SI 206 Final Report

Neeran Bari and Victor Schmitt

Link to Github: https://github.com/VicSchmitt/SI206-Final-Project.git

A.

In our initial proposal, we wanted to look at data on Weather.gov and OMDB to analyze how weather affected box office turnout. However, OMDB doesn't have region-specific box office data so we pivoted our project to use the Youtube Data API in tandem with the OMDB API (movie metadata such as box office revenue and rotten tomatoes score). Through these, our project goal is to analyze how online performance correlates with critical reception and box office success. From the OMDB API, we were mainly seeking the average Rotten Tomatoes and revenue from popular movies. From the Youtube Data API, key info included trailers that matched the OMDB movies' names, as well as the corresponding view counts.

B.

We were able to collect metadata on over 100 movies from the OMDB and matched them with hundreds of trailers from the Youtube Data API. We were able to aggregate the individual view counts for each trailer into a total per movie in our calculations. In the end, we had two databases. Movies: which held one entry per movie and its OMDB metadata, and Videos: which had trailers for each of the OMDB movies and their view counts.

C.

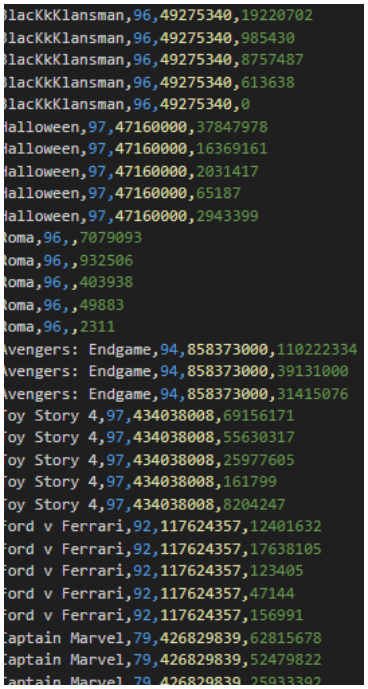
The first issue we faced was that Youtube has a very restrictive API Quota which made testing difficult. Our way around this was to scrape all the necessary data once and use the corresponding cache.json file to run our calculations from. In our initial project review we had a column in the database that marked whether the data came from the cache or an immediate API call. However, to make sure the code will always run locally, the data will all be from the cache and thus, this column was removed to make sure the database did not have duplicate string data.

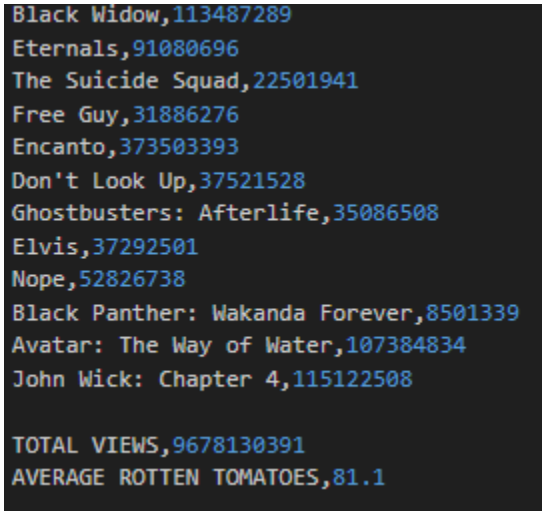
Our original database also did not make proper use of database structure and the movies database and videos database both only had one entry per movie (the videos db had total views for each movie). However, this could just as easily have been one table with total views at the end of each row of the data gathered from OMDB. To use the database properly and still have a shared movie ID, the Videos database was edited to have an entry for each trailer for a certain movie and its corresponding views. Thus, the videos database has multiple trailers with unique titles that have the same movie ID as

an entry in the Movies database. This also meant that the following calculations were altered to account for individual videos instead of total trailer views.

The final issue we faced was that some entries in both databases had incomplete data and had to be ignored.

