STAT 29000 Project 6

Topics: python3, scraping data

Motivation: Being able to systematically parse through and download data from the internet is an integral tool to add to your toolset. It allows you to build, augment, and compare your datasets.

Context: We've been using python to solve data driven problems, one class of data driven problems is finding, downloading, and parsing said data. In this project we will use the `requests` and `beautifulsoup4` packages to scrape imdb data for our `media` package.

Scope: Scraping data in python3 using `requests` and `beautfulsoup4`.

First, we must install the most up-to-date packages `requests` and `beautifulsoup4`. In scholar, open up a shell and type the following:

```
python3.6 -m pip install requests --user
python3.6 -m pip install beautifulsoup4 --user
```

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/stat29000project06examples.ipynb`.

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

Use the template found here or on scholar:

`/class/datamine/data/spring2020/stat29000project06template.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: top 250

At some point or another most of you have probably wandered onto IMDB's top 250 list: https://www.imdb.com/chart/top/?ref_=nv_mv_250

For this question, we want the end result to be a function that returns a pandas dataframe with the top 250 movies in one column called `movie`, the years the movies came out in another column called `year`, and the imdb star rating in a column called `stars`. We will guide you through this in this series.

1a. *(.5 pts)* Open a browser (Firefox or Chrome for this project) and navigate to https://www.imdb.com/chart/top/?ref_=nv_mv_250. Right click on the column header "Rank & Title", and inspect the element. You will get a view of the raw HTML code. If you take your mouse and hover over sections of the code, the browser will highlight the part of the website that is contained within the tags.

From the `<th>Rank & Title</th>` part that should be currently highlighted, navigate up until the table of movies is highlighted. Write out the the start tag (including attributes) and end tag as your solution. For example for `<div class="header" id="first"></div>`, `<div class="header" id="first">` is the start tag, `</div>` is the end tag, and "class" and "id" are attributes. In Jupyter notebooks you can make a code block render HTML by writing `%%html`, alone, in the first line. This will prevent the notebook from having a syntax error when run. See here, and there is also a simple example in the examples as well.

Hint: *The tag will be a "table".*

Keywords: *inspect, html tag, html attribute*

Item(s) to submit:
- Start and end tag (including attributes) in a Code cell with the `%%html` magic command at the top.

1b. *(.5 pts)* Just as was shown in the examples, use `requests` to download the page's HTML. Create "soup" (feed the HTML to our beautiful soup package to create a parser) use the attributes from (1a) and the `find_all` method to extract just the HTML contents of the table.

Hint: *If you get a SyntaxError, remember you can pass attributes to `find_all` using a dict like: `find_all(attrs={"class":"value1", "attribute2":"value2"})`.*

Keywords: *requests: get, text method; beautifulsoup4: BeautifulSoup, find_all;*

Item(s) to submit:
- The code used for the problem.
- Printed output of the first 800 characters, starting with the start tag found in (1a).

1c. *(2 pts)* Take a careful look at the HTML you get from (1b) (the `prettify` method combined with `print` can help here, for example, `print(soup.prettify())`). Remember our end goal is to get the movie name, year, and star rating. There should be two "td" tags that, together, contain all of the information we want, for each movie.

Use the `find_all` function in combination with the "class" attributes from each "td" tag to parse our data into variables: `movie_and_year_soup`, and `stars_soup`.

Look at a single element of `stars_soup` via: `stars_soup[0]`. Think about how you would extract the stars rating using this and this (<– read the "Here are some simple ways to navigate that data structure:" section). Loop through every element in `stars_soup` and append the values to a list called `stars`.

Using the exact same principles, loop through every element in `movie_and_year_soup` and save all of the titles to a list called `movie`, and save all of the years to a list called `year`. Make sure to remove the surrounding parenthesis of each `year` value.

Keywords: *find_all, string, replace string method*

Item(s) to submit:
- The code used for the problem.
- The first 10 values of the `stars` list, printed.
- The first 10 values of the `movie` list, printed.
- The first 10 values of the `year` list, printed.

1d. *(1 pt)* Put (1a), (1b), and (1c) together to create a function called `top250` that returns a pandas dataframe with the top 250 movies in one column called `movie`, the years the movies came out in another column called `year`, and the imdb star rating in a column called `stars`.

Make sure to use the function `int` to make sure every year in your year column is of type `int`, and the function `float` to make sure every star rating is of type `float`.

Item(s) to submit:

- The code used for the problem.
- The `.head()` of the result of a call to the `top250` function, printed.

1e (optional). *(0 pts)* Get the average star rating for movies from the top 250 list before (non-inclusive) 2000 and after (inclusive) 2000.

Keywords: *groupby, mean*

Question 2: following "crawling" a link

In this question, we will continue to practice what we learned in (1), but this time, we will be required to use what we scrape on one page, to automatically lead us to another (which we also scrape).

The end goal of this question is:

1. To produce a function that takes a imdb id, and returns the metacritic score for the movie.
2. To update our `top250` function to extract imdb id's from the top 250 movies page, and use our newly created function to crawl to each of the individual movie pages to extract the metacritic score. We will then add it as a column called `metascore` to our data frame.

2a. *(2 pts)* We will start by opening a browser and exploring The Lion King: https://www.imdb.com/title/tt0110357. Scrape the page using `get` from `requests`, and feed the result to `BeautifulSoup` to create a parser. Use the parser to extract the metascore.

Hint: *Start by inspecting the little green box containing the metascore.*

Item(s) to submit:

- The code used for the problem.
- Printed output of the metascore.

2b. *(2 pt)* Create a function called `get_metascore` that accepts an imdb id as an argument, and returns the metascore (as an integer). There are some movies that don't have a metascore. For those movies, make sure to check the length of the result of your `find_all` function. If the length == 0, return `None`. Test your function like below:

```
result = get_metascore('tt0110357')
print(result)
# 88
print(type(result))
# int
print(get_metascore('tt0095327'))
# None
```

Hint: *You will need to concatenate the id's to a "base url" prior to scraping using `requests`. f-strings are perfect for doing this.*

Keywords: *int, find_all, f-strings*

Item(s) to submit:

- The code used for the problem.
- The output from running the three print statements above.

2c. *(2 pt)* Copy and paste your `top250` function into a new cell in your notebook. Update this function to return the same dataframe, with an additional column called `metascore`, containing the metascore for each movie.

Hint: *You've already built a function to scrape the metascore given an imdb movie id. The first step should be to find all of the movie id's and use your new function to scrape your metascores.*

Hint: You may find that under the `<td class="ratingColumn">` tag the movie ids are stored in the attribute `data-titleid` within the `div` tag. However, directly searching for "ratingColumn" would return two types of `td` tags: "ratingColumn" and "ratingColumn imdbRating". The code below helps us to avoid the second type of `td` and find the movie ids:

```
html = requests.get("https://www.imdb.com/chart/top/?ref_=nv_mv_250")
soup = bsoup(html.text)
id_soup = soup.find_all(class_="ratingColumn")
ids = [i.find_all("div", attrs={"data-titleid": True}) for i in id_soup]
ids = [i[0]['data-titleid'] for i in ids if len(i) > 0]
```

Item(s) to submit:

- The code used for the problem.
- The `.head()` of the result of a call to the `top250` function, printed.

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/VSar6te4 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.