SI 206 Final Project Report

Meera Agrawal, Neha Kotagiri, and Lindsay Nelson
music lovers <3 turned movie lovers <3

APIs used = YouTube API, OMDb API, Box Office Mojo (beautiful soup)

Project Goals

Originally, we wanted to use the Spotify API, the US Covid Tracking API, and the India Covid Tracking API. As our team name states, we are music lovers, so we were eager to look into if there was a correlation between the number of minutes listened to in US vs. India during different Covid periods. After playing around with our original API's, we realized the data we wanted to collect was unattainable.

Therefore, we pivoted our project to focus on another creative medium - movies. We made our research question "Do movies with higher box office sales have higher amounts of YouTube trailer likes/views and higher imdb ratings?". We were curious if higher view/like counts on trailers for movies reflected higher IMDb ratings and overall box office sales. In order to further explore this question, we agreed upon 100 movies from 2019, since we wanted to avoid any confounding variables related to the pandemic. Since YouTube was established in 2005, we decided to choose a specific year of movies so that we could limit confounding variables due to technology differences for older movies. We pulled data from the YouTube API for official movie trailer like and view counts. We extracted data from the OMDb API to acquire IMDb rating. Then, we used BeautifulSoup on the Box Office Mojo by IMDbPro website to obtain box office sales.

Goals that were Achieved

We successfully completed many things during this project. For one, we were able to gather data from all our APIs/Beautiful Soup and use it to create tables in a database, perform calculations about our variables, and create visualizations that help us answer our research question.

Our first visualization shows us that there is a weak positive correlation between IMDb rating and movie runtime (in minutes). Generally higher runtimes lead to slightly better IMDb ratings. This can possibly be explained by the fact that longer movies have more content so audiences enjoy them more. This was also surprising, though, as we believed that audiences do not enjoy longer movies so they would give them lower ratings.

The second visualization displays that there is a positive correlation between box office sales and likes on YouTube trailers. This helps us answer our research question as generally movies with higher likes on their YouTube trailers have higher box office sales in theaters. This makes sense as people who liked a movie trailer are also likely to see it in theaters since they enjoyed the preview.

The third visualization and fourth visualizations show that there is a scattered, weak association (with a slightly positive skew) between movie trailer views and IMDb ratings, and movie trailer likes and IMDb ratings. Though certain movies and trailers gather more hype and attention, which draws people to the movie trailer, it doesn't necessarily mean the movie is good quality and received well by the public. Marketing and PR likely contribute more to movie trailer views and likes than the quality of the movie.

The fifth visualization shows that there is a positive correlation between box office sales and views on YouTube trailers. This helps us answer our research question as generally movies with higher views on their YouTube trailers means that more people are interested in watching the movie once it is released, resulting in higher box office sales.

Overall, there is not a definite strong correlation between box office sales and YouTube trailers views/likes and IMDb ratings, but there is some positive, association.

<u>**Problems Faced**</u>

Unfortunately, we faced many problems during this project. After we realized the data from our original APIs was unattainable, we had to completely switch up the direction of our project, causing us to find new APIs and websites to scrape. Once we settled on using YouTube, IMDb, and OMDb, we discovered that we would have to create a set list of 100 movies in order to obtain the same data from all our sources. This resulted in a lot of manual labor since we had to individually find and code a list of 100 IDs on our sites to match the correct movie. Another issue we had was when we were scraping the Box Office Mojo website: a lot of the information was under the same HTML classes, so it was difficult to pull just the information that we were looking for, which was total gross box office sales.

