Sydney Lauer, Megan Defrancesco, Maya Posner

Final Project Report: Movie Modelers

Repository: https://github.com/mayaposner/SI206Movies.git

Original Goals

Our original goals were to use the OMDB API, the Rotten Tomatoes API, and Beautifulsoup to scrape the official Oscars website. Using these three sources, we had planned to collect data about Oscar-winning and Oscar-nominated movies such as the movie title, year it was released, ratings, box office profits, and the category of the award it was nominated for. We aimed to calculate whether box office profits pulled from OMDB or critic scores pulled from Rotten Tomatoes had more of a correlation with a movie winning an Oscar. For our visualizations, we planned to create a bar graph comparing critic review scores with audience review scores for the Rotten Tomatoes API, a scatterplot to look at the relation between OMDB box office profits and critic review scores for each movie, and a pie chart for the Oscars website representing the percentage of awards each movie won. By accomplishing these tasks, we hoped to gain practice with pulling data from APIs and Beautifulsoup, inputting data into databases using SQL, and visualizing data using the visualization package Matplotlib.

Goals Achieved

After trials and errors through our original API's, we altered our project so that we pulled data from the IMDB API, the iTunes API, and the Watchmode API. From the IMDB API, we were able to create a list of some of the top movies from 2019-2021, find their Metacritic ratings, and see their Audience ratings. With this data, we answered the question, "Do critics or audience members rate movies higher?", and stored the data in both a bar graph and pie chart. We then used our iTunes API to find the price that the movies were sold for and how that price compared to the movie rating, and created a scatter plot to view that comparison and find out if there is a linear trend. Then, we joined data from the Watchmode API and our IMDB list to utilize the IMDB movie IDs in order to find the maturity rating (G, PG, etc.) of each movie. We visualized the frequency of these maturity scores through a pie chart. Through the data pulled into tables from our API's and the visualizations we created, we answered several questions that we had regarding modern day films. 1. Critics and Audiences rate movies very similarly (typically within ~8 points of each other), but the majority of the time Critic scores are higher. 2. There is not a linear correlation between critic rating and price, therefore there must be more factors that go into setting a movie price. 3. PG-13 is the most common maturity score amongst our movies. By filling a database with multiple tables of this data and generating several graphs and plots in order to visualize it, we gained proficiency working with SQLite and MatPlotLib. Most

importantly, we successfully coded this project as a team by utilizing github and Zoom screen sharing features, and being readily available to help each partner work through bugs and errors in our codes.

Project Problems

Problem 1 BS Oscars: Originally, we had planned to use Beautiful Soup to scrape the winning movies from the Oscars website. The HTML was consistent across years, but tags switched when awards were given to an individual rather than a movie.

Solution: We switched from the official Oscars' website to The Wrap. We switched to the wrap because the tags appeared to be the most consistent.

Problem 2 BS The Wrap: When restarting BS using The Wrap 2019, we wanted to get the winners. Originally, we used the strong class because winners were bolded. This proved to be a poor decision because too many other elements were bolded.

Solution: We switched from using strong to class+ = 'ul1'

Problem 3 BS The Wrap: Once we could collect all of the li tags within the ul, we needed to only get the movie title. After all, the inability to efficiently grab the movie title from the Oscars' website is why we switched to The Wrap. The method was to use regular expressions to match the portion of the li that represents the movie title. After many hours of debugging and comparing code to regex101, there was still an error.

Solution: Finally, we released that The Wrap, Regex101, and VS Code have different characters for quotation marks. As the solution, we copy and pasted the quotation marks from The Wrap into Regex101 and VS Code.

Problem 4 BS The Wrap: After grabbing all of the relevant information from The Wrap 2019, we realized that we would have to extend our year search to get 100 rows of data. When looking at the other years, the HTML tags were not consistent. We would have to edit our BeautifulSoup and regular expressions every year. This was incredibly inefficient.

