Homework 5

Due date: Jun 2, 2024

Submission instructions:

- **Autograder will be used for scoring, you are required to write a module in a file `hw5module.py` as in Homeworks 1 and 2.**
- You are also required to convert this notebook as a separate Python file, **`hw5.py`**.
- Also, please keep your indentifiable information (ID, name, and a list of collaborators) in a separate file **`hw5_studentinfo.txt`**.
- Submit **`hw5.ipynb`** (this notebook), `hw5.py`, `hw5module.py`, and `hw5_studentinfo.txt` on Gradescope under the window "Homework 5 - code". Do **NOT** change the file name. This will be checked by the autograder as well.
- Make sure all your code and text outputs in the problems are visible in your PDF submission.

Introduction

What's your favorite movie? Wouldn't it be nice to find more shows that you might like to watch, based on ones you know you like? Tools that address questions like this are often called "recommender systems." Powerful, scalable recommender systems are behind many modern entertainment and streaming services, such as Netflix and Spotify. While most recommender systems these days involve machine learning, there are also ways to make recommendations that don't require such complex tools.

In this homework, you'll use webscraping to answer the following question:

> What movie or TV shows share actors with your favorite movie?

The idea of this question is that, if the movie Y has many of the same actors as the movie X, and you like X, you might also enjoy Y.

This homework has two parts. In the first, larger part, you'll write a webscraper for finding shared actors on TMDB. In the second, smaller part, you'll use the results from your scraper to make recommendations.

You need to meet the specifications for a complete list of what you need to do to obtain full credit.

Instructions

1. Setup

1.1. Locate the Starting TMDB Page

Pick your favorite movie, and locate its TMDB page by searching on https://www.themoviedb.org/. For example, my favorite movie is *Harry Potter and the ~~Sorcerer's~~Philosopher's Stone*. Its TMDB page is at:

```
https://www.themoviedb.org/movie/671-harry-potter-and-the-philosopher-s-stone/
```

Save this URL for a moment.

1.2. Dry-Run Navigation

Now, we're just going to practice clicking through the navigation steps that our scraper will take.

First, click on the *Full Cast & Crew* link. This will take you to a page with URL of the form

```
<original_url>cast/
```

Next, scroll until you see the *Cast* section. Click on the portrait of one of the actors. This will take you to a page with a different-looking URL. For example, the URL for Alan Rickman, who played Severus Snape, is

```
https://www.themoviedb.org/person/4566-alan-rickman
```

Finally, scroll down until you see the actor's *Acting* section. Note the titles of a few movies and TV shows in this section.

Our scraper is going to replicate this process. Starting with your favorite movie, it's going to look at all the actors in that movie, and then log all the *other* movies or TV shows that they worked on.

At this point, it would be a good idea for you to use the Developer Tools on your browser to inspect individual HTML elements and look for patterns among the names you are looking for.

1.3. Create your module

No template is provided for this homework. You will write your two functions in a separate file `hw5module.py`.

1.4. Some hints

You may run into `403` (forbidden) errors once the website detects that you're a bot. See the web scraping lecture note and these links (link1, link2, link3, link4) for how to

work around that issue. Adding a delay for each page and changing user agent will often be most helpful!

Keep an eye out for `403` error you see! Make sure to examine the `status_code` attribute of the returned value from `requests.get()`. You want your status to be `200` (meaning OK). Print something if you see `403` (or raise an `Exception` if you are familiar with it). If they know that you are on Python or if you are requesting pages without much delays, they will certainly try to block you. One way to change user agent on your code is presented in the lecture note. For the autograder to finish in reasonable time, please do not put the delays longer than two seconds between requests.

2. Write Your Scraper

Now, you will write a web scraper for a movie of your choice by giving its subdirectory on TMDB website as an argument. We will implement two parsing functions.

