PROJECT REPORT

Amani Aggour, Isa Wilson, Rahma Musse

1. Project Goals (Planned APIs & Data)

We all enjoy movies and wanted to understand how budget plays a role in the popularity of a movie, so we began to explore these topics through our final project. Thus, we wanted to explain the relationship between the movie budget and its popularity with audiences through viewer rating and views. We explored this relationship through the topic question: “How do movie budgets and IMDb ratings relate to the popularity of their YouTube trailers?” This guided our functions, calculations, and visualizations.

Through our evaluation, we specifically are looking at the following API/websites:

- TMDB (movie budgets) https://developer.themoviedb.org/docs/getting-started:
    - This was used to gather movie data including specifics about the budget and revenue of each popular movie (movie_id, imdb_id, title, budget)
- OMDb API (movie ratings) https://www.omdbapi.com/:
    - This was used to gather audience rating scores from IMDb, Rotten Tomatoes, etc. (title, imdb_id, imdb_ratings, genre)
- Youtube Data API v3 (movie trailers) https://console.cloud.google.com/marketplace/product/google/youtube.googleapis.com?q=search&referrer=search&project=youtube-api-480400:
    - This was used to gather statistics from popular movie official trailers including (title, view_count, like_count, comment_count)

2. Goals Achieved (APIs Used & Data Collected)

APIs we used

- TMDB API
- OMDB API
- YouTube Data API (v3)

Data that we successfully collected:

- From TMDB:
    - Movie titles
    - TMDB IDs
    - IMDb IDs

    - Production budgets
- From OMDB:
    - Movie titles
    - IMDb IDs
    - Genres
    - IMDb ratings
- From YouTube:
    - Trailer titles
    - Video IDs
    - View count
    - Like count
    - Comment count

The final dataset includes metadata, ratings, and trailer performance for a large set of popular movies.

3. Problems Faced

We faced several issues while working on our project, and we compiled everything in a list:

1. Rate Limits:
    - OMDB and YouTube occasionally declined requests, causing us to constantly have to request new API keys due to reaching daily quota limits. This caused us to get “null” or) values as a result
2. Missing or Null Values:
    - Many OMDB movies did not include IMDb ratings. These had to be skipped before making OMDB requests. Many TMDB movies included budgets with 0, so we made sure to skip those ones to do our calculations.
3. YouTube Title Noise:
    - We had to strip down many titles using regex due to them often containing extra characters, symbols, or multiple versions of the same title.
4. Duplicate YouTube Results:
    - There were many times where we received multiple versions of “official trailers,” requiring us to strip the title down to the base name and then filtering it by creating a “seen list”, and then we checked if a title matched with anything in that list.
5. JSON Variable in TMDB:
    - Some fields in the TMDB (specifically the budget) were occasionally missing due to some foreign films not listing the budget of a movie, making the default value 0, which we had to filter out using an if statement to find budgets that are greater than 0. One of the most difficult parts was having the IMDB ID being listed as a string, and needing to strip it down and turning it into an integer just to avoid duplicate string data.

4. Database Calculations

For calculation #1, it finds the movie with the highest budget and IMDb rating. It only uses movies that appear in the TMDB budget column and OMDB rating column, and shows whether the higher budget movies have higher ratings. For calculation #2, it groups the movies by genre and finds the average IMDb rating for each genre. It then counts how many movies are a part of each genre. For calculation #3, it matches youtube trailers to movies and sums up the views, likes, and comments for each movie's trailers, which then connects the popularity data to the movie's budget from the TMDB API. We provided screenshots of the calculations below.

```python
def calculation_1_budget_vs_rating(conn):
    print("Calculation #1: Highest Budget and Highest IMDb Rating")

    cur = conn.cursor()

    # Join TMDB and OMDB tables together using imdb_id
    cur.execute("""
        SELECT tmdb_movies.title, tmdb_movies.budget, omdb_movies.imdb_rating
        FROM tmdb_movies
        JOIN omdb_movies
        ON tmdb_movies.imdb_id = omdb_movies.imdb_id;
    """)

    rows = cur.fetchall()

    if not rows:
        print("Not enough data to calculate.")
        return

    cleaned_rows = []

    for row in rows:
        title = row[0]
        budget = row[1]
        rating_value = row[2]

        try:
            rating = float(rating_value)
        except:
            continue

        cleaned_rows.append((title, budget, rating))

    if not cleaned_rows:
        print("No valid rating data found.")
        return

    highest_budget_row = max(cleaned_rows, key=lambda r: r[1])
    highest_rating_row = max(cleaned_rows, key=lambda r: r[2])

    print("Highest Budget Movie:")
    print(f"  Title: {highest_budget_row[0]}")
    print(f"  Budget: ${highest_budget_row[1]}")

    print("Highest Rated Movie:")
    print(f"  Title: {highest_rating_row[0]}")
    print(f"  IMDb Rating: {highest_rating_row[2]}")
```

```python
def calculation_2_avg_rating_by_genre(conn):
    print("Calculation #2: Average IMDb Rating by Genre")
    cur = conn.cursor()
    cur.execute("""
        SELECT genres.name, omdb_movies.imdb_rating
        FROM omdb_movies
        JOIN genres ON omdb_movies.genre_id = genres.genre_id;
    """)

    rows = cur.fetchall()
    if not rows:
        print("No OMDB data found.")
        return

    genre_totals = {}
    genre_counts = {}

    for genre_name, rating_value in rows:
        if not genre_name or not rating_value:
            continue

        try:
            rating = float(rating_value)
        except:
            continue

        if genre_name not in genre_totals:
            genre_totals[genre_name] = 0
            genre_counts[genre_name] = 0

        genre_totals[genre_name] += rating
        genre_counts[genre_name] += 1

    avg_ratings = {g: genre_totals[g] / genre_counts[g] for g in genre_totals}

    print("Average IMDb Rating by Genre:")
    for genre, avg in avg_ratings.items():
        print(f"  {genre}: {avg:.2f}")
```

```python
def calculation_3_compare_trailer_popularity_to_budget(conn):
    print("Calculation #3: Comparing Movie Trailer Popularity to Budget")
    cur = conn.cursor()

    cur.execute("SELECT tmdb_id, title, budget FROM tmdb_movies")
    movies = cur.fetchall()

    cur.execute("SELECT title, view_count, like_count, comment_count FROM youtube_trailers")
    trailers = cur.fetchall()

    if not movies or not trailers:
        print("Not enough data found.")
        return

    cleaned_rows = []

    for tmdb_id, movie_title, budget in movies:
        movie_clean = re.sub(r"[^\w\s]", "", movie_title.lower()).strip()

        matched = [
            t for t in trailers
            if movie_clean in re.sub(r"[^\w\s]", "", t[0].lower()).strip()
        ]

        if matched:
            total_views = sum(t[1] for t in matched)
            total_likes = sum(t[2] for t in matched)
            total_comments = sum(t[3] for t in matched)

            cleaned_rows.append((movie_title, budget, total_views, total_likes, total_comments))

    if not cleaned_rows:
        print("No matching trailer data found.")
        return

    top_movie = max(cleaned_rows, key=lambda r: r[2])

    print("Movie with the Most Trailer Views:")
    print(f" Title: {top_movie[0]}")
    print(f" Budget: ${top_movie[1]}")
    print(f" Total Views: {top_movie[2]}")
```

```
Calculation 1: Highest Budget and IMDb Rating
Category,Title,Budget
Highest Budget Movie,TRON: Ares,220000000
Category,Title,Rating
Highest Rated Movie,Kantara – A Legend: Chapter 1,8.5

Calculation 2: Average IMDb Rating by Genre
Genre,IMDb Rating
Animation,8.066666666666665
Action,6.277777777777779
Fantasy,6.2
Family,7.1
Drama,7.3
Comedy,7.150000000000001
Horror,5.4
Crime,6.3
Biography,6.1

Calculation 3: Movie Trailer Popularity vs Budget
Title,Budget,Total_Views,Total_Likes,Total_Comments
Predator,15000000,26687697,317516,26129
Avatar,237000000,30586018,615483,34614
Avatar: Fire and Ash,400000000,30586018,615483,34614
It,35000000,52067961,531195,22345
Five Nights at Freddy's,20000000,15681295,570868,33965
TRON: Ares,220000000,13668180,185179,14912
Bugonia,50000000,14912154,55259,3350
The Running Man,110000000,16331732,101667,12440
Wake Up Dead Man: A Knives Out Mystery,210000000,3831656,91393,3414
The Conjuring: Last Rites,55000000,25534859,226030,6707
Superman,225000000,1890732,122657,3619
Frankenstein,120000000,13178136,176993,7597
Five Nights at Freddy's 2,36000000,15681295,570868,33965
Moana 2,150000000,1781907,34958,0
Roofman,18000000,14179057,58620,3447
Predator: Badlands,105000000,26687697,317516,26129
```

5. Visualization

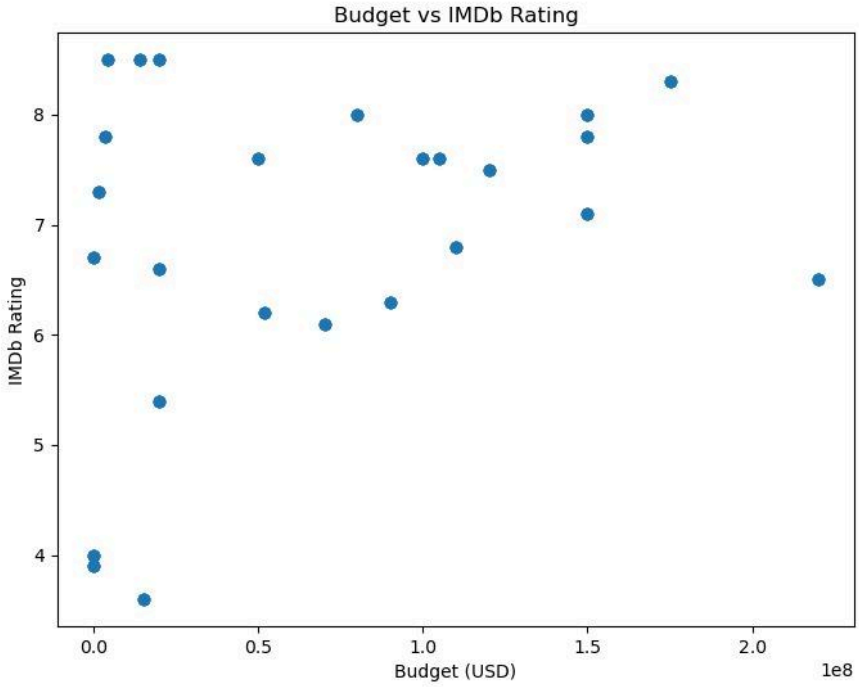
Visualizations we included are:

- Budget vs. IMDB Rating
- Average IMDB by genre
- Youtube trailer views vs movie budget
- Genre distribution in OMDB movies
- Budget distribution of movies

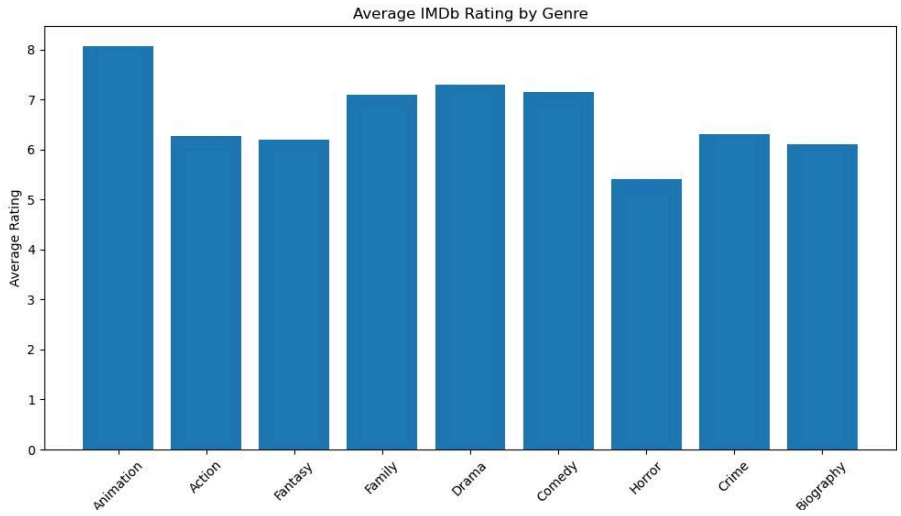
Description:

- Budget vs. IMDB Rating

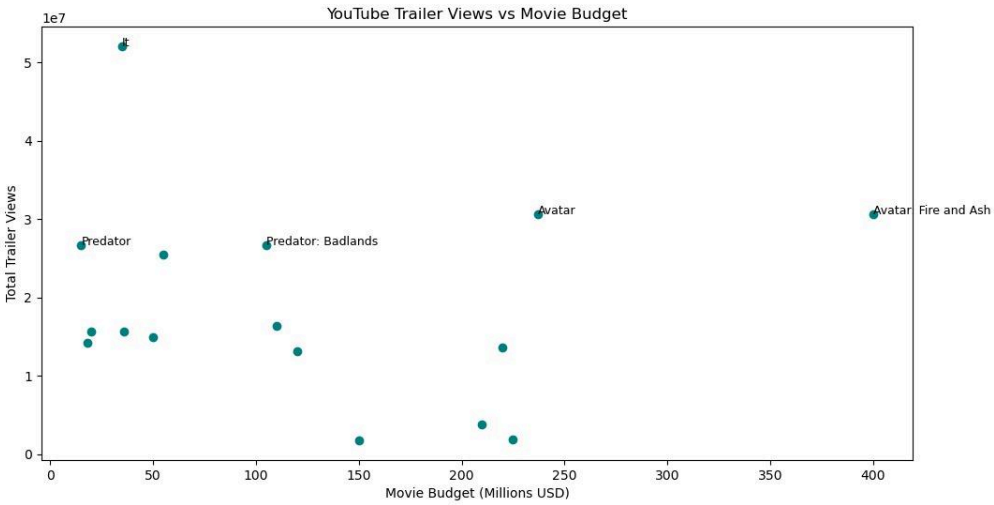
- The scatter plot shows that IMDB ratings vary across all budget levels. There is no correlation being shown between budget and rating. Both low and high budget movies can receive high ratings. Even high budget movies can receive only average ratings.



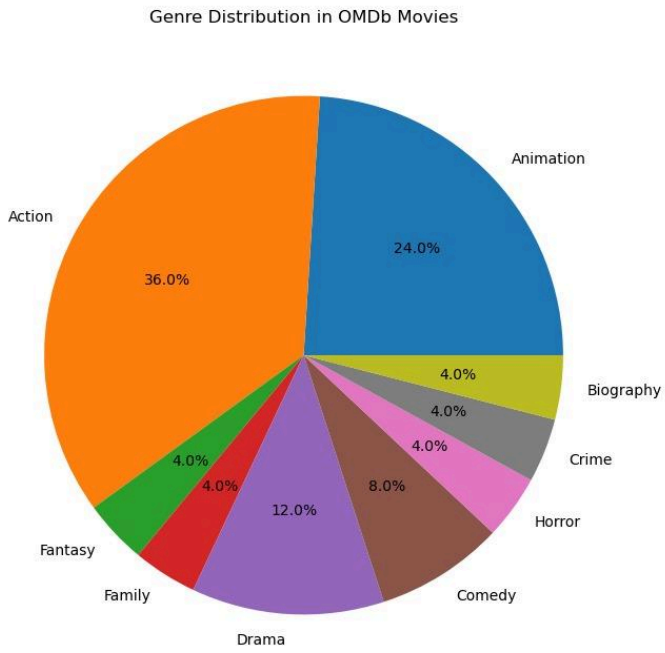
- Average IMDB by Genre
  - The bar graph shows that if there is a correlation between the audience ratings and genre of the movies, most genres such as Animation, Drama, Comedy, and Family movies tend to have higher average ratings, except for horror which has the lowest ratings. The other genres such as Action, Fantasy, Crime, and Biography, all fall into the mid range. This means that no specific genre is guaranteed a high or low movie rating, but there is a relationship between genre and ratings.



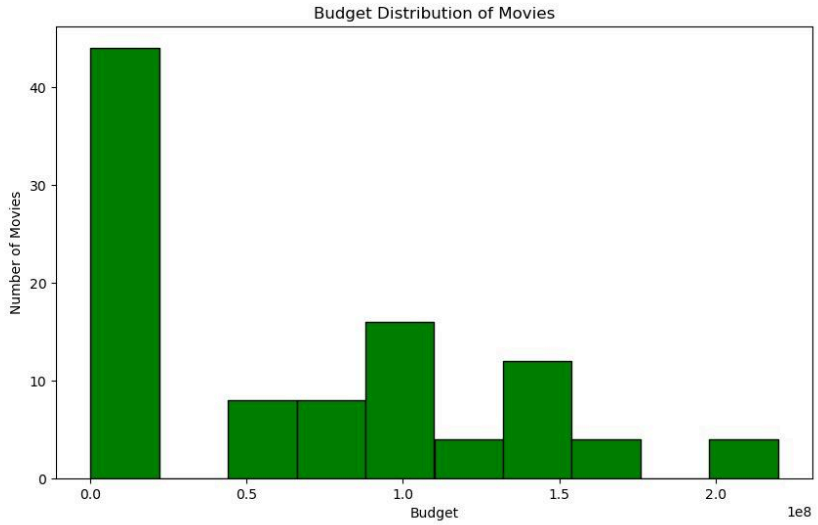
- Youtube trailer views vs movie budget
    - The visualization shows the relationship between trailer popularity (view count) and TMDB budget. The trend suggests that there is no such relation between higher budgets and trailer popularity. This is a result of the spread out scatter of movies



- Genre distribution in OMDB movies
    - A visual pie chart that differentiates the different genres in the OMDB and provides a percentage based on the ratio of that category compared to the rest of the movies in the database. Based on this chart, we can see that the API contains majority action movies, indicating that this most likely is the most popular genre.



- Budget distribution of movies
    - Most movies displayed in the histogram have lower budgets, meaning that when the number of movies decreases, it causes the movie budgets to increase. A small amount of movies have high budgets, meaning that the high budget movies are way less common compared to middle to low budget movies



6. Instructions for Running the Code

Our API keys are hidden in the github, but you should still have access to it while running the file. For the first file (mainfunctions.py), we've been having trouble with our API keys, specifically for OMDB and Youtube Data. I would try to avoid running that file because sometimes the youtube reviews output will return less than 100 rows of data, as well as the OMDB returning null. We think our API keys get messed up sometimes due to having to run the file a lot and reaching the request limit. It does return at least 100 rows of usable data and it did work during the grading session as well. If you do run the file, we made sure to create original saved copies (master version) if the mainfunctions.py doesn't work accurately. If you go to the main function at the end of database.py, we are using the master versions to add the data to the movies database. So to look at our data, we would recommend looking at the JSON files (The Master Version, which is the original with the usable data). As for the second file (database.py), it should run successfully as long as the mainfunctions file is not run again (due to our data fluctuating because of our API keys), this would cause our JSON file to become inaccurate. Also, we made a copy of the database (movies.db) for reference if anything goes wrong. For the third file (calculations.py), it should run successfully as well with the same rules applying as the database file (not running the mainfunctions.py).

MORE WORK BELOW

7. Updated Function Diagram

Final Project SI 201

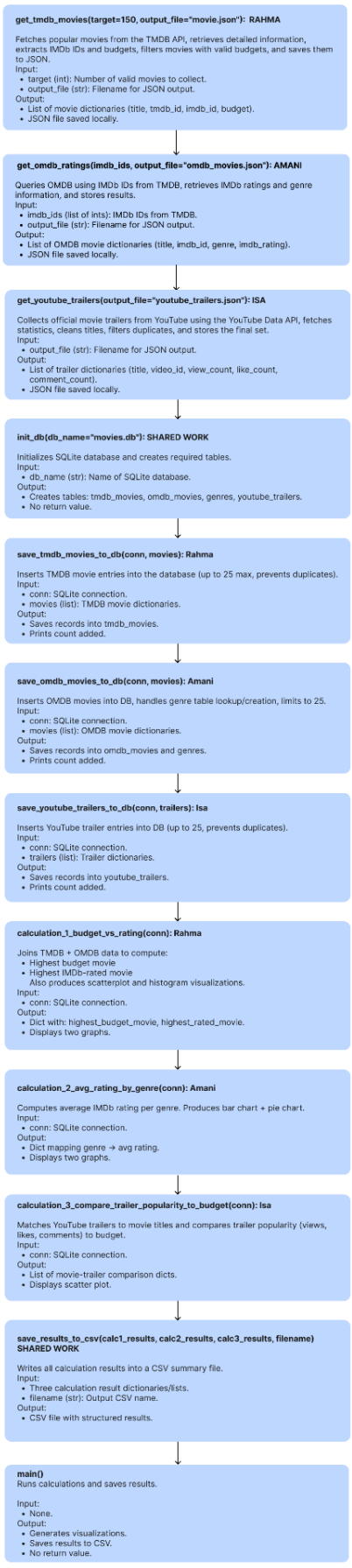
Collaborators:
Isa Wilson,
Rahma Musse,
Amani Aggour

APIs:

TMDB (movie budgets):
https://developer.themoviedb.org/docs/getting-started

OMDb API (movie ratings):
https://www.omdbapi.com/

Youtube Movie Reviews API:
https://developers.google.com/youtube/v3/docs



---

8. Documentation of Resources Used

TMDB API Documentation
https://developer.themoviedb.org/docs

OMDB API Documentation
https://www.omdbapi.com/

YouTube Data API v3 Documentation
https://developers.google.com/youtube/v3/docs

Chat GPT
https://chatgpt.com/

Date	Issue Description	Location of Resource	Results (did it resolve the issue)
12/11/25	API key protection -Through the entire process of the project, constantly getting 0 values or null when trying to gather data and store it into the json, we knew there must be a more pressing issue.	-Help from group partner family member -ChatGPT	Solved - After getting advise on ChatGPT and advice from others we knew who coded, we realized that a major reason that we have been continually having API key issues might be due to having the key on a public Github causing it to possibly get stolen by those accessing the site, thus, we utilized gitignore to hide our API keys to avoid them from getting altered.
12/11/25	TMDB/OMDB Join & duplicate string data -When working on the join for TMDB and OMDB on the IMDB_ID, we were not sure what the rubric met exactly by joining it if it meant to create one larger table with smaller tables for IMDB_ID to avoid duplicate strings	-TMDB API -Office hour/grading session	Solved - After attending the grading session and getting an understanding that we should be treating the API for OMDB and TMDB as different sets and the duplicate string data was pertaining genres within TMDB we created a subtable for TMDB genres to reference the id/number integers rather than listing them as strings.
12/10/25	TMDB returning budgets of 0	-calculations.py -TMDB API	Solved - In seeing that a lot of the budgets for TMDB