Solution: We switched to the Golden Globes

Problem 4 BS Golden Globes: At first, the Golden Globes's website seemed very promising. Between 2016 and 2020, the HTML was consistent; however, we ran into a similar issue at Problem 1. Once the awards switched from individuals to movies, the HTML tag switched.

Solution: Switch to an API

Problem 5 Awards Logic Problems: While working aforementioned syntax errors, we also discovered logic errors within our tables. We did not account for linking the winning categories to the correct film.

Solution: Save the scraped data into a dictionary and use this dictionary to cross reference movie_ids from the Movies table when inserting into an Oscars table. While this method seemed promising, given all of the other Beautiful Soup issues, we decided to switch to an API.

Problem 6 Watchmode API: We have one error with using Watchmode API. We are not able to run this code more than two times at a time. As a result, the Maturity_Scores table does not reach 100 rows. While our code appeared to be correct in the debugger, we did not continue to solve this error.

Solution: Watchmode limits our usage to 1000, so we decided to leave the code and continue.

Please Note: We started the project with plenty of time, but continued to run into errors with BS. Given the number of hours dedicated to working on BS (+10 hours), we decided it was in the group's best interest to switch to an API.

Calculations

iTunes_data.txt

```
The average movie price for the iTunes movies that were sampled is $12.93.
The average Metacritic Rating for the IMDB movies that were sampled is 73.
There were 3 movies where a higher than average price correlated with a higher than average rating, or a lower than average price correlated with a lower than average rating.
```

