Final Project Report

API Explorers

Lia Du, Eric Dorman, Ismail Hazime

SI 206: Data-Oriented Programming

December 16, 2024

GitHub Repository

Goals and Initial APIs

Our main goal for the project was software to easily see movie data. To be more specific, we planned to incorporate the use of various APIs such as *OMDB*, *Open Movie Database,* and *Simkl* to give us the maximum amount of data to work with for our calculations and final output. We wanted to analyze movie trends; specifically, if there was a correlation between movie runtime, finances, genre, user and critic ratings, and year released.

We aimed to determine the following:

1. Correlation between genre and rating
2. Correlation between genre and runtime
3. Correlation between genre and revenue
4. Correlation between ratings and revenue
5. Correlation between ratings across three APIs
6. Overall movie trends and statistics
    - Worst and top performing movie in a certain year
    - Worst and best rated movie in a certain year
    - Financial trends of movies

Goals and Final APIs

Of these core goals we were aiming for, we managed to achieve most of them, though not everything went as planned. For example, instead of our initial plan to use *Simkl,* we ended up instead opting for a different approach of using *WatchMode*. Simkl was a bit confusing and did not provide an API key, we moved on to TVDb but it had the same problem in which it did not provide an API key and we did not understand what it was providing or how to utilize it. Ultimately, we changed to WatchMode.

We used three APIs:

1. WatchMode: Collected movie title, user rating, and critic rating.
2. OMDB: Collected movie title, release year, genre, runtime, and box office revenue.
3. TMDB: Collected movie title, release date, revenue, budget, rating, user votes, and user popularity.

Of our core goals, we were able to determine the following:

1. Correlation between genre and rating
2. Correlation between genre and runtime
3. Correlation between genre and revenue
4. Correlation between ratings and revenue
5. Correlation between ratings across three APIs

Challenges

In order to reach our final product, we encountered countless issues, ranging from smaller issues like syntax errors to much larger ones, like not knowing how to do something such as implementing the visualizations. Initially, we realized that the APIs we had initially chosen were not effectively and efficiently suited for our goal in mind. After we realized this, we could not find a suitable API in the “public-apis” list provided to us, so we had to scour the internet in search for suitable APIs.

Additionally, when we found WatchMode API, it was difficult to use it consistently as it had a 1000 limit per user, forcing us to make a new account and request a new API key every couple of hours. Another problem we encountered was movies having multiple genres. This caused issues when creating visualizations as it would store (for example) a sci-fi and action movie’s genre as “Sci-Fi, Action” as a whole. After splicing the genres, we realized that it stored them as two separate entries (ex: “Movie 1 is Sci-Fi” and “Movie 1 is Action”).

Moreover, we simply had trouble achieving the “standards” in the rubric — especially processing less than 25 a time. This was more of a matter of learning, accessing resources, and trying again.

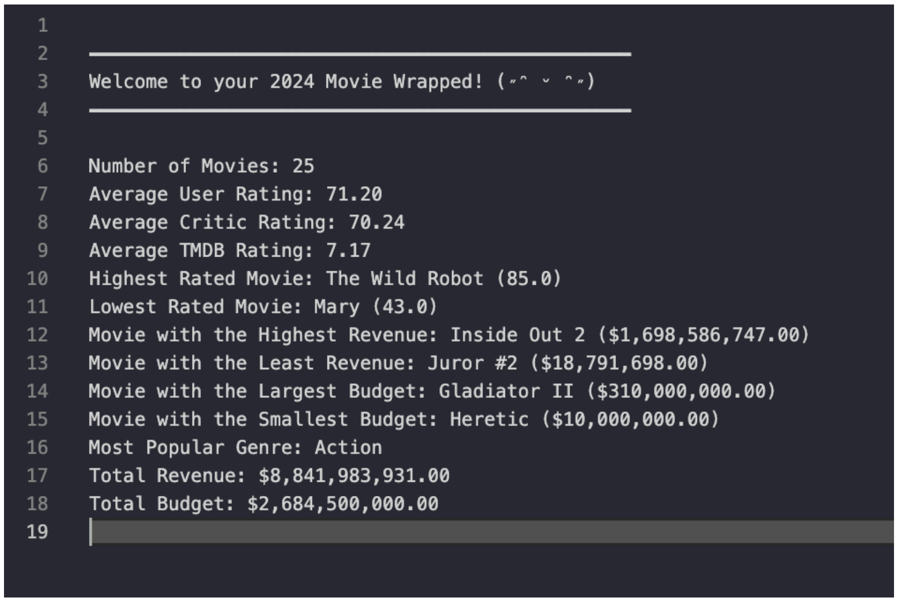
Calculations

For our calculations, we created a 2024 Movie Wrapped (similar to Spotify Wrapped) with the following elements:

1. Number of Movies
2. Average User Rating
3. Average Critic Rating
4. Average TMDB Rating
5. Highest Rated Movie

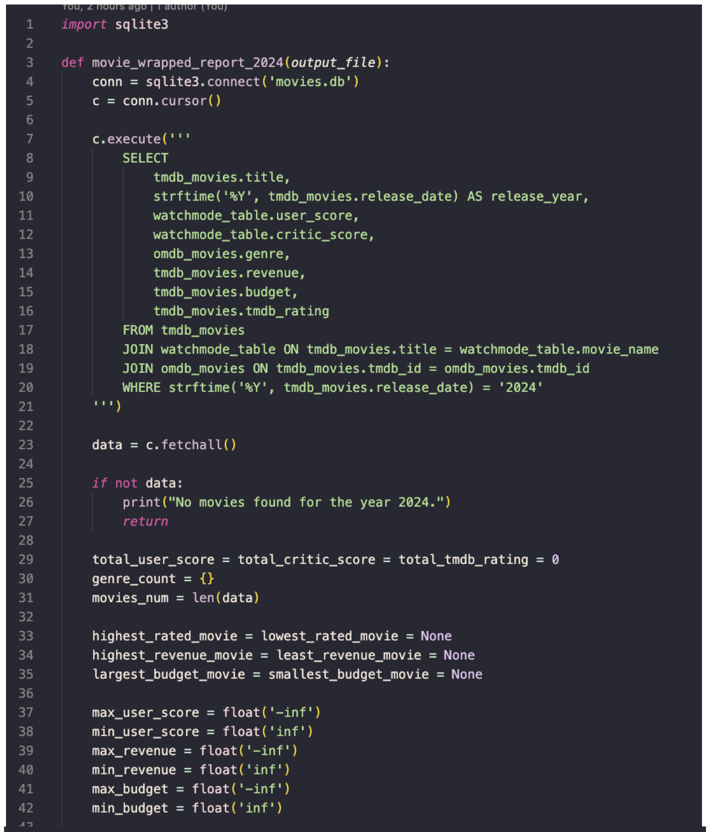
6. Lowest Rated Movie
7. Movie with the Highest Revenue
8. Movie with the Least Revenue
9. Movie with the Largest Budget
10. Movie with the Smallest Budget
11. Most Popular Genre
12. Number of movies with user score above 70
13. Total Revenue
14. Total Budget

Running the calculations file will result in *wrapped.txt:*

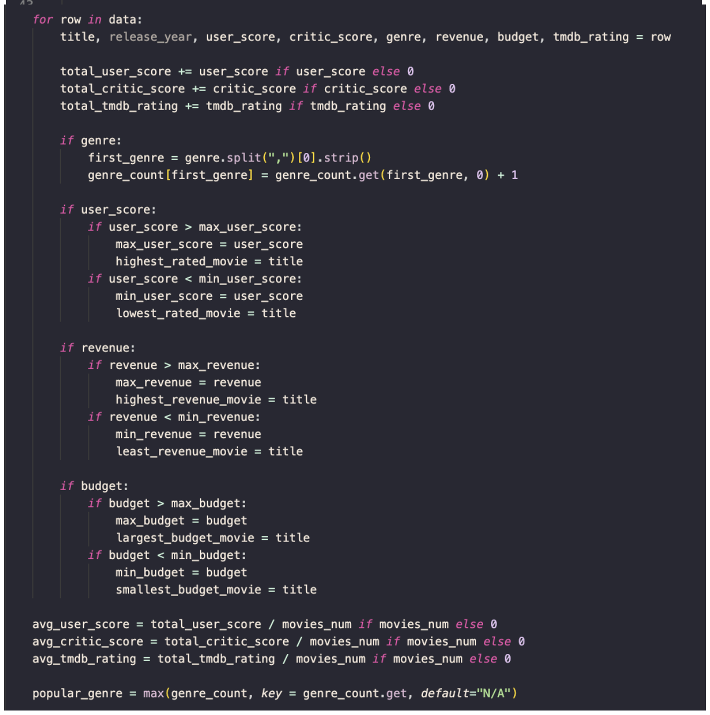


```

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
Welcome to your 2024 Movie Wrapped! (˶ˆ ˘ ˆ˶)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Number of Movies: 25
Average User Rating: 71.20
Average Critic Rating: 70.24
Average TMDB Rating: 7.17
Highest Rated Movie: The Wild Robot (85.0)
Lowest Rated Movie: Mary (43.0)
Movie with the Highest Revenue: Inside Out 2 ($1,698,586,747.00)
Movie with the Least Revenue: Juror #2 ($18,791,698.00)
Movie with the Largest Budget: Gladiator II ($310,000,000.00)
Movie with the Smallest Budget: Heretic ($10,000,000.00)
Most Popular Genre: Action
Total Revenue: $8,841,983,931.00
Total Budget: $2,684,500,000.00

```

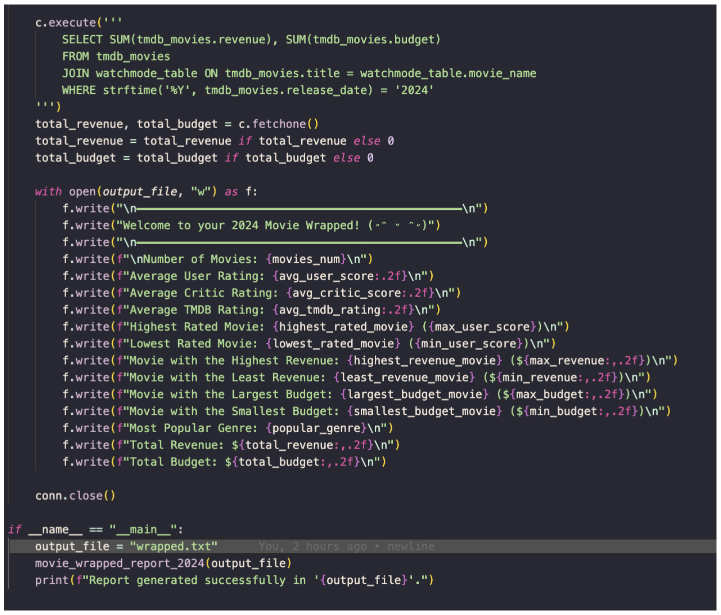
Our calculation file looked like the following:



```
You, 2 hours ago | 1 author (You)
import sqlite3

def movie_wrapped_report_2024(output_file):
    conn = sqlite3.connect('movies.db')
    c = conn.cursor()

    c.execute('''
        SELECT
            tmdb_movies.title,
            strftime('%Y', tmdb_movies.release_date) AS release_year,
            watchmode_table.user_score,
            watchmode_table.critic_score,
            omdb_movies.genre,
            tmdb_movies.revenue,
            tmdb_movies.budget,
            tmdb_movies.tmdb_rating
        FROM tmdb_movies
        JOIN watchmode_table ON tmdb_movies.title = watchmode_table.movie_name
        JOIN omdb_movies ON tmdb_movies.tmdb_id = omdb_movies.tmdb_id
        WHERE strftime('%Y', tmdb_movies.release_date) = '2024'
    ''')

    data = c.fetchall()

    if not data:
        print("No movies found for the year 2024.")
        return

    total_user_score = total_critic_score = total_tmdb_rating = 0
    genre_count = {}
    movies_num = len(data)

    highest_rated_movie = lowest_rated_movie = None
    highest_revenue_movie = least_revenue_movie = None
    largest_budget_movie = smallest_budget_movie = None

    max_user_score = float('-inf')
    min_user_score = float('inf')
    max_revenue = float('-inf')
    min_revenue = float('inf')
    max_budget = float('-inf')
    min_budget = float('inf')
```

