Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

Final Project Report - S&M&L

GitHub Repository: https://github.com/lpesce57/SI206_final_project.git

Initial Goals

Our initial plan for the project was to gather information from various movie rating websites and compile it to generate data from the top 100 movies from 2015-2023 and list of movies that are currently streaming in theaters. We initially sought out to utilize API's from IMdB, Rotten Tomatoes, and Fandango to obtain our data and insert it into a database. This data included the date of release, duration, and audience ratings regarding movies. Therefore, we aimed to compare top movies and their rankings on each website in relation to each other.

Ultimate Goals

After looking through many different websites and APIs, we decided on analyzing API data from OMdB, Rotten Tomatoes API, Open Weather API, and IMdB API for movie ratings. From Open Weather, we collected the date, maximum temperature, and minimum temperature from. From movies on OMdB, IMdB, and Rotten Tomatoes, we extracted movie titles from the Top 100 Movies list. We also obtained the release date, star-rating, and runtime for each of these titles. We used the Rotten Tomatoes API to search for the same titles and find their critic and audience score ratings. Our hypothesis was that people would be more likely to enjoy movies if the weather was objectively worse. We also wanted to explore the correlation between movie duration and star-rating, alongside the different ratings among sites. Therefore, our calculations showcase the average minimum and maximum temperatures regarding Open Weather, and the average rating for all top 100 movies within IMdB and OMDB, and the average tomato meter for the movies listed on Rotten Tomatoes. For our first plot, we used a bar graph to visualize the difference in ratings between Rotten Tomatoes and IMdB. On average, there appears to be a negative correlation, as Rotten Tomatoes tends to produce lower ratings than IMdB. For our second plot, we used a scatter plot to visualize the correlation between weather and overall movie ratings. Our data suggests a weak correlation, however, it appears as though ratings were higher when the temperatures were lower, which is contrary to our hypothesis. Our third plot also is displayed in a bar graph to visualize the relationship between movie genre and ratings. It suggests a constant correlation.

For our extra credit, we created two more visualizations to put our findings into a deeper perspective. Our first graph showcases the correlation between duration of movies in minutes and their ratings in a scatter plot. It suggests that the shorter the movie is, there tends to be a lower rating. Our second graph is a line graph and shows the correlation between the duration of the movies in minutes and their release dates. Our data suggests that movies that were produced earlier were shorter, on average, and movies that were produced later gradually became longer, on average. However, there are significantly more outliers among movies that were released on a later date.

Limitations

Problem 1: We found it quite difficult to start the project at the beginning. Since we had to code our project from scratch, figuring out how and where to start was probably the hardest part. In addition, the API's for all of the sites were very difficult to locate. Our biggest struggle regarding

Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

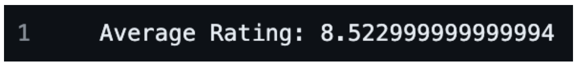
this issue was that many sites required us to purchase API keys, and even after doing so, there was no guarantee that would be available for usage right away. For example, we tried to generate the API for Fandango, which although it was free, the API key remained unavailable for a long period of time, so we were unable to use it.

Problem 2: Open Weather was where we confronted the majority of our issues. The API was only free per 1000 requests, and we ended up running into an issue regarding our code for Open Weather because of this limit, therefore spending a large amount of time focusing on resolving this issue. Moreover, the Open Weather API we found only went back 40 years. Therefore, since some of the movies were released over 40 years ago, we were unable to access the weather data for those movies.

Problem 3: As for OMDB, we ran into an issue where it was only loading 98/100 movies because two of them had the same release date. Since we used the date as the shared integer key, there couldn't be two movies that shared the same release date. However, since it was only two movies, we were able to bypass this issue. There also was never an issue with our top 25 movies.

