURDUEALIAS/.local/lib/python3.6/site-packages")
```

You can find useful examples that walk you through relevant material here or on scholar: `/class/datamine/data/spring2020/s`
It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar: `/class/datamine/data/spring2020/stat29000project03template.ipynb` to submit your solutions.

The pandas documentation is very good and would be the first place you should go to read about pandas related keywords.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online.

Question 1: importing, modules, & packages

realpython.com does an excellent job quickly introducing you to Python's `module` and `package` concepts. Reading this overview carefully will help you solve the following problems.

In addition, here is a discussion on stackoverflow that may be useful.

1a. *(.5 pts)* You can download a zipped directory here or on scholar: `/class/datamine/data/spring2020/media.zip`. Unzip the folder, which should be named `media`, and place this directory in the same directory your Jupyter notebook is in. Run `import media` in your notebook to confirm you can access the package as expected. What happens?

Keywords: *import*

1b. *(.5 pts)* Import and test out *only* the `search` function in the utilities module in the `rottentomatoes` sub-package. To test the function `search` out, you can select some (say, three to five) keywords, and search

for them using the function. For example if you were looking for the Harry Potter movies, you could try using the words "harry" and "potter".

In order to test `search`, the following code should work: `search("harry", "potter")`. Note that if you run `?search` in your Notebook (look towards the bottom of your screen after running it), you will see the argument in the signature of the `search` function has an asterisk in front of it `*words`. This is called *packing* and *unpacking* tuples/lists. In question (2c), you will need to figure out how to use it in your function, but for now, just test it by running `search("harry", "potter")`.

Hint: You can open up the different modules (.py files) in the package, and read about functions to understand how they work.

Hint: Another option is to use `print(<function>.__doc__)`. This will print any associated docstring for a function.

Note: *In the provided example, `media` is the package, `rottentomatoes` is a sub-package, and `utilities.py` is a module of the `rottentomatoes` sub-package. Any .py files directly in the `media` folder would be modules of the `media` package.*

Keywords: *nested packages, import*

1c. *(.5 pts)* Repeat (1b), but this time import all of the functions provided in the utilities module in the `rottentomatoes` sub-package. Once again, test the `search` function. In order to test the `search` function, the following code should work: `utilities.search("harry")`. Feel free to test this out using other words.

Keywords: *import sub-module*

1d. *(.5 pts)* Repeat (1c), but this time instead of typing out "utilities.<function>" each time, we want to shorten this to "utils.<function>". So to test the `search` function, the following should work: `utils.search("harry", "potter")`. Feel free to test this out using other words.

Keywords: *import as*

Question 2: functions

You work for a company that among other things, analyzes movie and movie review data. They have an internal Python package called `media`. They have a continually updated database containing movie and review data, and sample data into `pandas` dataframes for analysis. Two such dataframes come pre-loaded in the package. You've now been tasked with writing some functions to be added to the package for the company.

Here is a great primer on writing functions in Python3. Note that you'll probably still need to do some Google research on some of the keywords in order to solve some problems.

Important note: Inside the `media` package, functions reference the internal datasets as `ds.movies` and `ds.reviews`. We want to do the same, so make sure to first import the `datasets` module from the `rottentomatoes` sub-package and use the trick we learned in (1d), to refer to it as `ds`. For every function you write, when accessing the movies dataset inside that function, use `ds.movies` and/or `ds.reviews`. You can test that you've imported things properly by running something like `ds.movies.head()`, after all, `ds.reviews` and `ds.movies` are just `pandas` dataframes. You can verify that they are `pandas` dataframes by running `type(ds.movies)`.

2a. *(1 pt)* Write a function called `get_cast` that accepts a rotten tomatoes id as an argument. Consider the values in the `rotten_tomatoes_link` column to be unique movie ids (i.e., each movie has a unique id in the `rotten_tomatoes_link` column). Your function should try to find the provided rotten tomatoes id in the `ds.movies` dataset. If the id was found in the dataset, it should return a tuple containing the names of the cast members (for example, `("Matt LeBlanc", "Lisa Kudrow", "Jennifer Anniston", )`. If the id wasn't found, return an empty tuple (both `()` and `tuple()` are empty tuples, you can test by `type(())` or `type(tuple())`).

Hint: *Be careful here. Run the code below. It could help you avoid making a mistake returning the cast in the wrong type.*

```
first = (a for a in list('xyz'))
print(type(first))

second = tuple(a for a in list('xyz'))
print(type(second))
```

Keywords: *function, function argument, return keyword, return multiple values*

2b. *(1 pt)* Modify (2a) to accept multiple id's in the form of a tuple or list, and return a tuple containing every actor in all the films identified by the provided id's. For the sake of clarity, the following would be a valid argument for our new function: `my_tuple = ("/m/rush_hour", "/m/rush_hour_2", "/m/rush_hour_3",)`, and then you could do: `get_cast(my_tuple)`.

Keywords: *pandas isin, loc, pandas empty, string methods: split, strip, join*

2c. *(2 pt)* You brought your solution (2b) to a coworker who tells you that most of the time users will just pass a single id, but sometimes will pass multiple ids and it would be better if `get_cast` accepted n arguments where each argument is a single id. Modify (2b) to accomodate this.

Show how to use `get_cast` to get the cast for two movie ids.

Given a tuple `my_tuple = ("/m/rush_hour", "/m/rush_hour_2", "/m/rush_hour_3",)`, use `get_cast` to get the cast for every movie in `my_tuple`, without looping (see the hint below).

Show how to use `get_cast` to save the first item in your resulting tuple in the variable `first`, the second item in your resulting tuple in the variable `second`, and the rest of the cast members in a variable called `rest`, all in 1 line of code.

Hint: *Don't forget about packing/unpacking tuples.*

Keywords: *packing/unpacking arguments, packing/unpacking tuples*

2d. (optional) *(0 pts)* Write a function called `cast_in_common` that accepts n arguments where each argument is a single movie id, and returns a tuple containing every actor all movies have in common.

Hint: The function you built in (2c) may be useful here.

Keywords: *set intersection method*

Question 3: putting it all together

3a. *(2 pts)* Write a function called `get_reviews` that accepts a single movie id as an argument and returns a tuple where every element in the tuple is a string containing the `review_content` for each associated review.

Keywords: *tuple, to_list*

3b. *(2 pts)* Write a function called `get_polarizing` that returns a dataframe containing polarizing movies. Let's define polarizing movies as movies where the `tomatometer_count` is > 20 and the `tomatometer_rating` and `audience_rating` differ by $>= 20$.

Hint: If you get an error when trying to filter a dataframe on multiple criteria, Google the error.

Keywords: *abs, & in pandas*

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/xG2GTE3I using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .ipynb and .pdf formats. We've updated our instructions to include multiple ways to convert your .ipynb file to a .pdf on scholar. You can find a copy of the instructions on scholar as well: `/class/datamine/data/spring2020/jupyter.pdf`. If for some reason the script does not work, just submit the .ipynb.