SI 206 final report: Movie Watchers

Github repository: https://github.com/rsonnett/206_finalproject.git

Original Goal: Does Box Office sales have any correlation with Rotten Tomato and IMDb ratings?

Our original goal was to webscrape Box Office mojo in order to get the box office sales for top 100 movies as well as the rating for the movies on Rotten Tomato and IMDb. After researching, we realized that boxofficemojo.com is owned by IMDb and we were worried that it wouldn't count as a third website, so we decided to change our question.

Achieved Goal: How does the release date of movies affect the rotten tomato and imdb ratings? We found an api key that gave us every official holiday from 2023 till now, so we decided to create our new question. We used the "Most Popular 2024 Movies in Theaters" list from the Rotten Tomatoes website to get our 122 movie titles. We used web scraping to find the movie titles as well as the audience score from Rotten Tomatoes, an api key to get the movie release dates from OMDB, api key to find the imdb rating of each movie, and api to find a list of holidays in 2023 and 2024, and their dates.

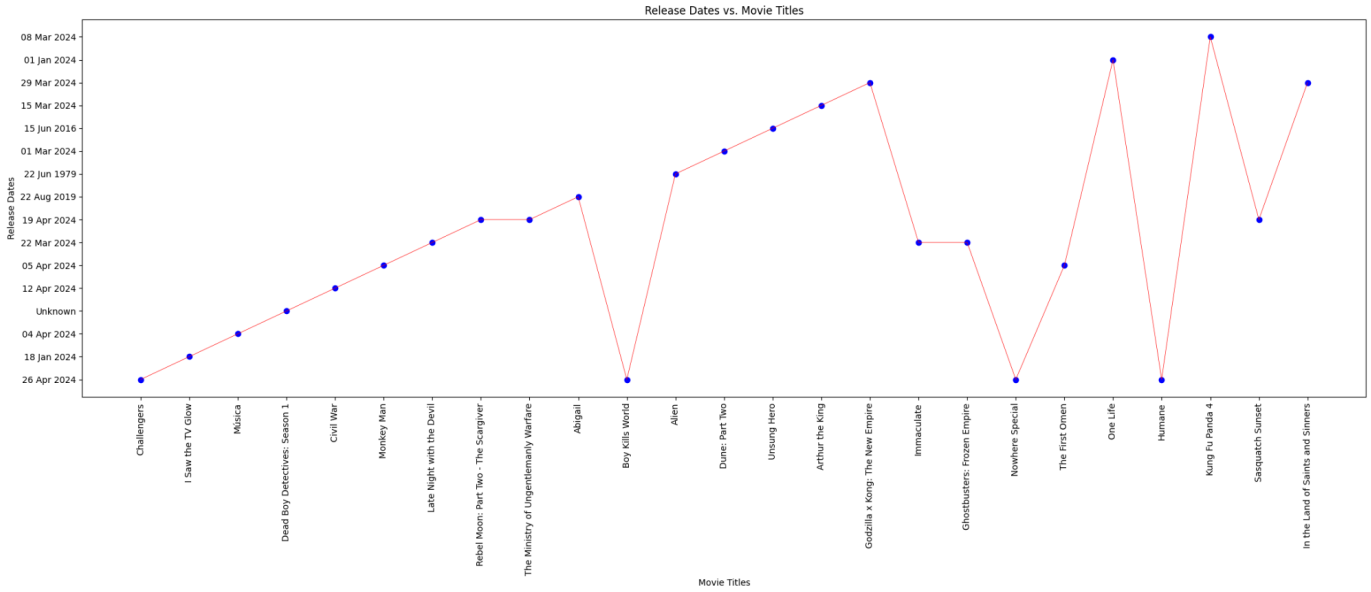
Problems: We finished most of the requirements for our project without having any issues, and then we began to get 429 errors on our openweather API. We spent a lot of time trying to get a better subscription to Open Weather because we thought that maybe they had a call limit per day, but even with purchasing a subscription that supposedly allowed us to get 2,000 calls/day we were still experiencing errors with the code. Our database was also very finicky, we had to run tables multiple times and exit out and rerun things constantly in order for tables to show up in the

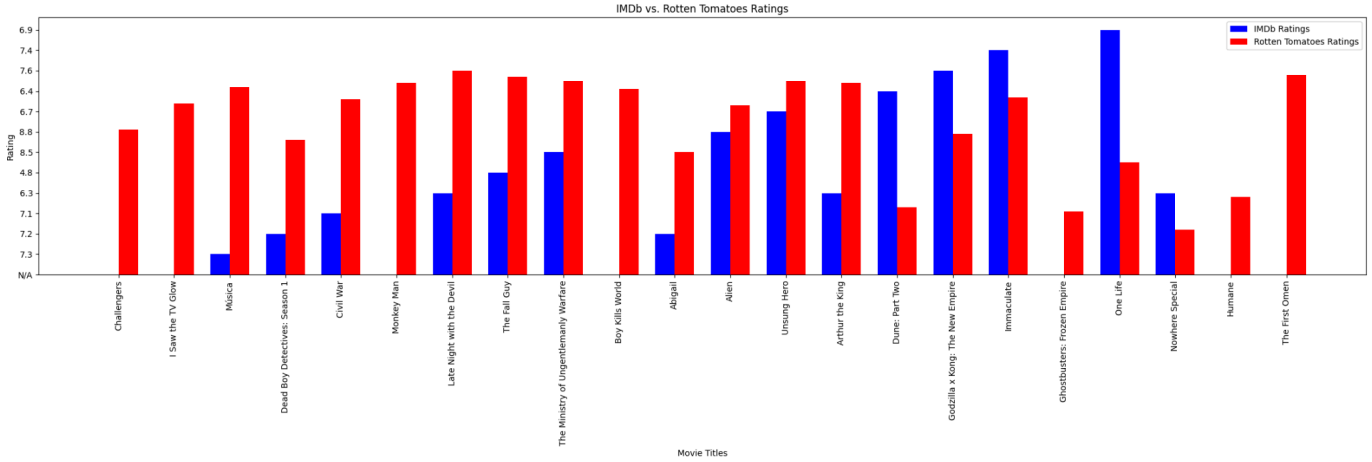
database. After all of the issues we experienced with the Open Weather API, we decided we needed to switch websites so we decided to incorporate holidays into our objective rather than temperature. After finding an API key with holidays, we struggled because when we retrieved the table, there was a holiday on every day of the year. Because we were trying to see if a movie being released on a holiday affected the rating, this would not be helpful to have a holiday listed as everyday.

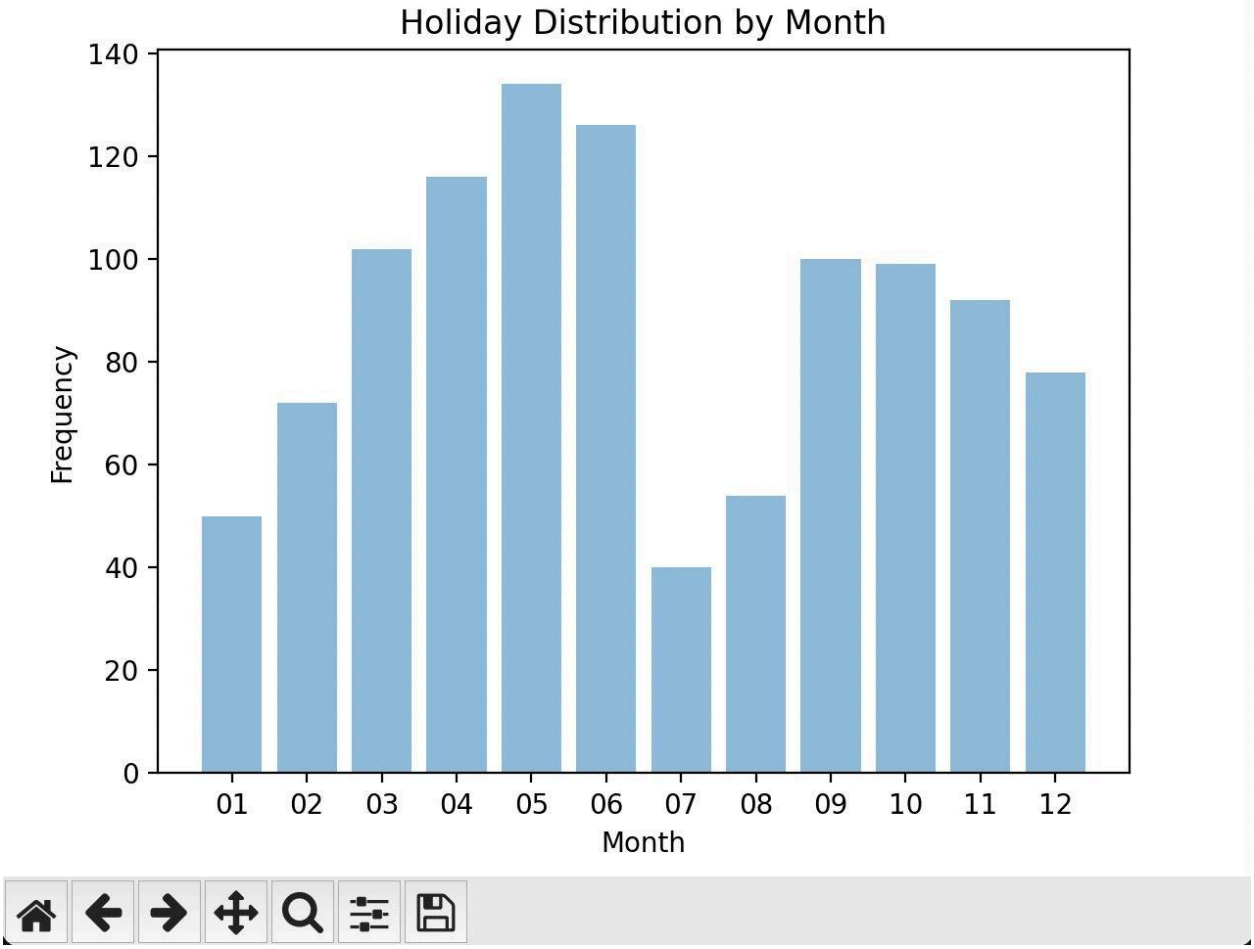
Calculations:

```
path = os.path.dirname(os.path.abspath(__file__))
conn = sqlite3.connect(path + "/" + 'SI_final_project.db')
cursor = conn.cursor()
cursor.execute('''SELECT Ratings.IMDbRating, Ratings.RTRating, Movies.Title
                  FROM Ratings
                  JOIN Movies ON Ratings.ID = Movies.ID
                  LIMIT 25''')
rows = cursor.fetchall()
conn.close()
imdb_ratings = []
imdb_ratings_from_db = [row[0] for row in rows]
for imdb_rating in imdb_ratings_from_db:
    imdb_rating = str(imdb_rating)
    imdb_ratings.append(imdb_rating)
rtratings_from_db = [row[1] for row in rows]
rtratingL= []
for rtrating in rtratings_from_db:
    rtrating = int(rtrating) / 10
```

Visualizations:







Instructions for Running the Code:

1. On data.py, run the file 5 times so that each time the database loads in 25 rows, on the fifth time there will be 119 rows in the Movies table and 77 rows in the Ratings table.
2. On the visualization.py file, run the file and the IMDB vs. Rotten Tomatoes and the Release Dates vs. Movie titles visualizations will appear
3. holidays.py
4. holidays_visualizations.py

Code Documentation:

data.py

- **get_movie_titles():** This function gets the movie titles from the Rotten Tomatoes website.
- **get_rtrating():** This function gets the rating from the Rotten Tomatoes website, which is labeled the “audience score” on the website.
- **get_release_date(movie):** This function uses the OMDb API to get the release date of a movie based on its title.
- **get_imdb_rating(movie):** This function uses the OMDb API to get the IMDb rating of a movie based on its title.
- **create_database_table(conn):** This function creates a table called “Ratings” in the SQL database, planning on storing IMDb ratings, Rotten Tomato ratings, and the movie title.
- **insert_into_table(imdb_rating, rt_rating, conn:** This function inserts the ratings into the table.
- **get_date_id(release_date):** This function converts the release date into the integer ID.
- **count_rows(table_name):** This function counts the number of rows stored in a table of the SQL database.
- **main():** This function gets the movie titles, release dates, rotten tomato ratings, and imdb ratings, it loops through the movie titles to get ratings and release dates and then stores them into the correct table in the SQL database. This function is in charge of executing the overall program.

visualizations.py

- **main():** This main function plots two visualizations. It connects to our database and joins the movie titles, imdb ratings, and rotten tomatoes ratings from the Ratings and Movies tables. The program then takes the Rotten Tomato Ratings and divides them by 10 in order to match the units on the IMDB ratings. It then plots these points in a bar chart. The next visualization it creates comes from the Movies table, where it selects the release dates for each movie and plots this in a scatter plot with a line.

holidays.py

- **get_holidays(api_key, year):** This function grabs holidays from the Calendarific API, the extracted data is returned as a list of tuples containing holiday names and dates.
- **create_holiday_month_table(conn):** This function creates a SQL table named “holiday_month” to store holiday names and their month.

- **count_rows(conn, table_name):** This function counts the number of rows stored in a table of the SQL database.
- **insert_into_holiday_month_table(conn, holidays):** This function inserts holiday data (name and month) into the table stored in the database.
- **main():** This function retrieves holidays for specific years, creates the database table, inserts holiday data into the table, and is in charge of executing the overall program.

holidays_visualizations.py

- **get_month_counts_from_db():** This function gets the counts of holidays within each month from the holiday_month table in the database, and returns a list of tuples with each tuple being the month and counts of holidays within each month.
- **plot_normal_distribution(month_counts):** This function takes in the month_counts list from the previous function and plots a bar chart to visualize the number of holidays within each month.
- **main():** This main function connects to the database and calls the get_month_counts_db and plot_normal_distribution functions.

Documentation of Resources Used:

Office Hours (multiple times), ChatGPT (helped with debugging and creating the normal curve), W3Schools

Problems Faced:

Date	Issue Description	Location of Resource	Result (Did we solve the issue)
04/08	We could not find an api key for weather history data that would let us run the code more than once a day.	Open Weather API Oikolab API Visual Crossing API	We ended up having to change our question, so we did not need weather anymore.
04/13	At first, when we		We fixed our function

	created our tabled, we had over 600 rows, even though the api we were using only had 150 movies.		to loop through and not add duplicates of movies into the tables.
04/17	It was hard to figure out how to get the release dates for the movies, and the dates of the holidays to coincide.		We had to change the ID's on our tables from strings to integers so the movie release dates and the holiday dates could find each other when we used join.
04/18/2023	We struggled to only store 25 items at a time when we would run our files.	Office Hours	Yes we used indices to access our lists, based on advice from IAs and GSIs in office hours.