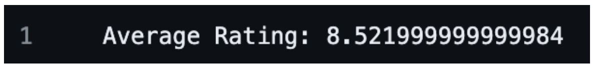
Calculations

Average OMDB Rating:



```
Average Rating: 8.522999999999994
```

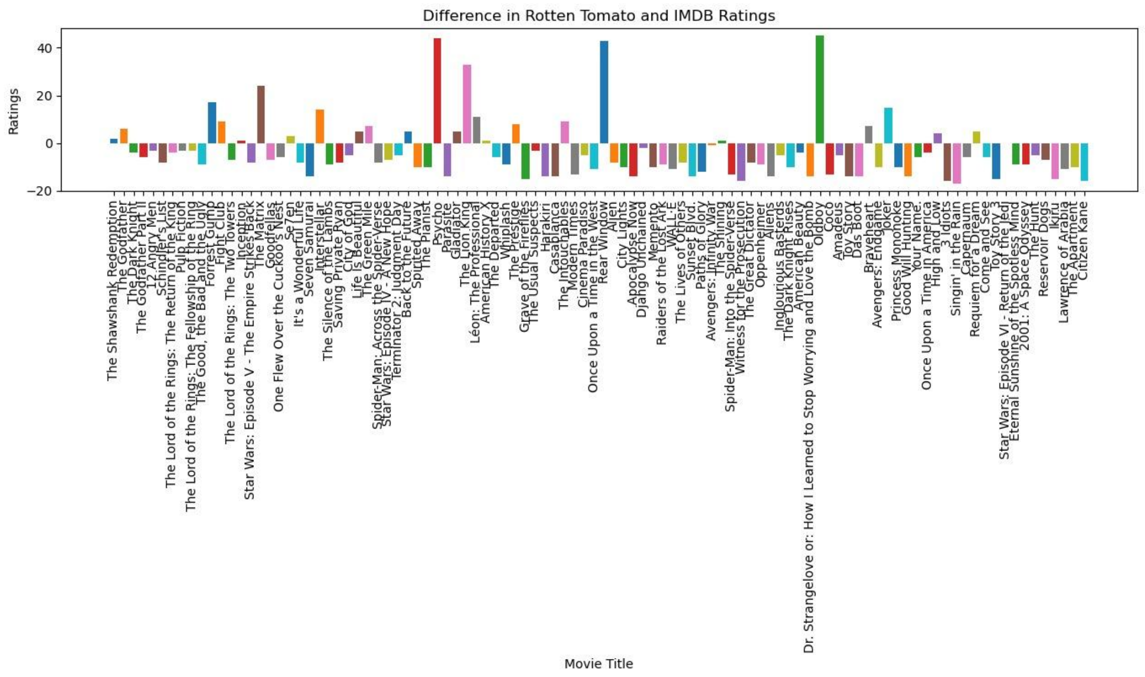
Average overall rating:



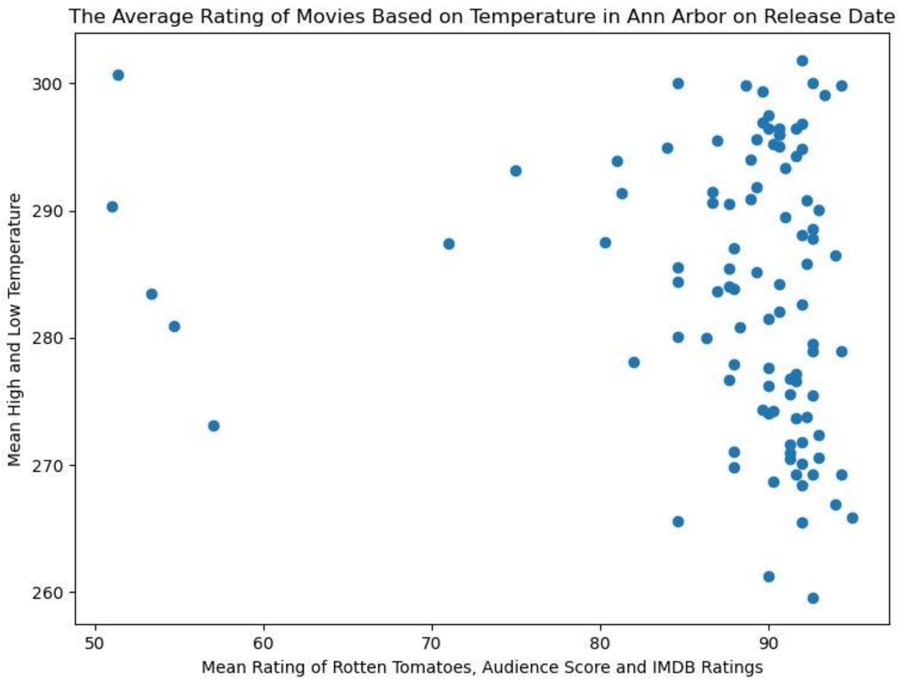
```
Average Rating: 8.521999999999984
```