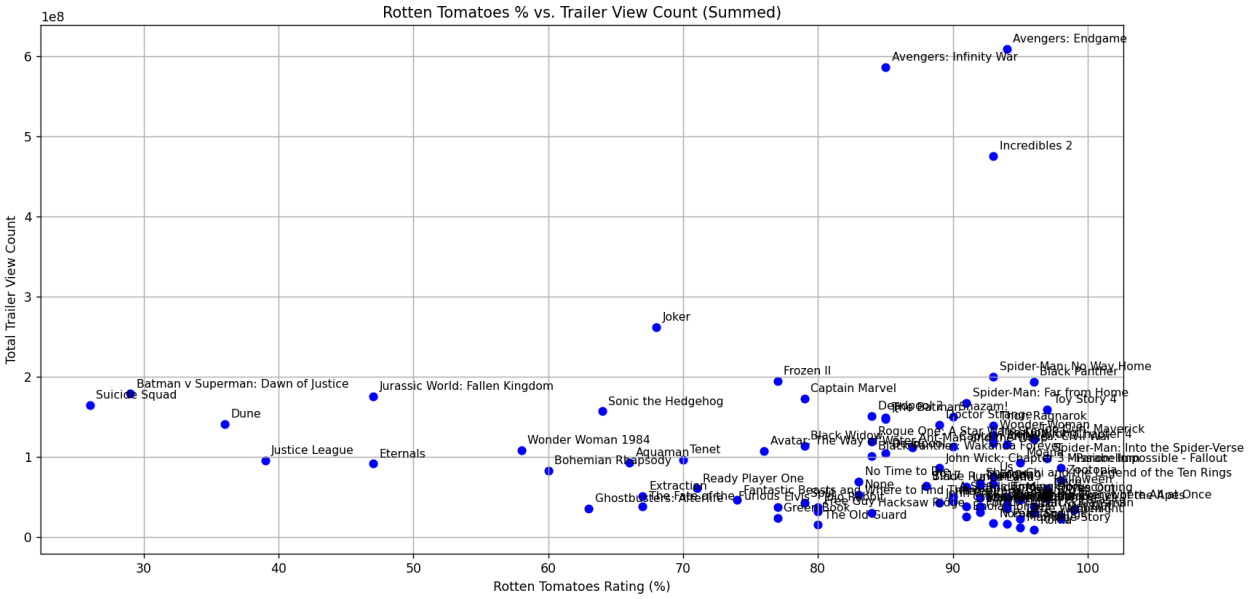
D. Using JOIN, we were able to create a list of movie trailers with its movie metadata on the left hand side and its views as the last column. From there, we were able to calculate the total views per movie, as well as the total trailer views and average Rotten Tomato score across all movies in the database.

