Chalse Okorom

SI 206 Final Project – API Mashup – Twitter & OMDB
README

This code gathers and caches information about three movies: *Queen of Katwe, Vampire in Brooklyn,* and *The Book of Eli* using the OMDB API. Then, these three movies are used are used as search terms using the Tweepy Twitter API. Using this information, I gathered information about people who tweeted about these movies recently as well as information about the 'neighborhood' of those users; this data was then put in a database that the program created, and output in a .txt file along with a couple of calculated stats. This program can be used to generate movie stats, tweets and user information for users associated with different movies.

To run this project, you need to run **206_final_project.py**; this program will generate additional files:

- 206_final_project_database.db: a database file to keep track of generated movies, users and tweets tables
- 206_final_project_cache.json: a JSON file to store cached information from OMDB and Tweepy
- 206_final_project_output.txt: text file with the program's output.

Dependencies:
NOTE: In addition to these dependencies you will need an additional .py file named twitter_info that contains your sensitive, personal information that will afford access to the Twitter API. This should include variables containing: access_token, comsumer_secret, consumer_key and access_token_secret variables provided when you successfully gain API access. Information about gaining access can he found here: http://docs.tweepy.org/en/v3.5.0/auth_tutorial.html

- tweepy
- json
- sqlite3
- omdb
- unittest
- Counter (from collections)
- time

Classes:
Movie:

- One instance represents one movie that was requested info from OMDB.
- Constructor Input:
    - imdb_id: IMDB movie ID
    - title: movie title
    - imdb_rating: IMDB rating
    - year_released: year released in theatres
    - studio: production studio
    - actors: top-billed actors
- Methods:
    - __str__ ()
        - No additional input. Returns a sentence-long string summary of the information stored in the class.

    - actor_summary ()
        - No additional input. Returns a sentence-long string summary of the number of actors and top-billed actors in the movie.

Tweet:

- One instance represents one Tweet (and its associated values).
- Constructor Input:
    - movie: associated movie search term (movie title)
    - tweet_id: tweet's ID
    - text: the text of the tweet
    - user_id: tweet poster's ID
    - retweets: retweets received by tweet
- Methods: None

User:

- One instance represents one movie that was requested info from OMDB.
- Constructor Input:
    - user_id: user's ID
    - screen_name: user's screen name
    - num_favs: number of item's a user has ever favorited
    - description: user bio (or description)
- Methods: None

Functions:

request_movies ()

- Input: (required) A list of three movie title strings.
- Return Value: A list of Movie instances
- Other Applicable Behavior: Requests movie data from OMDB and stores that data in the json file, caching that information for later use under the key 'omdb_movie_requests'. It then uses that data, to create movie instances and returns those.

cache_tweets ()

- Input: (required) A list of three movie title strings.
- Return Value: None
- Other Applicable Behavior: Cache's each movie's associated tweets if they haven't already been cached. A movie's associated instances can be found in the cache by using a key of this format: *movie title*_tweets

get_tweets ()

- Input: (required) A list of Tweet dictionary requests.
- Return Value: A list of Tweet instances associated with all movies.
- Other Applicable Behavior: Must be called after cache_tweets (). It takes the cached information from the cache file for each movie, creates, and returns a list of all Tweet instances.

get_neighborhood ()

- Input: (required) A list of Twitter dictionary requests
- Return Value: A list of User instances.
- Other Applicable Behavior: Cache's the information of each user who either posted a Tweet in the list or who was mentioned in one of the Tweets under the key 'user_neighborhood.' Then, returns a list of those created User instances.

Database:

The database has three tables within it: Movies, Users, and Tweets:
Movies:

- Rows:
    - imdb_id: IMDB movie IDs, containing TEXT attributes
        - This row is the PRIMARY KEY.
    - title: movie titles, containing TEXT attributes
    - imdb_rating: IMDB ratings, containing INTEGER attributes
    - year_released: year released in theatres, containing INTEGER attributes
    - studio: production studio, containing TEXT attributes
    - actors: top-billed actors, containing TEXT attributes

Tweets:

- Rows:
    - movie: associated movie search term (movie title), containing TEXT attributes
    - tweet_id: tweet's ID, containing TEXT attributes
        - This row is the PRIMARY KEY.
    - text: the text of the tweet, containing TEXT attributes
    - user_id: tweet poster's ID, containing TEXT attributes
    - retweets: retweets received by tweet, containing INTEGER attributes

Users:

- Rows:
    - user_id: user's ID, containing TEXT attributes
        - This row is the PRIMARY KEY.
    - screen_name: user's screen name, containing TEXT attributes
    - num_favs: number of item's a user has ever favorited, containing TEXT attributes
    - description: user bio (or description), containing TEXT attributes

Data Queries:

1. Select the title attribute for the movie with the highest rating. This is useful because it will give me access to best rated movie so I can print out which movie was rated the most so the user can get more familiar with our data.
2. Select the retweets, title, and text attributes for the tweet with the most retweets. This is useful because it allowed me to use that queried information to print out information about the most retweeted retweet and its associated movie.
3. Select the user screen_name and num_fav attributes for users whose screen_names are less than 10 characters long. This is useful because it allowed me to get access to a very specific subsect of user information that I could perform averages and other operations on.
    a. Additionally, allowed me to find average number of favorites for these users.
4. OTHER QUERIES:
    a. Select entirety of Tweets and Users. This allowed me to access specific data I wanted to add for each table summaries in the output files.

Output File Contains:

- Highest Rated movies
- Most Retweeted Tweet About a Movie
    - Number of times that it was retweeted
- Screen names of users whose screen names have less than 7 characters
    - The average favorites of these users
- Movie Summary

    - __str__() and movie_summary() methods are output for each movie
- Tweets Summary
    - For each distinct tweet (by primary key) in the database, print the output of the text as well as its number of retweets.
    - For each distinct user (by user id) in the database, print the output of the screen name, number of favorites and Twitter bio.

Why did I choose this project?

- My favorite part of the class was working with the Twitter API and learning the ins and outs of things like string manipulations working with BeautifulSoup for the New York Times website. I also really liked exploring the Twitter API as someone who's never used Twitter before.

NOTE: SPECIFICS FOR SI 206

- Line(s) on which each of your data gathering functions begin(s):
    - 102 – request_movies()
    - 112 – cache_tweets()
    - 121 – get_tweets()
    - 131 – get_neighborhood()
- Line(s) on which your class definition(s) begin(s):
    - 48 – Movie
    - 74 – Tweet
    - 88 – User
- Line(s) where your database is created in the program: 154
- Line(s) of code that load data into your database: 158 - 185
- Line(s) of code (approx.) where your data processing code occurs — where in the file can we see all the processing techniques you used: 189 – 218
- Line(s) of code that generate the output: 237 – 262