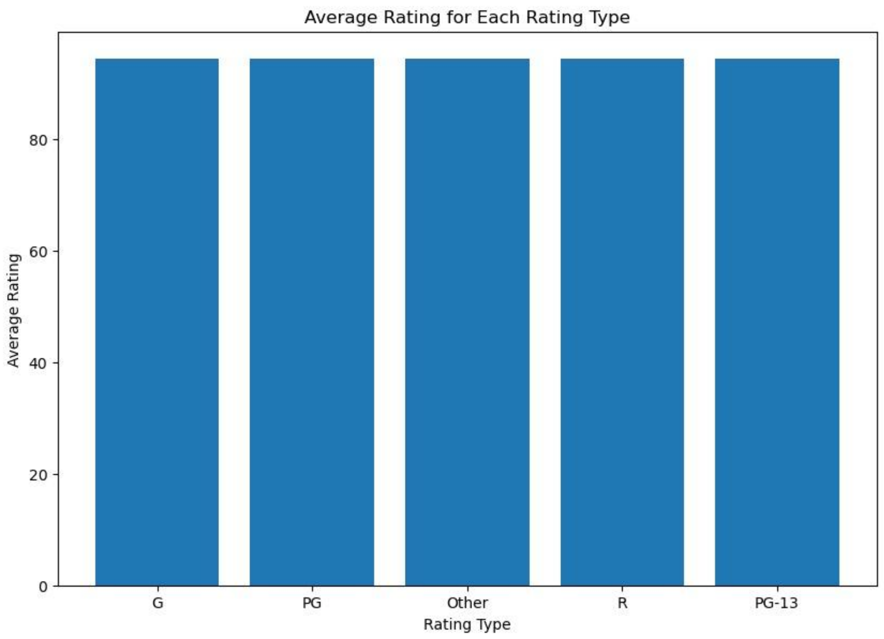
Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

Visualizations



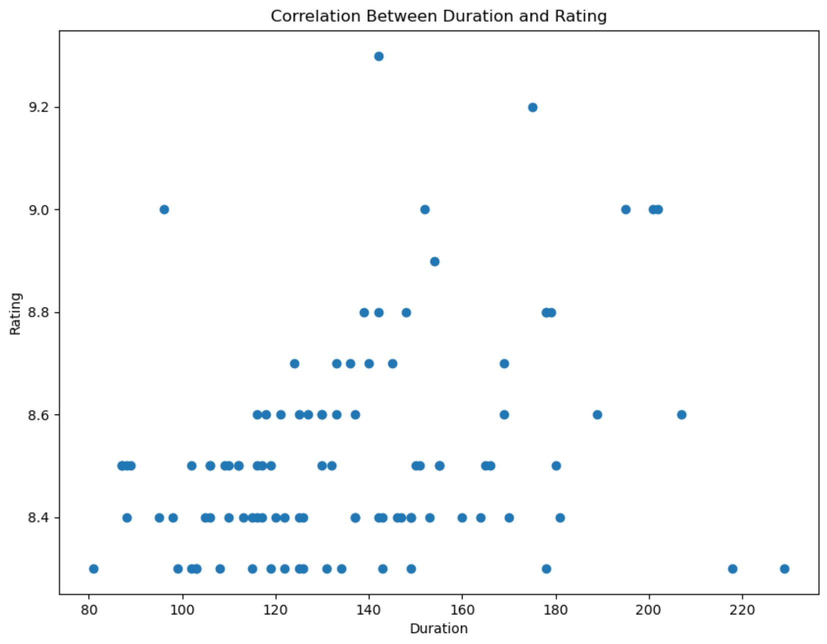
Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