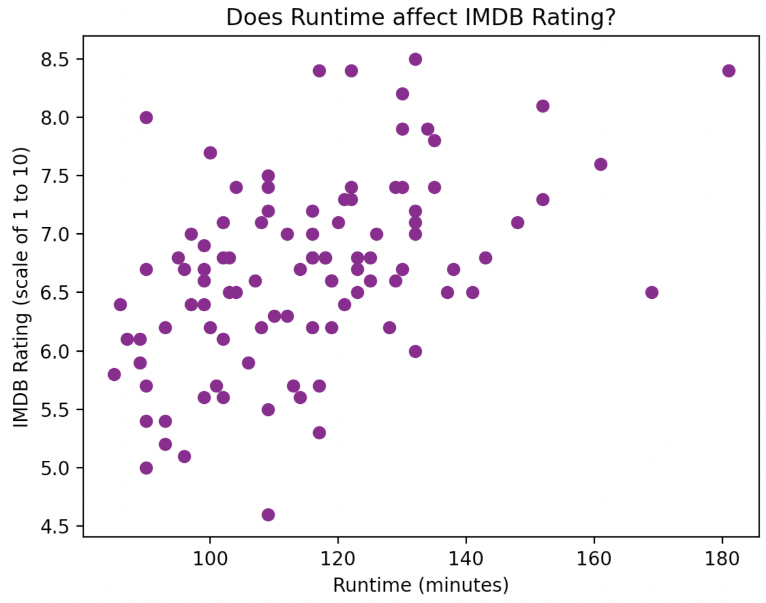
After we had our data and made our tables, our next obstacle was limiting our tables to only insert 25 rows of data at a time. We had to restructure a lot of our code and rewrite some of our functions because we kept running into bugs. This also happened when office hours were over for the semester, so we had to solve the issue using the internet and our knowledge.

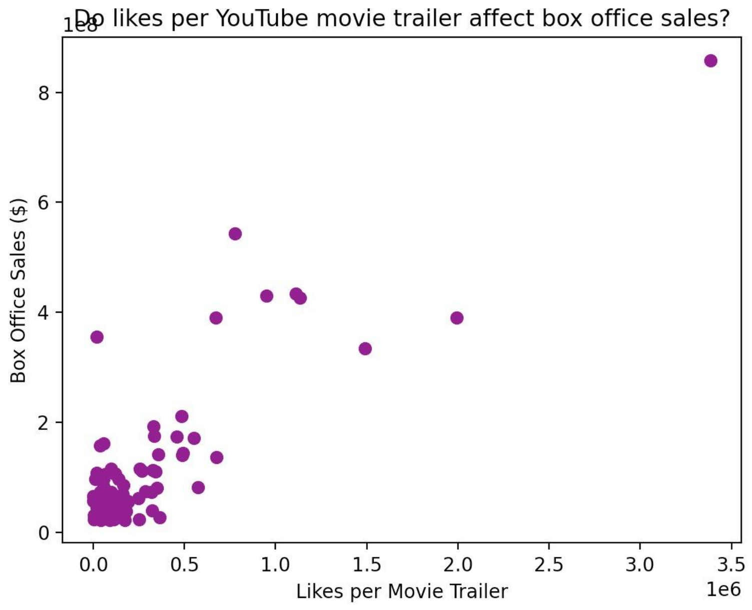
Our next problem was figuring out how to get data from each other's sources and writing our calculations to a single file. Once we understood how JOIN worked, we selected data from each other's files, performed calculations, and wrote them to a single csv.

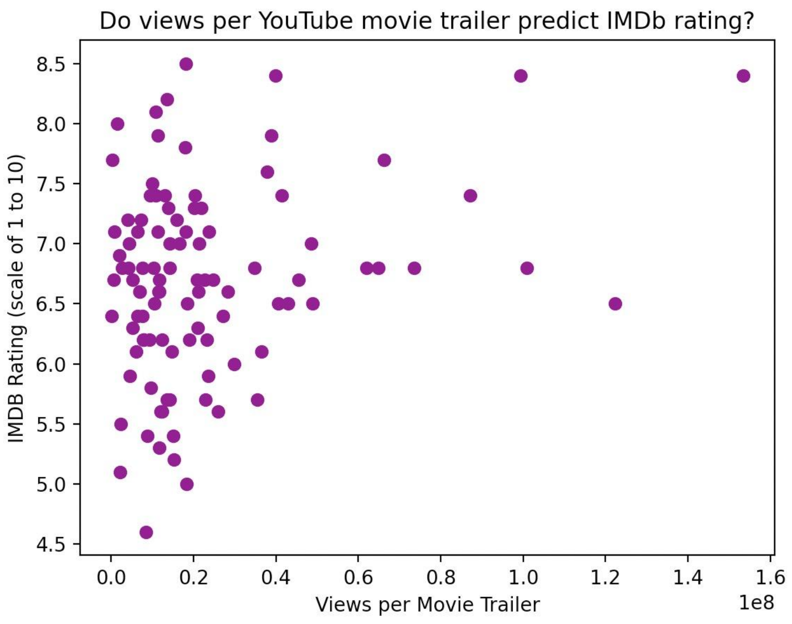
<u>**File that Contains Calculations from the Data in the Database**</u>

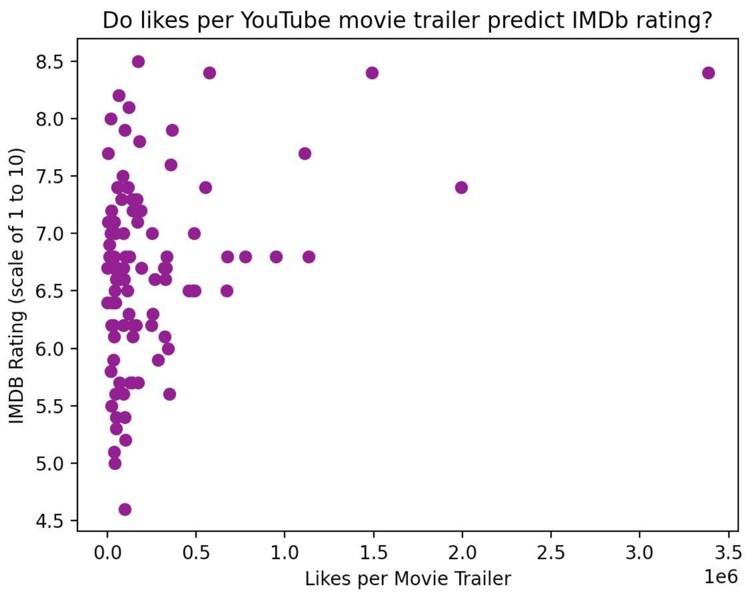
Please look at "final_calculations.csv" in the GitHub folder to access the file that contains calculations from data in the database.

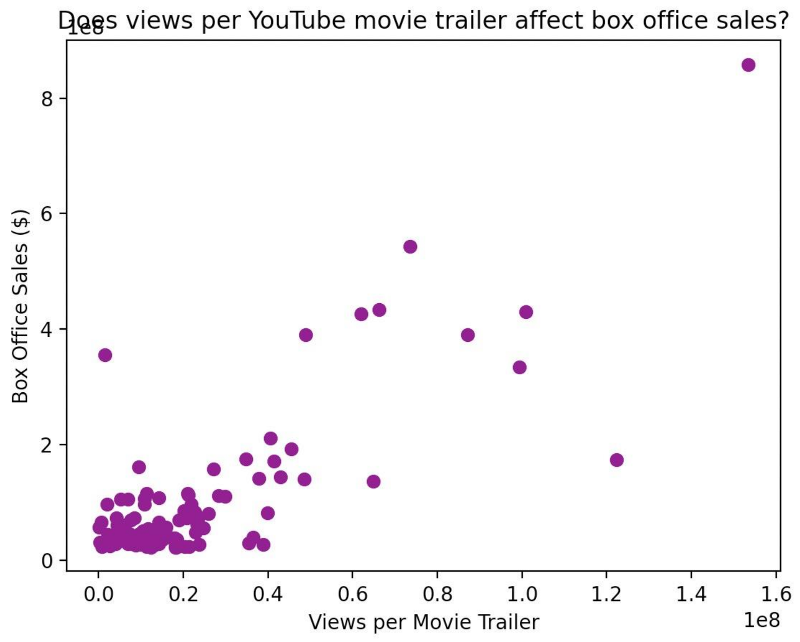
Visualizations






Do views per YouTube movie trailer predict IMDb rating?
IMDB Rating (scale of 1 to 10)
Views per Movie Trailer
1e8
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6


Do likes per YouTube movie trailer predict IMDb rating?
IMDB Rating (scale of 1 to 10)
Likes per Movie Trailer
1e6
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5



Instructions for Running Code

1) Download all files from GitHub and unzip the folder.
2) Run “boxofficemojo.py” and open up the created database (“movies.db”) in DB browser. This should insert the first 25 rows of data from “boxofficemojo.py”.
3) Run “boxofficemojo.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “boxofficemojo.py” (total = 50 rows).
4) Run “boxofficemojo.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “boxofficemojo.py” (total = 75 rows).
5) Run “boxofficemojo.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “boxofficemojo.py” (total = 100 rows).
6) Run “omdb.py” and refresh “movies.db”. This should insert the first 25 rows of data from “omdb.py”.
7) Run “omdb.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “omdb.py” (total = 50 rows).
8) Run “omdb.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “omdb.py” (total = 75 rows).
9) Run “omdb.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “omdb.py” (total = 100 rows).
10) Run “youtube.py” and refresh “movies.db”. This should insert the first 25 rows of data from “youtube.py”.

If the terminal gives you an error “write.writerow([title_names[i], ratio[i], diff[i], ratio2[i]]). IndexError: list index out of range”, ignore that. That error will be resolved once “youtube.py” is run 4 times.

11) Run “youtube.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “youtube.py” (total = 50). Ignore the IndexError once again.
12) Run “youtube.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “youtube.py” (total = 75). Ignore the IndexError once again.
13) Run “youtube.py” again and refresh “movies.db”. This should insert the next 25 rows of data from “youtube.py” (total = 100).
14) Open “final_calculations.csv” to view all the calculations .
15) Open visualizations to see graphs and charts.

Documentation for Functions

Name of Function	**Input**	**Output**
open_database	**db_name**: name of database	**cur, conn** for SQL tables
boxofficeurl	**url**: boxofficemojo url from 2019	Returns the **url**
getmovielist	**url**: box office mojo url from 2019	Returns **finalnameslist**: a list of 100 movies from 2019
getboxlist	**url**: boxofficemojo url from 2019	Returns **boxlist**: a list of total box office gross sales for the 100 movies in getmovielist()
make_boxoffice_table	**boxlist** (list of box office sales), **cur, conn**	Table titled **“BoxOfficeData”** which includes the id and the total box office sales
make_names_table	**nameslist** (list of movie titles), **cur, conn**	Table titled **“MovieTitlesData”**, which includes the id and the movie titles
get_data_url	**movie** (name of movie)	**url** for the inputted movie name
get_title	**movie** (name of movie)	**title** (the title of the movie)

Name of Function	**Input**	**Output**
get_imdb_rating	**movie** (name of movie)	**rating** (rating of movie on scale of 1-10)
get_runtime	**movie** (name of movie)	**runtime (**runtime of movie in minutes)
get_runtime_differences	**movie_titles** (list of movie titles)	**list_of_differences (**list of differences of average runtimes between all the movies)
create_imdb_table	**movie_titles** (list of movie titles), **cur, conn**	Table titled "movie_data" which includes the id, imdb rating, and movie runtime
get_request_url	**string** (YouTube ID of movie trailer for a movie)	**url** (API base url for that specific movie trailer)
get_view_count	**string** (YouTube ID of movie trailer for a movie)	**viewCountList** (list of view counts for all 100 movies)
get_like_count	**string** (YouTube ID of movie trailer for a movie)	**likeCountList** (list of like counts for all 100 movies)
get_movie_title	**string** (YouTube ID of movie trailer for a movie)	**title_list (**list of 100 movie title names)
make_ratings_table	**viewlist** (list of view counts)**,** **likelist** (list of like counts)**,** **cur, conn**	Table titled **"Ratings"** which includes id, views, and likes
join_ratings_and_boxOffice	**cur, conn**	Joins **Ratings** table and **BoxOfficeData** table
join_ratings_and_omdb	**cur, conn**	Joins **Ratings** table and **movie_data** table
join_titles_and_ratings	**cur, conn**	Joins **Ratings** table and **MovieTitlesData** table
like_to_view_calculations	Calls **get_view_count** and **get_like_count**	returns **calc,** the ratio of likes to views for each movie
imdb_to_boxOffice(cur, conn)	**cur, conn**	Calculates the ratio between

Name of Function	**Input**	**Output**
		imdb rating and box office sales. It also multiplies the calculation by 10^7 so the number is easier to read.
titles	**cur, conn**	list_of_titles (list of movie titles)
list_of_box_office_sales	**cur, conn**	**sales_list** (list of box office sales)
list_of_views	Calls **get_view_count**	**view_list** (list of trailer views for each movie)
list_of_likes	Calls **get_like_count**	**like_list (**list of likes for each movie)
list_of_imdb_ratings	Calls **join_ratings_and_omdb**	**imdb_rating_list** (list of imdb ratings for each movie)
write_calculations	**filename, cur, conn**	CSV file titled “final calculations” with calculations for like to view ratio, differences from average runtime, and IMDB Rating to Box Office Sales Ratios
view_vs_boxoffice	**Cur, conn**	Plots views per movie trailer vs. box office sales
like_vs_boxoffice	**Cur, conn**	Plots likes per movie trailer vs. box office sales
view_vs_rating	**Cur, conn**	Plots views per movie trailer vs. IMDb rating
like_vs_rating	**Cur, conn**	Plots likes per movie trailer vs. IMDb rating
make_runtime_rating	**movie_titles**	Plots runtime vs. IMDb rating

<u>Resources Used:</u>

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/2/2022	Trying to understand how to get an API key from Youtube to access data	https://medium.com/swlh/how-to-get-youtubes-api-key-7c28b59b1154	It provided the steps to obtain a Youtube API key
12/2/2022	Trying to understand how to access like and view count from Youtube videos	https://developers.google.com/youtube/v3/docs/search/list?apix=true&apix_params=%7B%22part%22%3A%5B%22snippet%22%5D%2C%22maxResults%22%3A25%2C%22q%22%3A%22official%20movie%20trailers%22%7D	It provided a developer tool that helped create a base url to access data. The developer tool was confusing to use, though, so we had to go to office hours to figure out how to use it
12/5/2022	Pulling the correct information and tags using from Box Office Mojo using BeautifulSoup	https://medium.com/@avivamazurek/basics-of-web-scrapping-how-to-scrape-data-off-of-box-office-mojo-5ec6c22dcca6	Helped point us to the specific HTML tags to access the names and box office numbers
12/9/2022	Difficulty limiting how much data to store from an API to 25 or fewer items each time you execute code	https://www.sqlitetutorial.net/sqlite-limit/	It did not the fix the issue, but we were able to figure out another resolution using a count increment
12/10/2022	Difficulty joining tables using SQL	https://www.w3schools.com/sql/sql_join.asp	It provided information and examples on how to join tables from the same database using SQL
12/11/2022	Uncertain how to make	https://www.w3schools.	Created a scatterplot

	a scatter plot with matplotlib	**com/python/matplotlib_scatter.asp**	showing runtime vs. imdb rating for 100 movies