STAT 29000 Project 5

Topics: python3, scripts, arguments

Motivation: It is common in industry and academics to repeat a process over and over. This could include running a report, performing an analysis, or updating a source of data. Writing a script is one way to automate tedious tasks. In this project we will learn to write a python3 script.

Context: The previous project we implemented a variety of new functions for our `media` package. In this project we will focus on learning about writing and using python scripts.

Scope: Python scripts, arguments, python basics, etc.

First, we must install missing packages `sklearn` and `stop-words`. In scholar, open up a shell and type the following:

```
python3.6 -m pip install sklearn --user
python3.6 -m pip install stop-words --user
```

For the following questions, *and* for the scripts you write as a part of this project, please copy and paste the following code at the top of your Jupyter notebook, and at the top of your scripts. 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.

```python
# 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`
It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar: `/class/datamine/data/spring2020/stat29000project05template.ipynb` to submit your solutions.

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. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Question 1: writing a script

Often times the deliverable part of a project isn't custom built packages or modules, but a script. A script is a .py file with python code written inside to perform action(s). Python scripts are incredibly easy to run, for example, if you had a python script called myscript.py, you could run it by opening a terminal and typing:

```
python3.6 /path/to/myscript.py
```

The python interpreter then looks for the scripts entrypoint, and starts executing. You should read this article about the `main` function and python scripts.

In order to keep the environment uniform among students, we will be working on scholar.

1a. *(1 pt)* For this question we are going to setup a workspace in scholar and submit the commands we use to do so as the solution to this problem. Make sure to write down the commands used as you use them. For example, if my task was to navigate to `~/folder1/folder2`, then ensure I was in the correct folder, then list the files in the folder, then remove `myfile.txt`, I would include the following solution in my notebook:

```
cd ~/folder1/folder2
pwd
ls -la
rm myfile.txt
```

Log in to scholar and open up the terminal emulator by clicking on `Applications > Terminal Emulator`. In the terminal, navigate to `$HOME` (command 1), and create a new directory called `project05workspace` (command 2), and navigate into it (command 3). Copy the python script from `/class/datamine/data/spring2020/p05question01.py` or download from here to the `project05workspace` directory as `question01.py` (if you choose to copy instead of download, command 4). Write the commands used in the terminal, in the order in which you used them for your solution.

Hint: *Remember the `man` command. Simply type `man <command>` and it will show the manual pages for .*

Keywords: *cd, ls, pwd, mkdir*

1b. *(1 pt)* In your terminal (which should still be inside the `project05workspace` folder), and run the script. Copy and paste the output into a markdown cell for this answer.

Keywords: *running python scripts*

1c. *(1 pt)* Open up your favorite text editor on scholar, navigate to `project05workspace`, and open up `question01.py`. Modify the provided script to accept one argument called `friend`. When the script is provided with `friend`, it will display a modified message that greets `friend` (assume `friend` is the name of the person being greeted). Feel free to further personalize the greeting as you'd like.

To run a python script and provide it with argument(s), do the following:

```
python3.6 /path/to/myscript.py arg1 arg2 arg3
```

Hint: *This article provided in the examples is probably the one you want to look at.*

Keywords: *sys.argv, f-strings*

1d. *(.5 pts)* You'll notice if you try to run your script created in (1c) but forget to provide it with an argument, you'll (most likely) get a `IndexError`. What this means is there is no element at the index you specified (because you didn't give the script an argument). Let's modify your `question01.py` script to keep the default behavior (the result of (1b)) if the `friend` argument is not provided to the script.

Hint: *sys.argv is a list, think about what the length of the list tells us in this scenario.*

Keywords: *len, if statement*

Question 2: argparse

For the following few problems, please read the argparse documentation. You will find it invaluable to solve these problems.

2a. *(1 pt)* Although `sys.argv` can be useful for a quick experimental scripts, argparse is "the recommended command-line parsing module in the Python standard library". Let's take a look. First copy and paste your `question01.py` script to a new script called `question02.py`. Don't make any further modifications to your `question01.py` script, you are on question 2 now. Open up your favorite text editor on scholar, navigate to `project05workspace`, and open up `question02.py`.

Replace your code relating to `sys.argv` with argparse. Specifically, `import argparse`, create your parser (inside `main`), add a *positional argument* named `friend` to your parser, add useful *help* text, save the result of `parse_args()` into a variable, and use that variable to update the text your script prints.

Copy and paste the output when you run: `python3.6 ~/project05workspace/question02.py buddy`, into a cell.

Copy and paste the output when you run: `python3.6 ~/project05workspace/question02.py`, into the same cell below the result from above.

2b. *(1 pts)* As you can see from (2a), the functionality doesn't yet completely match (1d). Modify `question02.py` to add our default value from (1d) as our default value here. Note that in order for a *positional argument* to also be *optional*, you must set `nargs="?"` in `add_argument`.

Keywords: *nargs, default, add_argument*

2c. *(1 pt)* I've provided you with `squeak`, a function here $^{\textit{to make mickey squeak}}$ (that is, to show the friend's name in the small text – unicode superscript text). Write a function TO MAKE MICKEY YELL, and add both the `squeak` and `yell` functions to `question02.py`. Read the Getting a little more advanced section of the `argparse` documentation. Add an *optional argument* named `volume`, and be sure to provide help text. Mickey has three volumes: (volume 1), normal (volume 2), and YELL (volume 3). By default Mickey squeaks (volume 1). Make sure the following commands work properly:

```
python3.6 ~/project05workspace/question02.py "Minnie Mouse" -v
```

```
python3.6 ~/project05workspace/question02.py "Minnie Mouse"
```

```
python3.6 ~/project05workspace/question02.py "Donald Duck" -vv
```

```
python3.6 ~/project05workspace/question02.py "Pluto" -vvv
```

```
python3.6 ~/project05workspace/question02.py "Pluto" --volume --volume --volume
```

```
python3.6 ~/project05workspace/question02.py -vvv
```

Hint: *Make sure you define or place your* `squeak` *and* `yell` *functions above: (otherwise you'll get a NameError when trying to call them)*

```python
if __name__ == "__main__":
    main()
```

Question 3: media script

(3.5 pts)

We've provided you with the most up-to-date version of the `media` package from previous projects. You can find the zipped directory here or on scholar: `/class/datamine/data/spring2020/p05media.zip`. Extract the `p05media.zip` into `~/project05workspace/`.

Copy the python script from `/class/datamine/data/spring2020/p05question03.py` or download from here to the `project05workspace` directory as `question03.py`. Modify the script to accept one required *positional* argument called `rt_id`, and one *optional* argument called `count`. The *optional* `count` argument should have a default value of 50. The script does the following:

1. Gets the arguments.
2. We've provided you with a modified function called `get_reviews` in the `media` package. Given 1. a rotten tomatoes id, and 2. an integer (let's say n) indicating the number of reviews to "get", as arguments,`get_reviews` now returns a list of n reviews as well as a list of n classifications for each corresponding review. Use `get_reviews` to save the list of reviews into a variable called `corpus`, and the list of classifications into a variable called `classifications`. See here for an important note on `get_reviews`.

3. Use the `corpus_terms` function (from the `utilities.py` module in the `rottentomatoes` sub-package) on the `corpus` variable from (2) to get your terms. Save those terms in a variable called `terms`.
4. Use the `tfidf` function and save the result into a variable called `tfidf_scores`.

Now you have finished a script finding the best words to use to classify the rotten and fresh reviews!

The following commands should work:

`~/project05workspace/question03.py` (this should trigger an error saying `rt_id` is required)

`~/project05workspace/question03.py frozen_2013`

```
# words may not match, but the numbers should
Great words to use as features for a review classifier:
1. tribute (49.484)
2. center (46.238)
3. technique (45.208)
4. aladdin (45.208)
5. im (45.208)
6. favor (45.208)
7. lifeless (45.208)
8. hand (45.208)
9. saying (45.208)
10. desperate (45.208)
```

`~/project05workspace/question03.py frozen_2013 -c 5`

```
# words may not match, but the numbers should
Great words to use as features for a review classifier:
1. tribute (49.484)
2. center (46.238)
3. whips (45.208)
4. efforts (45.208)
5. hand (45.208)
6. technique (45.208)
7. cut (45.208)
8. staying (45.208)
9. lacks (45.208)
10. glory (45.208)
```

`~/project05workspace/question03.py frozen_2013 --count 5`

Hint: *Feel free to work through individual steps of this problem in a notebook. If you decide to do this, make sure the `media` folder is in the same directory as your practice notebook.*

Important note on `get_reviews`: *If there are not `n` number of reviews available, `get_reviews` will do the following (this is expected behavior, don't worry):*

```python
from media.rottentomatoes.reviews import get_reviews
get_reviews('10', 10)
get_reviews('10', 100)
# SystemExit: There are only 13 reviews for the movie with rotten tomatoes id 10.
```

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/X_9Dx2Qe 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. Please note that for this project you will be submitting a total of 5 files: 1 .ipynb, 1 .pdf, & 3 .py.