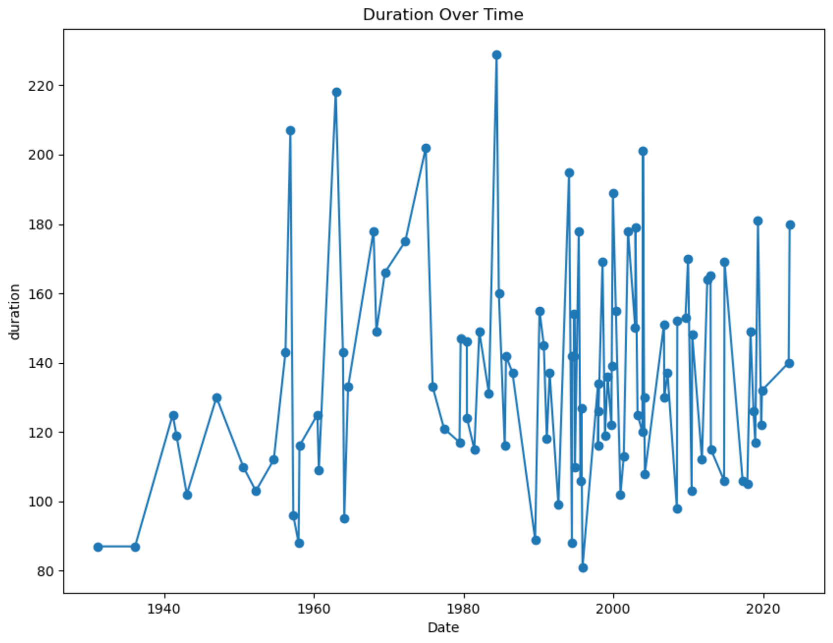




Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

Extra Credit Visualizations





Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

Instructions for running code

To run our code, we had to install API's for Open Weather, IMdB (API is called OMDB), Rotten Tomatoes, and TMDB. From there, we installed SQLite to enable us to load our data into databases. Once we installed all the required materials and wrote our code, with the API's and JSON imported, the data was automatically implemented into the databases. We ran our files in the order: IMdB, OMDB (which TMDB is combined with), Rotten Tomatoes, and Open Weather. We ran our files four times each, considering we could only run 25 listings at a time. Our SQL database includes all of our python files written into tables, and our shared integer key is the release date for movies, which we used in the OMDB data collection and. We ran separate python files for the extra credit visualizations to create graphs using matplotlib. All files are located in our GitHub repository.

Documentation for each function

omdb_api.py

After importing the SQL, the first function, def setUpDatabase(database), sets up the database that will read from OMDB. The second function, def get_movie_data(database), generates a list of all the movies collected from the IMdB API and adds all the movie titles to a list. The next few functions are centered around the structure of the data. def get_movie_data(database) structures the release date properly, and def create_movies_table(cur, conn) creates an OMDB data table with release date, movie title, movie rating, and the duration of the movie in minutes. def create_rated_table(cur, conn) regards the movie rating, and creates a table that takes in the ratings for each movie. def movie_in_database(cur, movie) double checks whether or not the movie is already in the database, and def date_in_database(cur, date) serves the same purpose for the release date. Once the data is organized and filtered, def get_omdb_data(cur, conn, titles) adds the data from IMdB into the database. It first checks if the movie is already in the database, moves one movie down the list if it is, and then obtains the movie data from the first movie not in the database, in addition to the next 25 movies after. Furthermore, it gets the title, year, rating and movie id for each movie and converts each of the data into the correct format. Lastly, it inputs all of the data into a table. def get_rated_data(cur, conn, date_titles) adds the data pertaining to movie rating from the OMDB to the database using the same process as def get_omdb_data(cur, conn, titles). def compute_average_omdb_rating(cur) computes the average ratings from OMDB and writes them to a text file, and the def main() function executes all above functions.