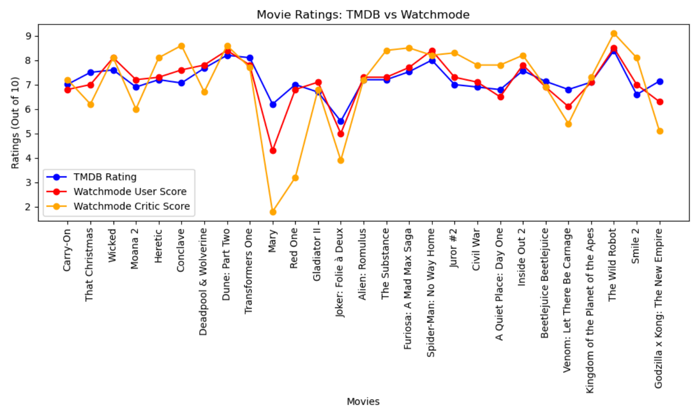
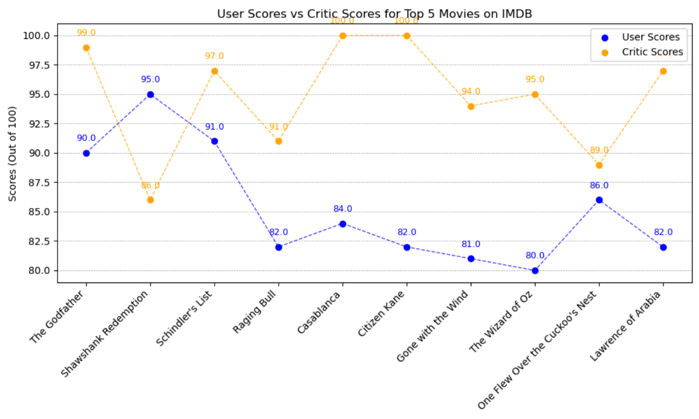


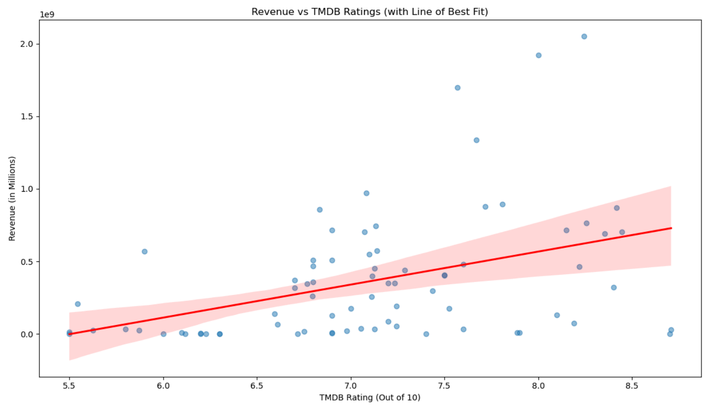
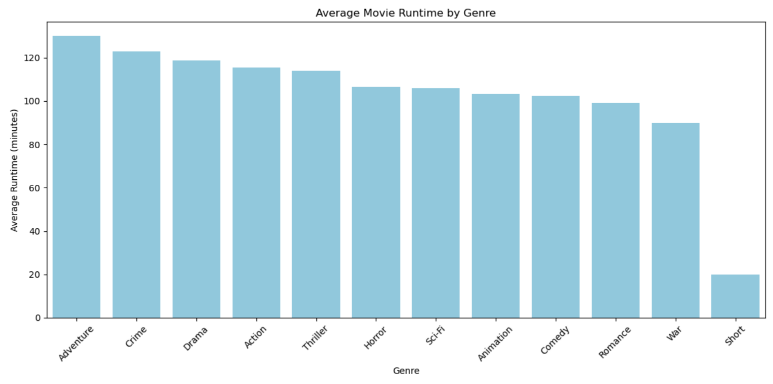
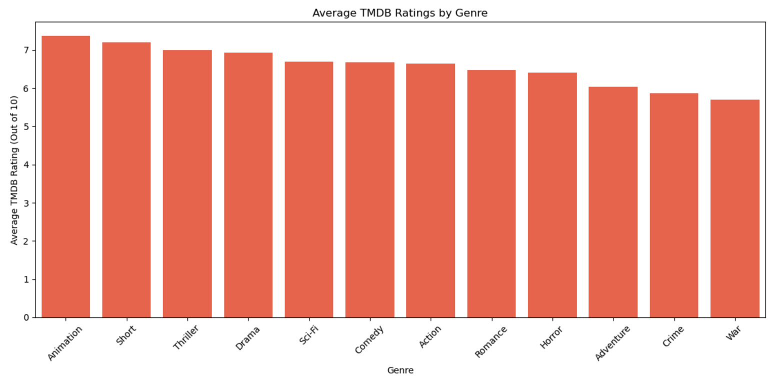
```
for row in data:
    title, release_year, user_score, critic_score, genre, revenue, budget, tmdb_rating = row

    total_user_score += user_score if user_score else 0
    total_critic_score += critic_score if critic_score else 0
    total_tmdb_rating += tmdb_rating if tmdb_rating else 0

    if genre:
        first_genre = genre.split(",")[0].strip()
        genre_count[first_genre] = genre_count.get(first_genre, 0) + 1

    if user_score:
        if user_score > max_user_score:
            max_user_score = user_score
            highest_rated_movie = title
        if user_score < min_user_score:
            min_user_score = user_score
            lowest_rated_movie = title

    if revenue:
        if revenue > max_revenue:
            max_revenue = revenue
            highest_revenue_movie = title
        if revenue < min_revenue:
            min_revenue = revenue
            least_revenue_movie = title

    if budget:
        if budget > max_budget:
            max_budget = budget
            largest_budget_movie = title
        if budget < min_budget:
            min_budget = budget
            smallest_budget_movie = title

avg_user_score = total_user_score / movies_num if movies_num else 0
avg_critic_score = total_critic_score / movies_num if movies_num else 0
avg_tmdb_rating = total_tmdb_rating / movies_num if movies_num else 0

popular_genre = max(genre_count, key = genre_count.get, default="N/A")
```