	-When looking through our database file we realized that several of the TMDB inputs were showing up with a budget of 0, causing the calculation between budget and youtube trailer views to show outliers to the dataset		were showing up as 0 we created an if statement within the mainfunctions file to sort out movies with budgets of 0, mainly those that were unreported like indie or international files, which allowed is to get a much clear dataset to easily compare with other APIs.
12/9/25	Github branch username authorization declined -As a result of forking a repository which logged a group member into a personal account, there were authentication issues for one of our group members which made it hard to get and share updated code.	-Help from group partner family member -ChatGPT	Solved - Before we could get help on the issue, we utilized email to share our code amongst our group partners, so that we could keep up-to-date. After attempting to use ChatGPT to get an understanding of what the specific issue was an how to fix it using various methods like updating the vim to check the mac username, we ended up getting help from a group partner sibling to troubleshoot the git error to uncover that I was incorrectly logged into another github account which created authentication issues and getting help fixing a merge conflict that had also occurred.
12/9/25	Issues with YouTube visualizations and database showing duplicate movies -When looking at data outputted from the Youtube API key, we realized that a lot of them were duplicate movies as well as many characters were showing up as numbers and random symbols	-mainfunctions.py	Solved - As a result of seeing these duplicates, we began to add regex as a method to strip the data down and removing characters in order to create a cleaning json file in addition, I created a seen list to store movies that had been seen more than once, and a unique list to store those titles that were only included once in the API to avoid duplicate data.
12/9/25	Youtube & OMDB API key returning 0 -As a result of running	-OMDB API -YouTube Data API v3	Solved - We researched on each API website and looked at the dashboard of how many requests we had

	our code to debug issues that we were encountering, we began to notice that some API keys were returning an output of 0 and did not know why		made and realized that we had met the cap for the api keys which happened several times throughout the project. We ended up creating new API keys and requesting less pages within our for loops to avoid adding so many requests to an API at a time
12/7/25	Github merge conflict encountered -Despite constantly texting each other when we were working on the file, our group accidentally forgot to git pull before pushing which created server merge conflicts, causing our team members to be out of sink with coding updates	-Office Hours -Help from group partner family member	Solved - After talking to TAs in class and attending office hours, we were advised to create a new repository to avoid having to worry about the merge conflict as a result of not being able to get help on the issue in person/for an extended amount of time
12/6/25	Youtube API key returning only 50 rows -Since switching to the Youtube API key although TMDB and OMDB were both returning 100 rows, we noticed that the Youtube API was not due to the cap that was on per user request to the API	-YouTube Data API v3 Documentation -ChatGPT	Solved - As a result of doing research into the specific API website as well as getting advice from ChatGPT on how to approach this issues, we ended up utilizing the pagination method suggested as a way to limit the overload on the API server and avoid hitting rate limits by splitting the data amongst pages
12/4/25	Exploring a new research question -After switching to NYTimes Article Search API, we realized that the article search was too broad, causing it to not allow us to filter out specific movie titles in the overall article heading or by genre.	-NYTimes Article Search API -YouTube Data API v3	Solved - After trying to make the API work through trying to strip the heading and sort by genre, we realized it was best to reexamine the research question and get a new API key and looked at the new question of how movie budget and ratings correlate with movie trailer popularity through the YouTube Data API v3 which was a lot more reliable
12/2/25	Issues with NYTimes	-NYTimes Movie Review	Solved - As a result, we

	data collection - When utilizing the New York times movies reviews API as the last API key with TMDB and OMDB, I quickly noticed it was constantly giving errors and not collecting any data, utilized ChatGPT and asked peers why this may be happening and they suggesting printing the response code and we noticed it was due to the API being deprecated	API -ChatGPT -Help from group partner family member	ended up switching the API to search movies through the NYT Article Search API to see if there was a correlation between NYT mentions and budget
12/1/25	Trouble figuring out how to store the data - As a result of not fully understanding the instructions about storing the data and giving different feedback on storing it into a json first or directly into the database, we started to seek help from others.	-Functions.db -Piazza -In-person office hours	Solved - We utilized resources to get clarification on how to store the data gathered, we ended up loading it into a json file and getting to 100 results.
11/24/25	Struggles with opening API keys - We were unsure of how to open and sign in to each API key as a result of each website following a different format and the sign in buttons not being obvious on each	-Google -ChatGPT conversation	Solved - We utilized resources to get code examples/step-by-step for getting started with API key, which allowed us to be able to open the API to access data properly without errors