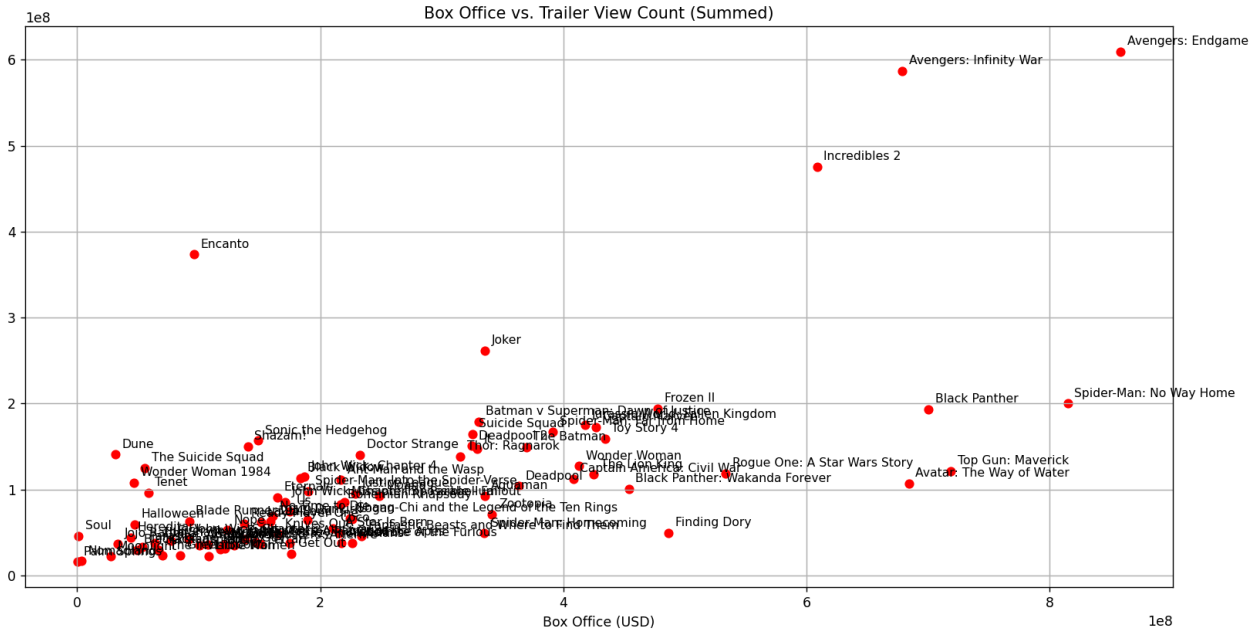




Matplot allowed us to compare trailer views with both critical turnout and box office revenue in a visually appealing manner (shown below).

E.





F. First, one must have the google API client downloaded, the command line operation is as follows:

pip install google-api-python-client pandas matplotlib

Next, to run this code one must simply run main. Using the subprocess library, every run of main will update the database with 25 more items from the cache, then perform the necessary join for calculations and write the calculation data to results.csv. The database and calculation code can be run separately without the visualizations by running

store_data (each run will store 25 items), then select_data, then calculate_data, then write_results.

G. **Documentation:**

`main.py`

- `load_cache()` → Loads JSON cache from file.
  Input: None → **Output**: Sets global cache dicts.

- `save_cache()` → Saves all caches to `cache.json`.
  Input: None → **Output**: Writes file.

- `build_url(movie_title)` → OMDb API URL for a movie.
  Input: str → **Output**: str (URL)

- `get_movie_omdb_data(movie_title)` → Movie metadata from OMDb.
  Input: str → **Output**: dict with Title, Year, RT score, Box Office.

- `search_trailers(movie_title)` → Finds YouTube trailers.
  Input: str → **Output**: list of dicts with video metadata.

- `get_video_stats(video_ids)` → Gets stats for YouTube videos.
  Input: list of str → **Output**: list of dicts with view/like/comment counts.

- `get_movie_viewcount(movie_title)` → Sums trailer views for a movie.
  Input: str → **Output**: int

`store_data.py`

- `load_cache_file()` → Loads and returns cache dict from file.
  Input: None → **Output**: dict

- `create_tables()` → Initializes SQLite schema.
  Input: None → **Output**: None

- `get_existing_titles()` → Returns a set of already stored movie titles.
  Input: None → **Output**: set

- `get_row_counts()` → Returns total counts of movies and trailers.
  Input: None → **Output**: tuple (int, int)

- `store_data_from_cache()` → Writes new data from cache to DB.
  Input: None → **Output**: None

`select_data.py`

- `join_movie_video_data()` → Joins Movies and Videos by foreign key.
  Input: None → **Output**: list of tuples (title, RT, box office, view count)

`calculate_data.py`

- `calculate_stats()` → Aggregates trailer views by movie and averages RT scores.
  Input: None → **Output**: dict with total views, average RT, and inner dict of total views per movie

H. Resources

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4/2/2025	Obtain OMDB Metadata	https://www.omdbapi.com/	Data acquired for free
4/3/2025	Obtain Youtube Trailer Data	https://developers.google.com/youtube/v3	Required going into account and activating, but data was free with a limit each day
4/4/2025	Organize DB by Video	https://docs.python.org/3/library/sqlite3.html	Documentation showed the file path necessary to access my data and return the correct video data instead of the total
4/5/2025	Youtube API Quota Reached	https://github.com/googleapis/google-api-python-client	Google API Client allows api access and cache info so the project could be run using this json
4/10/2025	Display Data	https://matplotlib.org/stable/users/inde	Direct API access made it so this data could compare all data from the two databases at once and

		x.html https://pandas.pydata.org/docs/	was not reliant on the database code.
4/19/2025	Run code as 1 script	Copilot AI	Provided the suggestion of using subprocess library: https://www.geeksforgeeks.org/python-subprocess-module/
4/19/2025	Document Code	ChatGPT 4o	Formatted the documentation for each function in all files with input and output specified