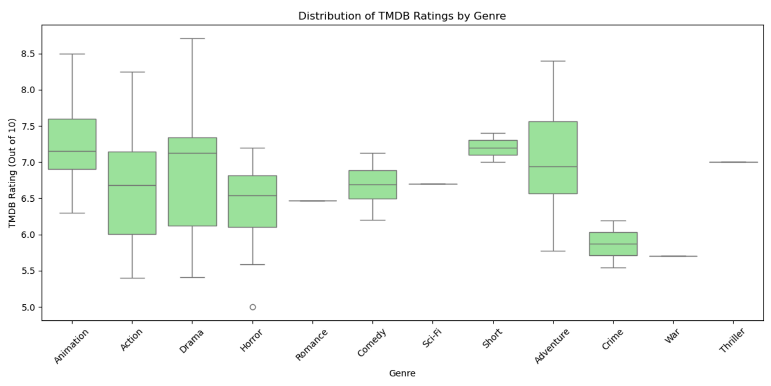
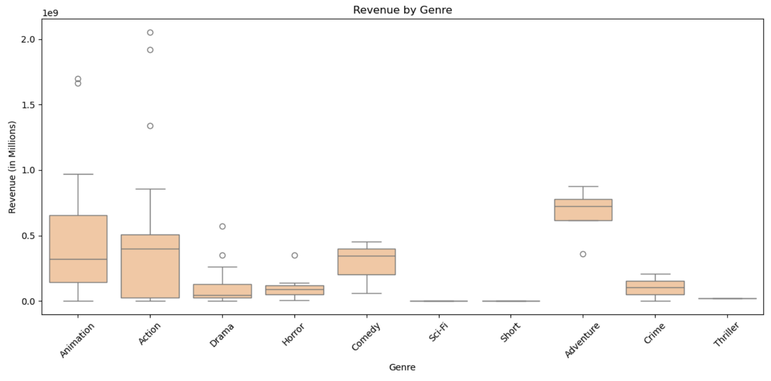


```
    c.execute('''
        SELECT SUM(tmdb_movies.revenue), SUM(tmdb_movies.budget)
        FROM tmdb_movies
        JOIN watchmode_table ON tmdb_movies.title = watchmode_table.movie_name
        WHERE strftime('%Y', tmdb_movies.release_date) = '2024'
    ''')
    total_revenue, total_budget = c.fetchone()
    total_revenue = total_revenue if total_revenue else 0
    total_budget = total_budget if total_budget else 0

    with open(output_file, "w") as f:
        f.write("\n────────────────────────────────────────────────\n")
        f.write("Welcome to your 2024 Movie Wrapped! (˶ˆ ˘ ˆ˶)")
        f.write("\n────────────────────────────────────────────────\n")
        f.write(f"\nNumber of Movies: {movies_num}\n")
        f.write(f"Average User Rating: {avg_user_score:.2f}\n")
        f.write(f"Average Critic Rating: {avg_critic_score:.2f}\n")
        f.write(f"Average TMDB Rating: {avg_tmdb_rating:.2f}\n")
        f.write(f"Highest Rated Movie: {highest_rated_movie} ({max_user_score})\n")
        f.write(f"Lowest Rated Movie: {lowest_rated_movie} ({min_user_score})\n")
        f.write(f"Movie with the Highest Revenue: {highest_revenue_movie} (${max_revenue:,.2f})\n")
        f.write(f"Movie with the Least Revenue: {least_revenue_movie} (${min_revenue:,.2f})\n")
        f.write(f"Movie with the Largest Budget: {largest_budget_movie} (${max_budget:,.2f})\n")
        f.write(f"Movie with the Smallest Budget: {smallest_budget_movie} (${min_budget:,.2f})\n")
        f.write(f"Most Popular Genre: {popular_genre}\n")
        f.write(f"Total Revenue: ${total_revenue:,.2f}\n")
        f.write(f"Total Budget: ${total_budget:,.2f}\n")

    conn.close()

if __name__ == "__main__":
    output_file = "wrapped.txt"      You, 2 hours ago • newline
    movie_wrapped_report_2024(output_file)
    print(f"Report generated successfully in '{output_file}'.")
```

Visualizations

	This graph illustrates the relationship between TMDB ratings, user ratings, and critic ratings. The trends between each category are similar, with “Mary” rating very low and “The Wild Robot” rating high.
	This graph illustrates the trend between user score and critic ratings for the top 10 movies of all time on IMDB. It shows how for all films in the top 10, critics almost always had a much higher score than the users. This divide increases the further down the list you go.

	This visualization depicts the relationship between the movie genre and revenue using a scatter plot. Overall, there is a positive correlation between ratings and revenue.
	This visualization depicts the relationship between the movie genre and runtime. These demonstrated that crime and adventure movies tend to have the longest runtime and short films and documentaries tend to have the shortest runtime.
	This visualization depicts the relationship between the movie genre and TMDB user ratings. Animation and comedy tended to have the highest user ratings. Crime and war tended to have the lowest.