- `parse_full_credits(movie_directory)` should assume that you start on the *Full Cast & Crew* page with the url `https://www.themoviedb.org/movie/<movie_directory>/cast`. Its purpose is to call the function `parse_actor_page(df, actor_directory)` for the page of each actor listed on the page. Crew members are not included (consider using `not` command in CSS selector). Initialize an empty `DataFrame` with two columns `actor` and `movie_or_TV_name`, then call the function `parse_actor_page` for each actor. The `parse_full_credits()` function returns the fully loaded `df`, with actor names and movie titles each actor worked on. The `DataFrame` should not have duplicate entries, and it should be sorted by actor name as the primary key, then movie titles. Try to avoid visiting the same page multiple times.
  - Example: `df = parse_full_credits("671-harry-potter-and-the-philosopher-s-stone")`
- `parse_actor_page(df, actor_directory)` should assume that you start on the page of an actor. For each movie with the "Acting" role, you will add a row to the `DataFrame df` with two columns, `actor` and `movie_or_TV_name`. Please only include the works listed in "Acting" section of the actor page. Keep in mind that "Acting" might not be on the top of their lists; for example, David Holmes is credited with an acting role in HP1, but spent most of his career as a stunt double of Daniel Radcliffe (as a part of Crew). On his page, you will see "Crew" before "Acting". Note that you will need to determine both the name of the actor and the name of each movie or TV show through parsing the HTML page. It should return the `DataFrame df` with all the works of the actor added at the end of `df`.
  - Example: `df_updated = parse_actor_page(df, "10980-daniel-radcliffe")`

Provided that these functions are correctly implemented, you can run the code

```
df = parse_full_credits("671-harry-potter-and-the-philosopher-s-stone")
```

to create a `DataFrame` with a column for actors and another for movies or TV shows for *Harry Potter and the Philosopher's Stone*. You might want to save the result as a `.csv` file before proceeding to the next part.

Test your functions; make sure to check the following:

- `parse_actor_page()`
  - only parses all the works under the "Acting" section
  - even if "Acting" is not on the top of the lists
  - remove duplicate work names within each actor (added 5/23)
- `parse_full_credits()`
  - is parsing all the actors,
  - is not parsing crew members,
  - does not parse duplicate pages, and
  - of course, if the results are correct.

Challenge

If you're looking for a challenge, think about ways that may make your recommendations more accurate. Consider scraping the number of episodes as well or limiting the number of actors you get per show to make sure you only get the main series cast. If you do so, please use separate function names.

3. Make Your Recommendations

Once you're happy with the operation of your webscraper, compute a sorted list with the top movies and TV shows that share actors with your favorite movie. For example, it may have two columns: one for "movie names" and "number of shared actors".

Feel free to be creative. You can show a pandas data frame, a chart using `matplotlib` or `plotly`, or any other sensible display of the results.

4. Documentation

In this Jupyter Notebook, you should describe how your scraper works, as well as the results of your analysis. When describing your scraper, I recommend dividing it up into the two distinct parsing function, and discussing them one-by-one. For example:

> *In this report, I'm going to make a super cool web scraper... Here's how we set up the project...*
>
> `<implementation of parse()>`
> *This function works by...*

> `<implementation of parse_full_credits()>`

> *To write this function, I...*

In addition to describing your scraper, your report should include a table and visualization of numbers of shared actors.

You should guide your reader through the process of setting up and running the scraper.

Specifications

Coding Problem

1. Each of the two parsing methods are correctly implemented.
2. A table or list of results or pandas dataframe is shown.
3. A visualization with `matplotlib`, `plotly`, or `seaborn` is shown.

Style and Documentation

4. Each of the two `parse` functions has a short docstring describing its assumptions (e.g. what kind of page it is meant to parse) and its effect, including navigation and data outputs.
5. Each of the two `parse` functions has helpful comments for understanding how each chunk of code operates.

Writing

6. The report is written in engaging and clear English. Grammar and spelling errors are acceptable within reason.
7. The report explains clearly how to set up the project, run the scraper, and access the results.
8. The report explains how each of the two `parse` methods works.