imdb_api.py

After importing the SQL and requests, this file begins with def setUpDatabase(database), which sets up the database that will read from IMdB. Then, def create_movies_table(cur, conn) creates a movie data table with id, title, year, rating, and duration, and def movie_in_database(cur, movie) double checks whether or not a movie is already in the database. def get_movie_data(cur, conn) ultimately adds the data from IMdB to the database, beginning with the top movie. It checks if the movie is in the database, which if it is, it moves down the list by 1. Then, it gets the movie data from the first movie not in the database and the next 25 movies after, proceeding with obtaining the title, year, rating and movie id for each movie, ensuring that all data are formatted correctly. Lastly, it gets a new URL based on the movie id to determine the movie duration and implements all data into a table. def compute_average_rating(cur) computes the average of the movie ratings from IMdB. The def main() function executes all above functions.

rotten_tomatoes_api.py

This file first imports requests and the SQL, then begins with def setUpDatabase(database), which sets up the database that reads from Rotten Tomatoes. Then, def get_movie_data(database) generates a list of all the movies in the database and iterates through all the titles in the database, adding them to the titles list. def create_movies_table(cur, conn) creates a rotten tomatoes data table with title, tomato meter and audience score, and def movie_in_database(cur, movie) double checks whether or not a movie is already in the database. def get_rt_data(cur, conn, titles) essentially adds all the data from Rotten Tomatoes to the database, by checking if the movie is in the list and moving down one movie if so, getting the movie data from the first movie not in the database and the next 25 movies after, obtaining the title, tomato meter, and audience score for each movie, and ultimately adding all data into a table. def compute_average_rt_scores(cur) computes the average from all scores on the tomato meter, and the def main() function executes all above functions.

open_weather_api.py

This file begins with importing requests and SQL from datetime. Then, def setUpDatabase(database) creates a database that will read from Open Weather, while def get_dates(database) makes a list of all the dates of movies released in the OMDB database. def date_in_database(cur, index) checks whether or not a date is already in the database, and def create_weather_table(cur, conn) creates a weather table containing date, maximum temperature, and minimum temperature for the designated date. def add_weather_data(cur, conn, dates) adds

Sam Gonzalez, Lauren Pesce, Maya Rai
SI 206 Final Project
Dr. Ericson
12/12/2023

the data to the database by first checking if the date is already there, if so moving one date down, obtaining the date data from the first date not in the database and the next 25 date after, and iterating through all dates in the data ensuring that they're correctly formatted, overriding the error code, and recording all high and low temperatures. Finally, it adds all data into the table. def compute_average_temperatures(cur) computes the average minimum and maximum temperatures, writes them to a text file, and the def main() function executes all above functions.

Documentation of resources

Date	Issue Description	Location of Resource	Result
12/11/2023	We had the wrong syntax for our SQL in terms of adding data to the table	Stack Overflow	SQL solved our issue by including problems and solutions implemented by other users
12/11/2023	We had problems with our Open Weather file in converting from Kelvins to Celsius	ChatGPT	ChatGPT helped us with implementing the right code to convert from Kelvins to Celsius
12/12/2023	We had some errors when trying to add the data into our SQL database in batches of 25 rows at a time	UMGPT	UMGPT helped us debug our code and figure out a simplified method of adding our information into the database.
12/12/2023	We had some confusion regarding the report, specifically the question about how to run our code.	Piazza	Piazza led us in the right direction in answering our confusion by describing how we need to explain how to run our code.