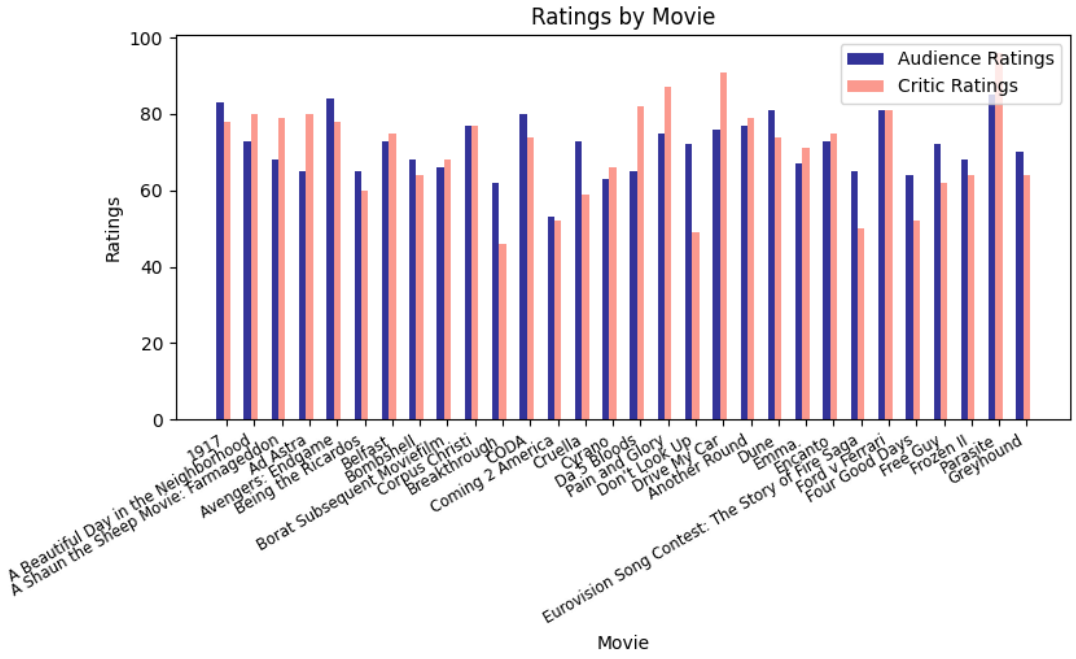
ImdbOutfile.txt

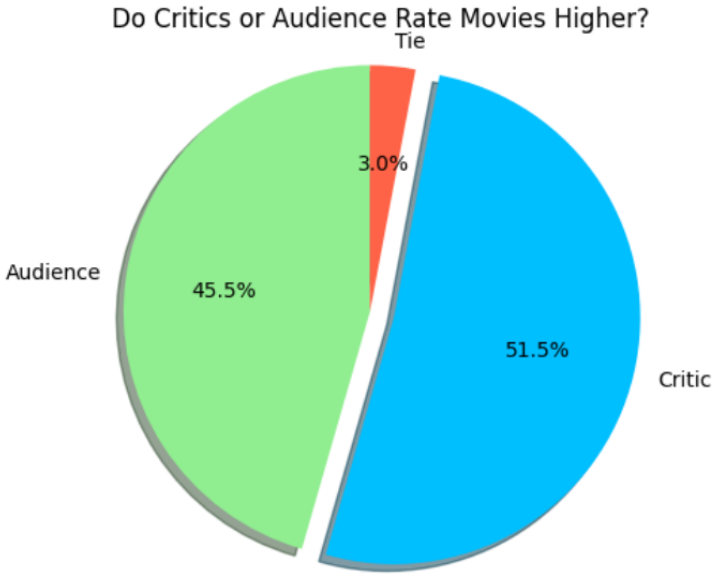
```
Number of times (out of 100) the Audience score was greater than the Critic score: 46
Number of times (out of 100) the Critic score was greater than the Audience score: 52
Number of times (out of 100) the Audience score was the same as the Critic score: 3
Average difference between critic and audience rating scores: 8.544554455445544
```

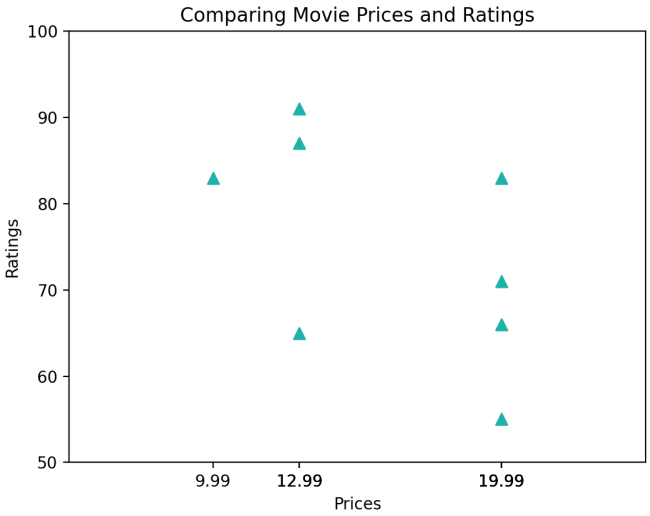
Maturity.txt

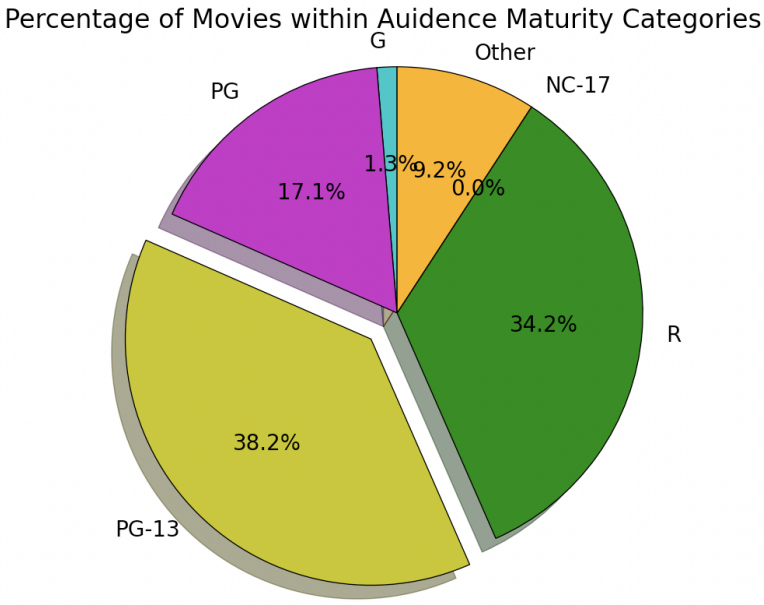
```
The total number of G rated movies is 1.
The total number of PG rated movies is 13.
The total number of PG-13 rated movies is 29.
The total number of R rated movies is 26.
The total number of NC-17 movies is 0.
The total number of not rated movies is 7.
```

Visualizations






Comparing Movie Prices and Ratings
Ratings
Prices
100
90
80
70
60
50
9.99
12.99
19.99


Percentage of Movies within Auidence Maturity Categories
G
Other
NC-17
PG
R
PG-13
1.3%
9.2%
0.0%
17.1%
34.2%
38.2%

Code Instructions

Run in the Following Order:

BaseFile.py

1. Run the main method at least 5 times to accumulate 100 rows in the “Movies” table within the Movies.db database.

Imdb_api.py

1. Run the main method (as is) 4 times to accumulate 100 rows in the “IMDB_Ratings” table within the Movies.db database.
2. Comment out the “movieratings = retrieveIMDBdata()” and “fillRatingTable(movieratings, cur, conn)” lines in the main function now that the table is filled.
3. Uncomment the last four lines of the main method in order to visualize and calculate the data, and write the calculations to “ImdbOutfile.txt”.

iTunes_api.py

1. Run the main method (as is) at least 5 times to accumulate 100 rows in the “iTunes” table within the Movies.db database.
2. Uncomment the two last lines of the main method and run the code to calculate and visualize the data.

AudienceScores.py

1. Run the main method (as is) 2 times to accumulate 49 rows in the “Maturity_Scores” table within the MOvies.db database.
2. Within the main method, comment the following lines:
    - us_rating_list = retrieveWatchModeScore(cur, conn, formatted_ids)
    - fillMaturity_ScoresTable(cur, conn, us_rating_list)
3. Then, remove the comment from the following line:
    - `# pieChartAndText('Maturity.txt', data)`

Documentation

BaseFile.py

```
def setUpDataBase(db_name):
    ''' This function takes in the name of the database ('Movies.db') as input.
    It sets up the databse and returns the database cursor and connection as output. '''

def createMainTable(cur, conn):
    ''' This function takes in the database cursor and connection as input. It creates a
    table called "Movies" if one does not already exist with a column for movie titles and
    a column for corresponding movie IDs. This function does not return anything. '''

def retrieveIMDBdata():
    ''' This function does not take any parameters as input. It sends a request to the IMDB API
    to get data about movies released in 2019, 2020, and 2021. The text of this response in the
    form of a JSON string is converted to a dictionary object from which movie titles are extracted
    and stored in a list. The function returns this list of movie titles as output. '''

def fillMovieTable(movies, cur, conn):
    ''' This function takes in the list of movies returned from retrieveIMDBdata(), the database cursor,
    and the database connection as input. It retrieves the number of rows already in the Movies table and
    loops through the retrieved list of movie titles to add 25 new rows to the Movies table.
    This function does not return anything as output.'''
```

AudienceScores.py

```
def maturityTable():
    '''
        Opens the database Movies.db
        Creates a new table called Maturity.
        Returns cur and conn
    '''

def fillMaturityTable(cur, conn):
    options_list = ["G", "PG", "PG-13", "R", "NC-17", "Other"]

    for i in range(len(options_list)):
        cur.execute("INSERT or IGNORE INTO Maturity (id,Options) VALUES (?,?)",(i ,options_list[i]))
    conn.commit()
    '''
      Takes in cur and conn
      Uses the items in the options_list to populate the Maturity table
    '''

def movieMaturities(cur, conn):
    cur.execute('CREATE TABLE IF NOT EXISTS Maturity_Scores (movie_id INTEGER PRIMARY KEY UNIQUE, maturity_id NUMERIC)')
    conn.commit()
    '''
      Takes in cur and conn
      Creates a new table called Maturity_Scores
    '''

def retrieveImbdID(cur, conn):
    '''
      Takes in cur and conn
      Selects the imbd_ids from IMDB_Ratings table to be used with the API
      Returns the list of the imbd_ids
    '''

def retrieveWatchModeScore(cur, conn, formatted_ids):
    '''
      Takes in cur, conn, and the list of formatted IMBD ids
      Loops through the formmated IMBD ids and uses them as a URL parameter for the Watchmode API
      From the python dictionary, retreived the us_rating
      Select each movie id
      Create tuple of movie id and rating id
      Add the tuples to a list called us_rating_list
      Return us_rating_list
    '''

def fillMaturity_ScoresTable(cur, conn, us_rating_list):
    '''
      Takes in cur, conn, and the list of tuples (us_rating_list)
      Retreives the number of rows in the table
      Loops through to add 25 new rows
      Fill Maturity_Scores table with the category id (stored as Options in the Maturity table)
    '''

def retrievePieChartData(cur, conn):
    '''
      Takes in cur and conn
      Selects movie titles and Options to use in calculations and visualizations
      Returns this in a variable called data
    '''

def pieChartAndText(file, data):
    '''
      Takes in a file for writing and data
      Loops through the data to get maturity ratings
      If the maturity rating matches the category, the count for the category increases
      Settings up pie chart
      Creating pie chart
      Show pie chart

      Setting up writing the file
      Writing
    '''
```

Imdb_api.py

```
def createRatingTable():
 """Opens the database Movies.db and creates a new table called IMDB_Ratings. Returns cur and conn."""

def retrieveIMDBdata():
 """Pulls data from the IMDB API and adds the Metacritic Rating, Audience Rating, Movie Title, and Movie ID
from each movie into a tuple. Returns a full list of these tuples."""

def fillRatingTable(movieratings, cur, conn):
 """Takes in the list of tuples from retrieveIMDBdata, cur, and conn. Uses the items in the tuple to populate
 the IMDB_Ratings table. Additionally, uses the movie title in the tuple to select the movie_id from the
 Movies table, and adds that id into the IMDB_Ratings table."""

def graph_ratings_movies1to30(cur, conn):
 """Takes in cur and conn in order to select data from the Movies table and IMDB_Ratings table that will
 be used in a bar chart. Then uses MatPlotLib to create a bar chart showing the comparison between critic
 score and audience score for the first 30 movies in the IMDB_Ratings table."""

def pieChartofRatings(cur, conn):
 """Takes in cur and conn in order to select data from the IMDB_Ratings table that will be used in a pie chart.
 Then uses MatPlotLib to create a pie chart showing how often the critic rating > audience rating, how often
 audience rating > critic rating, and how often critic rating = audience rating. Returns a tuple of the number
 of times in the data that we saw each of these categories."""


def findAverageRatingDifference(cur, conn):
 """Takes in cur and conn in order to select data from the IMDB_Ratings table to be used in a calculation.
 Calculates the average difference between audience ratings and critic ratings. Returns average value from
 calculation."""

def averageRatetoFile(cur, average, audience, critic, tie):
 """Takes in cur, the average value returned by findAverageRatingDifference, and the audience, critic, and tie
 values from the tuple returned in pieChartofRatings. Uses these inputs in order to write calculations for
 the number of times Audience Score > Critic Score, the number of times Critic Score > Audience Score, the number
 of times Audience score = Critic Score, and the average difference between Critic and Audience Scores to a text
 file 'ImdbOutfile.txt'."""
```