In [2]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time


def parse_full_credits(movie_directory):
    # Load movie's full credits page content
    url = f'https://www.themoviedb.org/movie/{movie_directory}/cast'
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    # Find the cast section
    cast_section = soup.find('ol', class_='people credits')
    actor_set = set()
    df = pd.DataFrame(columns=['actor', 'movie_or_TV_name'])
    for row in cast_section.select('div.info a'):
        actor_directory = row['href'].replace('/person/', '')
```

```
        if actor_directory not in actor_set:
            df = parse_actor_page(df, actor_directory)
            print(actor_directory)
            actor_set.add(actor_directory)
            time.sleep(0.5)
    # Sort the DataFrame by actor then movie_or_TV_name
    df_sorted = df.sort_values(by=['actor', 'movie_or_TV_name'])
    return df_sorted


def parse_actor_page(df, actor_directory):
    # Load actor's page content
    url = f'https://www.themoviedb.org/person/{actor_directory}'
    response = requests.get(url=url)
    if response.status_code != 200:
        return df
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find the actor's name
    actor_name = soup.find('head').find('title').get_text().split(' — ')[0]

    # Find the section with "Acting"
    acting_section = None
    for h3 in soup.select_one('section.credits').find_all('h3'):
        if 'Acting' in h3.get_text():
            acting_section = h3
            break
    if acting_section:
        # The next sibling of the "Acting" header is the table
        acting_table = acting_section.find_next_sibling('table')
        if acting_table:
            title_set = set()
            for row in acting_table.select('a.tooltip'):
                title = row.get_text(strip=True)
                if title not in title_set:
                    new_row = pd.DataFrame({'actor': [actor_name], 'movie_or
                    df = pd.concat([df, new_row], ignore_index=True)
                    title_set.add(title)
    return df


if __name__ == '__main__':
    # df = pd.DataFrame(columns=['actor', 'movie_or_TV_name'])
    # df = parse_actor_page(df, "2710-james-cameron")
    parse_full_credits('671-harry-potter-and-the-philosopher-s-stone')
```

```
10980-daniel-radcliffe
10989-rupert-grint
10990-emma-watson
194-richard-harris
10993-tom-felton
4566-alan-rickman
1923-robbie-coltrane
10978-maggie-smith
10983-richard-griffiths
10985-ian-hart
10981-fiona-shaw
5049-john-hurt
11180-david-bradley
96841-matthew-lewis
11179-sean-biggerstaff
11184-warwick-davis
10982-harry-melling
96851-james-phelps
140368-oliver-phelps
8930-john-cleese
10992-chris-rankin
234923-alfred-enoch
234922-devon-murray
956224-jamie-waylett
11212-josh-herdman
20240-zoe-wanamaker
477-julie-walters
10991-bonnie-wright
871100-luke-youngblood
10987-verne-troyer
1643-adrian-rawlins
10988-geraldine-somerville
1220119-elizabeth-spriggs
19903-richard-bremmer
58778-nina-young
10732-terence-bayler
1815748-harry-taylor
10655-leslie-phillips
1261131-simon-fisher-becker
10984-derek-deadman
56650-ray-fearon
11183-eleanor-columbus
10986-ben-borowiecki
1796502-danielle-tabor
1796505-leilah-sutherland
11185-emily-dale
1796509-will-theakston
1797001-jamie-yeates
10979-saunders-triplets
1796507-david-holmes
1796510-scot-fearn
1795303-jean-southern
1639982-kieri-kennedy
430776-leila-hoffman
143240-julianne-hough
1462953-zoe-sugg
```

```
1214513-jimmy-vee
1019545-derek-hough
1230975-dani-harmer
1232615-mark-ballas
225473-paul-marc-davis
1507605-violet-columbus
1430611-paul-grant
```

In [3]:
```python
import plotly.graph_objects as go
import pandas as pd
from hw5module import *


def process_data(df):
    # Count the occurrences of movie_or_TV_name
    counts = df['movie_or_TV_name'].value_counts()
    # Keep only records appearing more than twice
    counts = counts[counts >= 2]
    # Create a DataFrame to save the results
    result_df = pd.DataFrame({'movie_or_TV_name': counts.index, 'appearance_
    # Sort by appearance count in descending order
    result_df = result_df.sort_values(by='appearance_count', ascending=False
    # Save the results to result.csv
    result_df.to_csv("result.csv", index=False)
    return result_df