	This visualization depicts the relationship between the movie genre and TMDB user ratings using a box plot.
	This visualization depicts the relationship between the movie genre and revenue using a box plot. Action and animation movies both have outliers with high revenues. Sci-Fi, thriller, and short films also had the least variability in terms of revenue made as well.

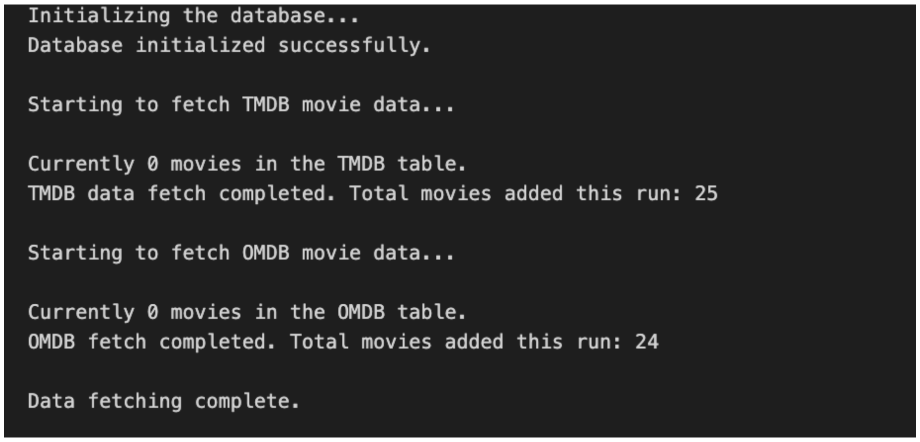
Instructions

Ensure that all external Python packages are installed prior to running the code:

1. requests
2. matplotlib
3. numpy
4. seaborn
5. pandas
6. tmdbv3api

Before you run, also ensure that “movies.db” is deleted.

Run collectmoviedata.py six times: The *omdb_movies* and *tmdb_movies* tables will fill up. As they are being processed in batches at a time, it will take a while to fill up. You should see messages similar to the following, depicting the number of movies added during a run, as well as how many movies you are starting with.



```
Initializing the database...
Database initialized successfully.

Starting to fetch TMDB movie data...

Currently 0 movies in the TMDB table.
TMDB data fetch completed. Total movies added this run: 25

Starting to fetch OMDB movie data...

Currently 0 movies in the OMDB table.
OMDB fetch completed. Total movies added this run: 24

Data fetching complete.
```

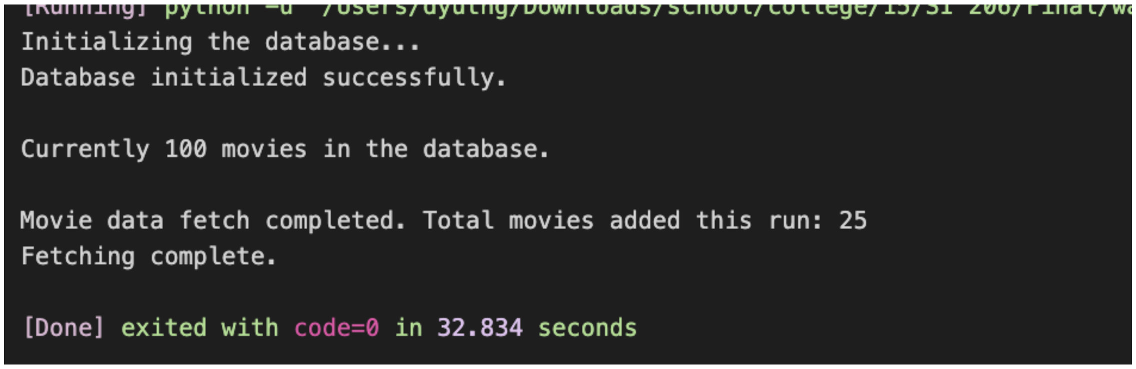
Run visualizations.py one time: After it runs, you will see five graphs:

1. rating_vs_genre_bar.png
2. rating_vs_genre.png
3. revenue_vs_genre_box.png
4. revenue_vs_rating_with_fit.png
5. runtime_vs_genre.png

These graphs will be saved locally on your computer. Closing out the first figure will result in the next figure opening, and so on.

Run visual_scorecompare.py one time: After it runs, you will see the graph "ratings_tmdbvswatchmode.png". This graph will be saved locally on your computer.

Run watchmode_database.py five times: The *watchmode_tabl*e table will fill up over time. As it is being processed in 25 at a time, it will take a while to fill up. You should see messages similar to the following, depicting the number of movies added during a run, as well as how many movies you are starting with.



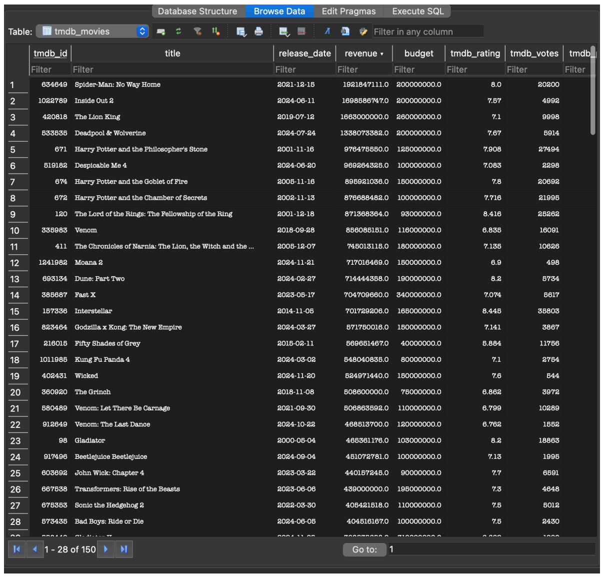
```
[Running] python -u "/Users/ayuting/Downloads/school/college/15/SI 206/Final/wa
Initializing the database...
Database initialized successfully.

Currently 100 movies in the database.

Movie data fetch completed. Total movies added this run: 25
Fetching complete.

[Done] exited with code=0 in 32.834 seconds
```

If you get the error message saying there are too many requests, this is because you have reached your limit with WatchMode API. Under WATCHMODE_API, there are spare API Keys in comments that can be used in order for the file to run. If you exceed the limit, you can request an API key here.

Run watchmodetest.py one time: After running this file, it retrieves the data from the top 10 movies in IMDB, then plots it to a graph that is shown on screen when the code completes.

Run calculations.py one time: After you run this file, the calculations should be on a separate txt file “final_movie_report.txt.”

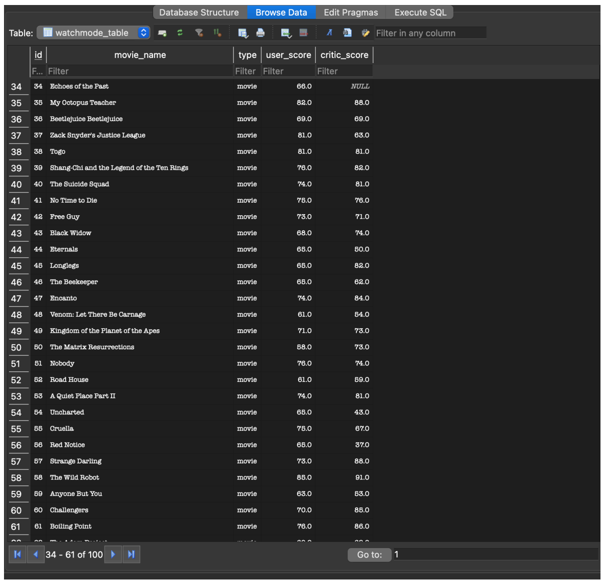
Database and Tables

tmdb_movies From TMDb API. Processes 25 at a time.	

	tmdb_id	title	release_date	revenue	budget	tmdb_rating	tmdb_votes	tmdb_
1	634649	Spider-Man: No Way Home	2021-12-15	1921847111.0	200000000.0	8.0	20200	
2	1022789	Inside Out 2	2024-06-11	1698586747.0	200000000.0	7.57	4992	
3	420818	The Lion King	2019-07-12	1663000000.0	260000000.0	7.1	9998	
4	533535	Deadpool & Wolverine	2024-07-24	1338073382.0	200000000.0	7.67	5914	
5	671	Harry Potter and the Philosopher's Stone	2001-11-16	976475550.0	125000000.0	7.908	27494	
6	519182	Despicable Me 4	2024-06-20	969264325.0	100000000.0	7.083	2298	
7	674	Harry Potter and the Goblet of Fire	2005-11-16	895921036.0	150000000.0	7.8	20692	
8	672	Harry Potter and the Chamber of Secrets	2002-11-13	876688482.0	100000000.0	7.716	21995	
9	120	The Lord of the Rings: The Fellowship of the Ring	2001-12-18	871368364.0	93000000.0	8.416	25262	
10	335983	Venom	2018-09-28	856085151.0	116000000.0	6.835	16091	
11	411	The Chronicles of Narnia: The Lion, the Witch and the ...	2005-12-07	745013115.0	180000000.0	7.135	10626	
12	1241982	Moana 2	2024-11-21	717016469.0	150000000.0	6.9	498	
13	693134	Dune: Part Two	2024-02-27	714444358.0	190000000.0	8.2	5734	
14	385687	Fast X	2023-05-17	704709660.0	340000000.0	7.074	5617	
15	157336	Interstellar	2014-11-05	701729206.0	165000000.0	8.445	35803	
16	823464	Godzilla x Kong: The New Empire	2024-03-27	571750016.0	150000000.0	7.141	3867	
17	216015	Fifty Shades of Grey	2015-02-11	569651467.0	40000000.0	5.884	11756	
18	1011985	Kung Fu Panda 4	2024-03-02	548040835.0	80000000.0	7.1	2754	
19	402431	Wicked	2024-11-20	524971440.0	150000000.0	7.6	544	
20	360920	The Grinch	2018-11-08	508600000.0	75000000.0	6.862	3972	
21	580489	Venom: Let There Be Carnage	2021-09-30	506863592.0	110000000.0	6.799	10289	
22	912649	Venom: The Last Dance	2024-10-22	468513700.0	120000000.0	6.762	1552	
23	98	Gladiator	2000-05-04	465361176.0	103000000.0	8.2	18863	
24	917496	Beetlejuice Beetlejuice	2024-09-04	451072781.0	100000000.0	7.13	1995	
25	603692	John Wick: Chapter 4	2023-03-22	440157245.0	90000000.0	7.7	6591	
26	667538	Transformers: Rise of the Beasts	2023-06-06	439000000.0	195000000.0	7.3	4648	
27	675353	Sonic the Hedgehog 2	2022-03-30	405421518.0	110000000.0	7.5	5012	
28	573435	Bad Boys: Ride or Die	2024-06-05	404516167.0	100000000.0	7.5	2430	

watchmode_table

From WatchMode API. Processes 25 at a time.

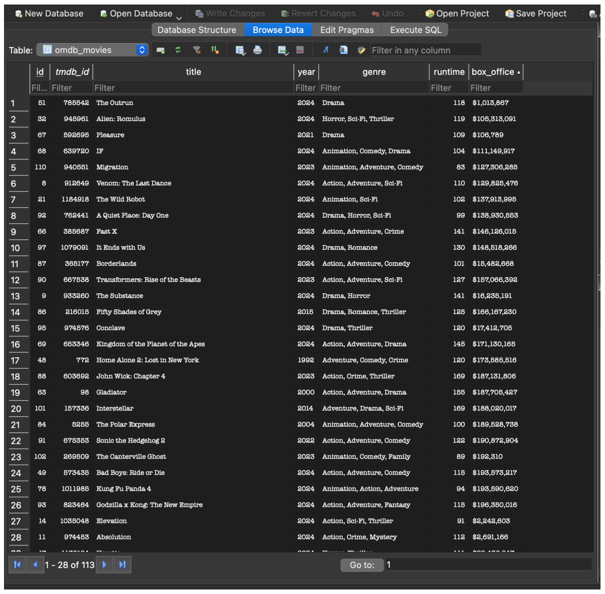


	id	movie_name	type	user_score	critic_score
34	34	Echoes of the Past	movie	66.0	NULL
35	35	My Octopus Teacher	movie	82.0	88.0
36	36	Beetlejuice Beetlejuice	movie	69.0	69.0
37	37	Zack Snyder's Justice League	movie	81.0	63.0
38	38	Togo	movie	81.0	81.0
39	39	Shang-Chi and the Legend of the Ten Rings	movie	76.0	82.0
40	40	The Suicide Squad	movie	74.0	81.0
41	41	No Time to Die	movie	75.0	76.0
42	42	Free Guy	movie	73.0	71.0
43	43	Black Widow	movie	68.0	74.0
44	44	Eternals	movie	65.0	50.0
45	45	Longlegs	movie	65.0	82.0
46	46	The Beekeeper	movie	65.0	62.0
47	47	Encanto	movie	74.0	84.0
48	48	Venom: Let There Be Carnage	movie	61.0	54.0
49	49	Kingdom of the Planet of the Apes	movie	71.0	73.0
50	50	The Matrix Resurrections	movie	58.0	73.0
51	51	Nobody	movie	76.0	74.0
52	52	Road House	movie	61.0	59.0
53	53	A Quiet Place Part II	movie	74.0	81.0
54	54	Uncharted	movie	65.0	43.0
55	55	Cruella	movie	75.0	67.0
56	56	Red Notice	movie	65.0	37.0
57	57	Strange Darling	movie	73.0	88.0
58	58	The Wild Robot	movie	85.0	91.0
59	59	Anyone But You	movie	63.0	53.0
60	60	Challengers	movie	70.0	85.0
61	61	Boiling Point	movie	76.0	86.0

34 - 61 of 100 Go to: 1

omdb_movies

From OMDb API. Processes less than 25 at a time.



	id	tmdb_id	title	year	genre	runtime	box_office
1	51	785542	The Outrun	2024	Drama	118	$1,013,867
2	32	945961	Alien: Romulus	2024	Horror, Sci-Fi, Thriller	119	$105,313,091
3	67	592695	Pleasure	2021	Drama	109	$106,789
4	68	639720	IF	2024	Animation, Comedy, Drama	104	$111,149,917
5	110	940551	Migration	2023	Animation, Adventure, Comedy	83	$127,306,285
6	8	912649	Venom: The Last Dance	2024	Action, Adventure, Sci-Fi	110	$129,825,476
7	21	1184918	The Wild Robot	2024	Animation, Sci-Fi	102	$137,913,995
8	92	762441	A Quiet Place: Day One	2024	Drama, Horror, Sci-Fi	99	$138,930,553
9	66	385687	Fast X	2023	Action, Adventure, Crime	141	$146,126,015
10	97	1079091	It Ends with Us	2024	Drama, Romance	130	$148,518,266
11	87	365177	Borderlands	2024	Action, Adventure, Comedy	101	$15,482,668
12	90	667538	Transformers: Rise of the Beasts	2023	Action, Adventure, Sci-Fi	127	$157,066,392
13	9	933260	The Substance	2024	Drama, Horror	141	$16,235,191
14	86	216015	Fifty Shades of Grey	2015	Drama, Romance, Thriller	125	$166,167,230
15	95	974576	Conclave	2024	Drama, Thriller	120	$17,412,705
16	69	653346	Kingdom of the Planet of the Apes	2024	Action, Adventure, Drama	145	$171,130,165
17	48	772	Home Alone 2: Lost in New York	1992	Adventure, Comedy, Crime	120	$173,585,516
18	88	603692	John Wick: Chapter 4	2023	Action, Crime, Thriller	169	$187,131,806
19	63	98	Gladiator	2000	Action, Adventure, Drama	155	$187,705,427
20	101	157336	Interstellar	2014	Adventure, Drama, Sci-Fi	169	$188,020,017
21	84	5255	The Polar Express	2004	Animation, Adventure, Comedy	100	$189,528,738
22	91	675353	Sonic the Hedgehog 2	2022	Action, Adventure, Comedy	122	$190,872,904
23	102	269509	The Canterville Ghost	2023	Animation, Comedy, Family	89	$192,310
24	49	573435	Bad Boys: Ride or Die	2024	Action, Adventure, Comedy	115	$193,573,217
25	78	1011985	Kung Fu Panda 4	2024	Animation, Action, Adventure	94	$193,590,620
26	93	823464	Godzilla x Kong: The New Empire	2024	Action, Adventure, Fantasy	115	$196,350,016
27	14	1035048	Elevation	2024	Action, Sci-Fi, Thriller	91	$2,242,603
28	11	974453	Absolution	2024	Action, Crime, Mystery	112	$2,691,166

1 - 28 of 113 Go to: 1

Documentation

collectmoviedata.py		
Function Name & Purpose	**Inputs**	**Outputs**
initializedb() Initialize SQLite database	None	None

with two tables: tmdb_movies and omdb_movies.		
fetch_tmdb_data() Fetch TMDb movies. Process 25 movies at a time, store 100+ total in the database.	None	None
fetch_omdb_data() Fetch OMDb movies. Process 25 movies at a time, store 100+ total in the database.	None	None

visualizations.py		
Function Name & Purpose	**Inputs**	**Outputs**
fetch_data() Connects to SQLite database, fetches relevant movie data from the tmdb_movies and omdb_movies tables, and returns the data as a cleaned Pandas DataFrame	None	**df**: Pandas dataframe
plot_runtime_vs_genre(df) Plots average movie runtime for each genre.	**df**: Pandas dataframe	Bar plot saved as runtime_vs_genre.png and displayed.
plot_ratings_vs_genre_bar(df) Plots the average TMDB ratings for each genre using a	**df**: Pandas dataframe	Bar plot saved as rating_vs_genre_bar.png and displayed.

bar plot.		
plot_ratings_vs_genre_box (df) Plots the average TMDB ratings for each genre using a box plot.	**df**: Pandas dataframe	Box plot saved as rating_vs_genre.png and displayed.
plot_revenue_vs_rating(df) Plots the relationship between revenue and TMDB ratings using a scatter plot.	**df**: Pandas dataframe	Scatter plot saved as revenue_vs_rating.png and displayed.
plot_avg_revenue_vs_genre_box(df) Plots the distribution of revenue within each genre using a box plot.	**df**: Pandas dataframe	Box plot saved as revenue_vs_genre_box.png and displayed.

visual_scorecompare.py		
Function Name & Purpose	**Inputs**	**Outputs**
get_common_movies() Retrieve movies that are present in both TMDB and Watchmode databases, along with their ratings	None	List of tuples where each tuple contains the following values for a common movie: ● tmdb_id ● title ● tmdb_rating: rating from TMDB (out of 10) ● user_score: user rating from Watchmode (out of 100)

		• critic_score: critic score from Watchmode (out of 100)
plot_scores() Plot a line graph comparing TMDB ratings with Watchmode user and critic scores	None	Line graph “ratings_tmdbvswatchmode.png.” comparing ratings from TMDB and Watchmode for each movie

watchmode_database.py		
Function Name & Purpose	**Inputs**	**Outputs**
initialize_database() Sets up a SQLite database and creates a “watchmode_table” table to store movie data	None	None
fetch_movies(movide_id) Fetch movie details (user rating and critic score) from the Watchmode AP	**movie_id:** the unique ID of the movie to fetch details for	A tuple containing two lists: 1. user_scores: list of user scores 2. critic_scores: list of critic scores
store_movie_data(movie_name, movie_type, user_score, critic_score) Stores movie details (name, type, user score, and critic score) into the database	**movie_name:** name of the movie **movie_type:** type of media (like 'movie' or 'tv_show') **user_score:** user score of the movie **critic_score:** critic score of	None

	the movie	
get_movie_list(page = 1, limit = 25) Retrieves a list of movie titles from Watchmode API	**page:** page number to fetch, defaults to 1 **limit:** number of movies to fetch per page, defaults to 25	A list of tuples containing the movie name, ID, and type.
get_movie_data(starting_page=1) Fetch movie data (name, user score, and critic score) for a batch of movies starting from a specific page. Store the data in the database.	**starting_page:** the page number to begin fetching from, defaults to 1.	A tuple containing three lists: 1. movie_names: list of movie names 2. user_scores: list of user scores 3. critic_scores: list of critic scores

watchmodetest.py		
Function Name & Purpose	**Inputs**	**Outputs**
fetch_movies(movie_id)	**movie_id:** Id that IMDB assigns to that particular movie	Generates the variable **info_tuple** which contains a tuple of the user rating then critic score for the given movie.

calculations.py		
Function Name & Purpose	**Inputs**	**Outputs**
process_data(output_file)	**output_file:** The txt file the calculations will be written on	Generates a movie statistcs summary, with average user score, critic score, and TMDB score.

movie_wrapped_report_2024(output_file) Creates a 2024 movie summary report ("wrapped") from an SQLite database. Calculates average ratings, most popular genre, and total revenue and budget for movies released that year.	**output_file:** The txt file the calculations will be written on	Generates a "Movie Wrapped", with number of movies in the database from 2024, average user and critic ratings in 2024, most popular genre in 2024, total revenue and budget in 2024.

Resources

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
Dec 4	Trakt.tv API Key	Trakt API Application	Generated a new API key for us to use, although we ended up not using it.
Dec 4	TVDB API Key	TVDB API Information	Attempted to generate a new API key from TVDB. For some reason, it would not let me create a new account and then log in at all. Kept on making new accounts in hopes of getting in — did not work so we did not end up using this API.
Dec 5	Plotting formatting issues, difficulty creating more "advanced" graphs	Matplotlib Cheat Sheet, Matplotlib Customization Guide	Learned how to create different types of graphs and efficiently "personalize" the graphs (label and colors)
Dec 14	OMDB API Key	OMDB API Key	Generated a new API key from OMDb for us to use.

Dec 14	TMDB API Key	TMDb API	Generated a new API key from TMDb for us to use.
Dec 14	collectmoviedata.py processed 25 items at a time, but it would just repeat the same 25 items.	Google generative AI (AI response that is shown when searching on Google) and Stack Overflow	Coded to check if movie already existed in the database
Dec 14	Outputting “tmdbv3api not resolved from the source”	Stack Overflow	Learned that tmdbv3api external Python package was not downloaded. Fixed it with “pip install tmdbv3api”
Dec 14	General confusion on how to use DB Browser for SQLite	SQL Tutorial	Learned how to use the application, run the database and check if it was processing less than 25 at a time
Dec 14	OMDb didn't have 100 entries and didn’t know how to increase it to at least 100.	ChatGPT	Gave suggestions, although they were not too helpful.
Dec 14	Genre of movie saved as one (“Sci-Fi, Action” is one genre)	Homework 6 and ChatGPT	Refresher on splicing. Ended up only taking the first genre.
Dec 14	No way to automatically find top 5 movies	IMDB website	Manually picked top 5 movies for the list
Dec 14	Suddenly outputting “Too many fetch requests”	Watchmode API Guide	Learned about API usage limits, created a new account with a new API key. While this did solve the

			issue, it solved it only temporarily — Watchmode has a limit of 1000, which we found to be on the smaller side. We had to continuously generate new API keys in order to continue on.
Dec 15	Tried to remove some elements that we didn't want (ex: region, director, runtime, etc.) but it kept on returnin error.	ChatGPT	Removal of these elements were not consistent across the file (ex: elements were only removed at curr.execute but not elsewhere).