iTunes_api.py

```
def createTable():
    ''' This function does not take any parameters as input. It establishes
    a connection to the Movies.db databse and a corresponding cursor. It then
    creates a table called iTunes in the database if one does not already exist
    that has a movieID column (that corresponds to the Movies table) and an
    iTunes price for that movie. This function returns the cursor and
    connection as output.'''

def getData():
    ''' This function does not take any parameters as input. It uses the
    iTunes API to send three respective requests for movies released in the
    years 2019, 2020, 2021. The response objects for each request are
    converted from a JSON strings to python dictionaries from which the
    title and price of each movie are extracted and added to a list of tuples.
    This function returns this list of tuples with each movie title and price
    as output. '''

def addiTunesToMovies(tupls, cur, conn):
    ''' For input, this function takes in the list of tuples returned from
    getData(), the database cursor, and the database connection. It retrieves
    the amount of rows already in the Movies table and adds 25 more rows
    of iTunes movies using the data from the tuples list. The function checks
    to make sure no repeat movies are added to the Movies table. This
    function does not return anything as output.'''


def fillItunesTable(tupls, cur, conn):
    ''' This function also takes as input the list of tuples returned from
    getData(), the database cursor, and the database connection. It retrieves
    the amount of rows already in the iTunes table and adds the movieID and
    price of 25 movies to the iTunes table. This function does not return
    anything as output.'''


def writeCalculatedData(file, cur, conn):
    ''' This function takes as input a file to which calculated data will
    be written ('iTunes_data.txt'), and the database cursor and connection.
    It calculates the average price of the movies in the iTunes table
    and the average rating of the movies in the IMDB table. Using a JOIN
    statement, it retrieves the prices and ratings for all of the movies
    that are in both the IMDB and iTunes tables. The function then
    calculates how many times a movie has a both a higher than average
    price and rating, or both a lower than average price and rating,
    to analyze a potential correlation between the two. The function writes
    the calculated data (average price, average rating, and correlation
    quotient) to the passed file and returns lists of the retreived prices
    and ratings as output. '''

def visualize_data(prices, ratings):
    ''' This function takes as input the prices list and ratings list
    returned from writeCalculatedData(). Using matplotlib, it creates
    a scatterplot where the x-axis gives the price of the movies and the
    y-axis gives the ratings of the movies. The scatterplot is designed
    to further anylize if there is any correlation between movie prices
    and ratings. The function displays the scatterplot but does not return
    anything further as output. '''
```

Resources (each)

Date	Issue Description	Location of Resources	Result
4/21/21	Meet with Angela Young (her name is on Canvas/an email about UMSI sponsored tutors) to discuss the logic issue. Maya emailed her explaining that she knows we have a logic error but can not determine how to link the category to the movie.	Zoom	Helped brainstorm the dictionary method. Ultimately did not end up using it because we did not continue with BS
4/21/21	OH with Simon: addressing problems 2 and 3, and checking if problem 6 method of using a dictionary would work	Zoom	Switched to using ul1 class, found the quotation mark error, validated that new method works logically
4/22/22	Needed assistance creating a pie chart through MatPlotLib.	https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html#sphx-glr-gallery-pie-and-polar-charts-pie-features-py	Found sample code for a pie chart that I could edit to best fit my data.
4/22/22	Needed assistance creating a bar graph through MatPlotLib.	https://pythonspot.com/matplotlib-bar-chart/	Found sample code for a bar graph that I could edit to best fit my data.
4/22/22	The movie titles are very long, and therefore would not fit as horizontal ticks on the bar chart.	https://stackabuse.com/rotate-axis-labels-in-matplotlib/	Learned how to rotate the bar chart ticks so that you could see the full title for every movie included in the chart.
4/23/22	Wanted to change the colors for my bars on the bar graph and the pie chart.	https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py	Found colors that I felt added good contrast for my visualizations.

4/24/22	**Needed to remove two different tables (Years and Oscars) from our Movies database because we no longer were using the data from those tables.**	**https://www.tutorialspoint.com/python_data_access/python_sqlite_drop_table.htm**	**Learned the DROP TABLE function in SQLite and successfully dropped the old tables.**