# Plot a bar chart
def plot_bar_chart():
    # Read the result CSV file
    result_df = pd.read_csv("result.csv")

    # Create a bar chart
    fig = go.Figure([go.Bar(x=result_df['movie_or_TV_name'], y=result_df['ap

    # Set the title and axis labels for the chart
    fig.update_layout(
        title="Appearance Count of Movies or TV Shows",
        xaxis_title="Movie or TV Show Name",
        yaxis_title="Appearance Count"
    )

    # Show the chart
    fig.show()


if __name__ == '__main__':
    df = parse_full_credits("385687-fast-x")
    data = process_data(df)
    plot_bar_chart()
```

```
12835-vin-diesel
17647-michelle-rodriguez
8169-tyrese-gibson
8171-ludacris
56446-john-cena
1251069-nathalie-emmanuel
22123-jordana-brewster
61697-sung-kang
117642-jason-momoa
928572-scott-eastwood
1784612-daniela-melchior
64295-alan-ritchson
15735-helen-mirren
60073-brie-larson
976-jason-statham
6885-charlize-theron
13299-rita-moreno
22462-joaquim-de-almeida
2984075-leo-a-perry
37149-luis-da-silva-jr
1678751-jaz-hutchins
1897706-luka-hays
4074147-alexander-capon
1427948-pete-davidson
4074148-shadrach-agozino
2282001-ludmilla
508582-miraj-grbic
3184164-meadow-walker-thornton-allan
124304-michael-irby
4074151-shahir-figueira
2545367-ben-hur-santos
123846-debby-ryan
2138286-josh-dun
18918-dwayne-johnson
90633
8167-paul-walker
4211960-ali-baddou
3579266-emily-buchan
```

Web Scraper Analysis Report for "Harry Potter and the Philosopher's Stone" Introduction This report outlines the functionality and performance of the parse_full_credits function within a web scraping project aimed at extracting detailed actor and filmography data from The Movie Database (TMDB). This function is crucial for gathering data on shared actors across different movies and TV shows starting from a specific movie, in this case, "Harry Potter and the Philosopher's Stone."

Setup of the Project The scraper is implemented in Python, utilizing libraries such as requests for fetching web content and BeautifulSoup for HTML parsing. This setup is designed to navigate the structured data of TMDB's website to access and process information efficiently.

Function Overview parse_full_credits(movie_directory) Functionality: This function is responsible for retrieving the full cast of a specified movie by its directory on TMDB. It captures unique identifiers for each actor, which are subsequently used to extract detailed filmography through further scraping processes.

Code Execution:

The function constructs a URL to access the 'Full Cast & Crew' page of the specified movie using the movie_directory parameter. It uses requests.get to fetch the page and BeautifulSoup to parse the HTML content. The cast section is located, and each actor's TMDB profile URL is extracted and stored in a set to ensure uniqueness. These actor profiles are then iterated over to gather comprehensive data on other movies or TV shows they have been involved in, employing the parse_actor_page function. A delay of 0.5 seconds is maintained between requests to respect TMDB's rate limits and scraping policies. Expected Results Upon execution, parse_full_credits('671-harry-potter-and-the-philosopher-s-stone') is expected to produce a DataFrame populated with data about all actors from "Harry Potter and the Philosopher's Stone." This data includes their names and the titles of other movies or TV shows they have worked on. The DataFrame should be sorted by movie or TV show name, providing a structured overview of the cast's broader filmography.

Conclusion The parse_full_credits function is a vital component of our web scraping project, enabling us to trace actor linkages across different cinematic works efficiently. By automating the extraction of actor and filmography data from TMDB, we facilitate deeper analysis into trends and connections in film and television, enhancing our ability to recommend related content based on actor overlap. This function exemplifies the project's capability to harness web data for enriching the viewer's experience through personalized content discovery.

In [ ]: