SI 206 Final Project Report
The A-Team
Rickey Dong, Jason Le, Sieon Lim

# Movies

https://github.com/HotRiceEatNow/Si-206-final-project

A. Original goals (The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points))

The primary objective of our project was to investigate movies and their revenues, budgets, and performance. We originally wanted to use the following APIs and website:

- TMDb + OMDb APIs: To retrieve movie titles, ratings, and metadata such as release year and genre.

- SERP API: To gather additional information on whether a movie is currently being shown in theaters. This would help us assess ongoing popularity and potential impact on revenue or ratings.

- Box Office Mojo (via web scraping): To extract box office revenue data for comparison with ratings.

Our plan was to store the collected data in an SQLite database, join the tables via unique movie identifiers, and perform calculations such as average ratings, revenue per genre, and correlation metrics. Visualizations were to be created using Matplotlib to clearly display our findings.

B. Achieved goals (The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points))

It turned out that the SERP API wasn't usable for free plans. It would always return an error message no matter what we tried, however, we did manage to achieve most of our original goals:

- We successfully used the OMDb API and TMDb API to retrieve IMDb ratings and metadata for a large number of movies.

- We scraped revenue data from Box Office Mojo to supplement our database with financial performance information.

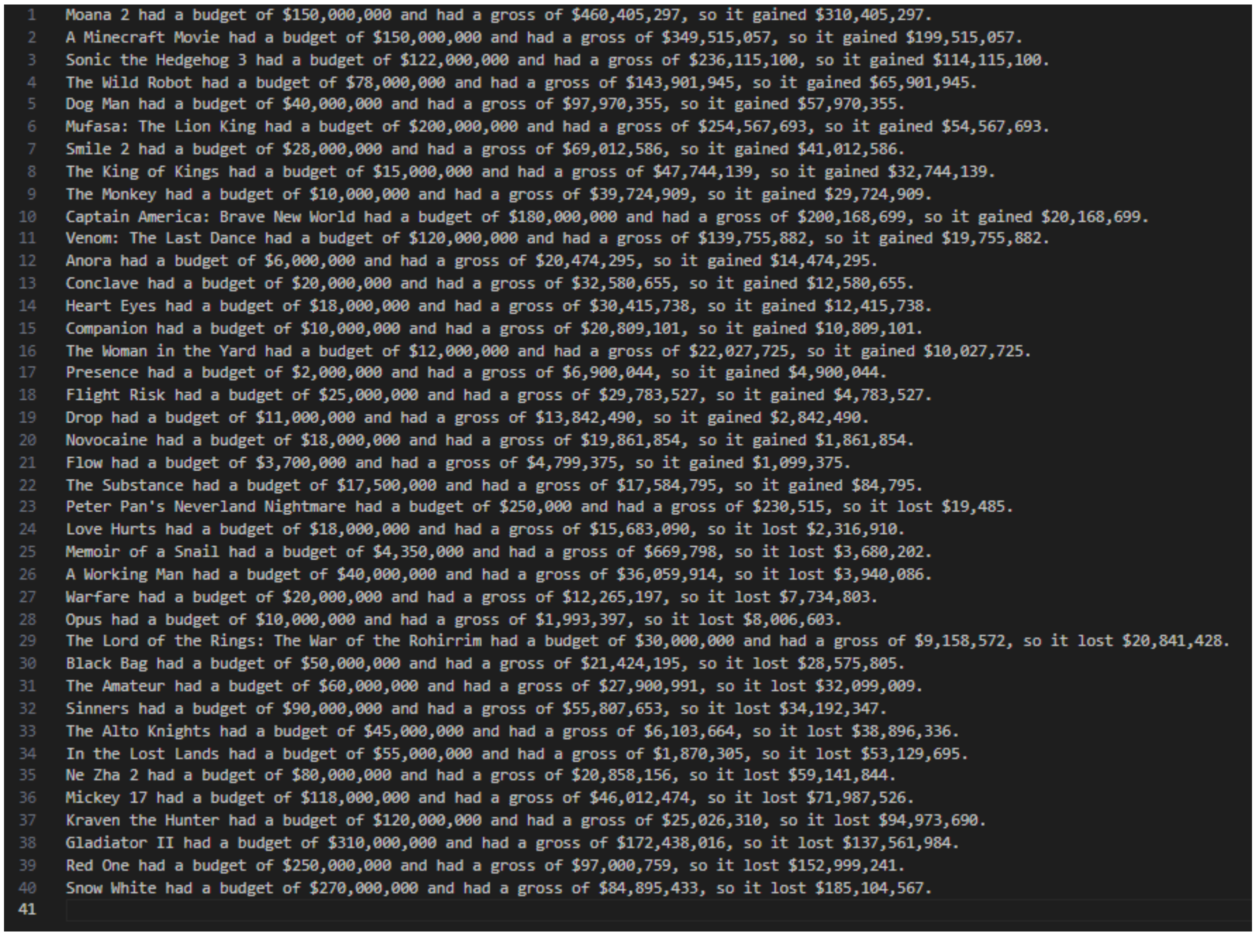
Finally, despite this change, our core objective remained intact: to analyze movies' ratings, performance, budget, and revenue. We were able to store, analyze, and visualize this data to explore all of this effectively.

C. Problems that we faced (The problems that you faced (10 points))

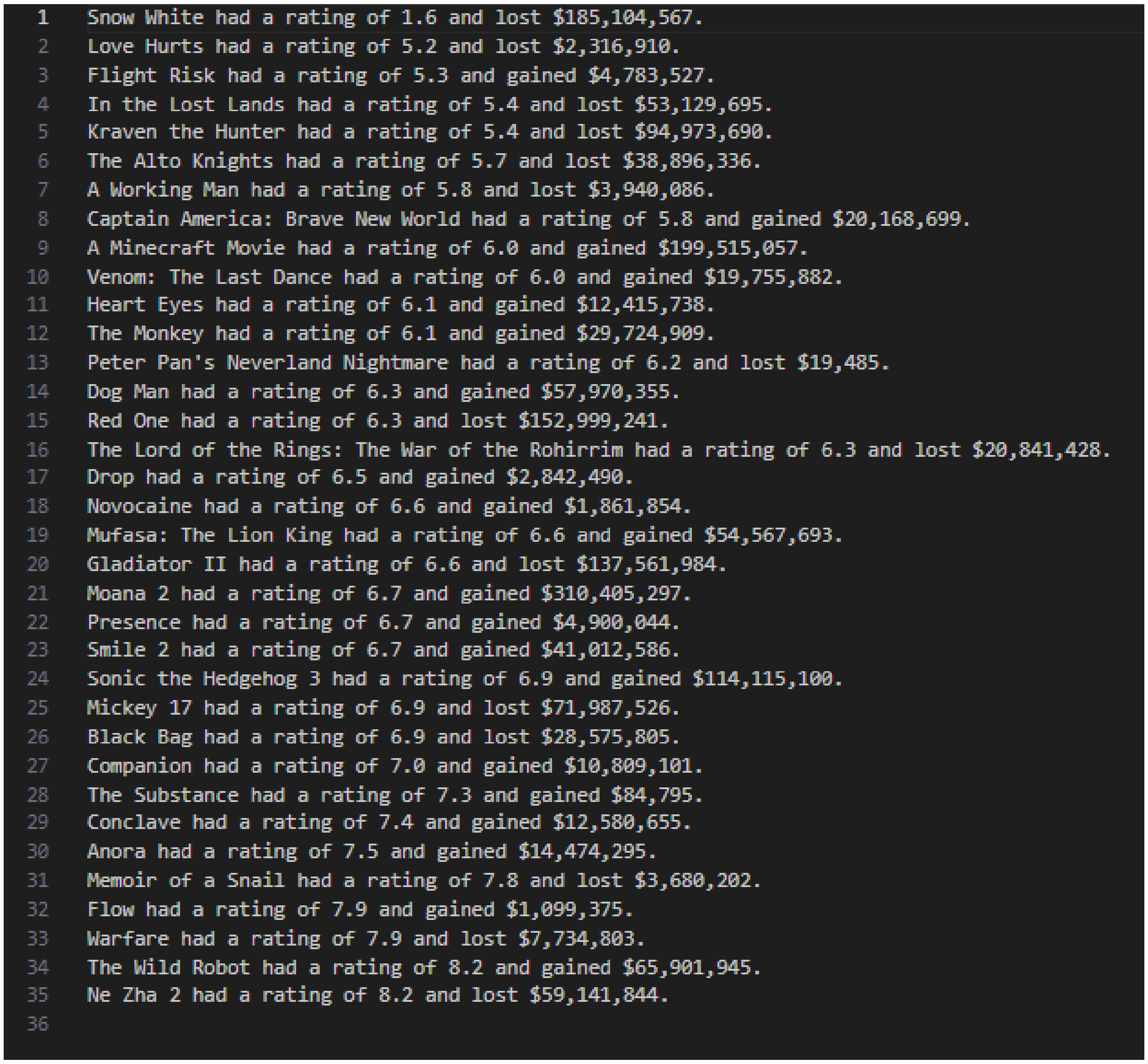
One big problem that we faced was that the API sources we used often had missing or incomplete data. For example, even after running our data gathering Python file many times to ensure that we had 100 rows in our database, only about 30 rows were actually useful for calculations and visualizations. This could be because one API had missing information, like $0 for the budget field. To make matters worse, even if one API (the TMDb API, for example) had all valid data, there was a chance that when we try to fetch more data for that corresponding movie from the OMDb API, the OMDb API could have even just one invalid field, like NULL for IMDb_rating. In the end, we decided to just deal with it and live with the fact that we could make calculations / visualizations for a subset of all the rows we collect. To help remedy this as best we could, we collected more than 100 rows to create as best of an insight as we could.

Another problem we faced was difficulty ensuring data integrity when performing API calls between different resources. Because some movies had slightly different naming conventions or formatting across APIs (e.g., “Spider-Man: No Way Home” vs. “Spiderman No Way Home”), simple API calls using titles often failed or returned incorrect matches. We initially tried to use fuzzy matching to solve this issue, but this was too complicated. Eventually, we pivoted to using IMDb IDs wherever possible to enforce consistent linking across tables, which improved accuracy but limited our dataset to only movies where all APIs provided a matching ID.

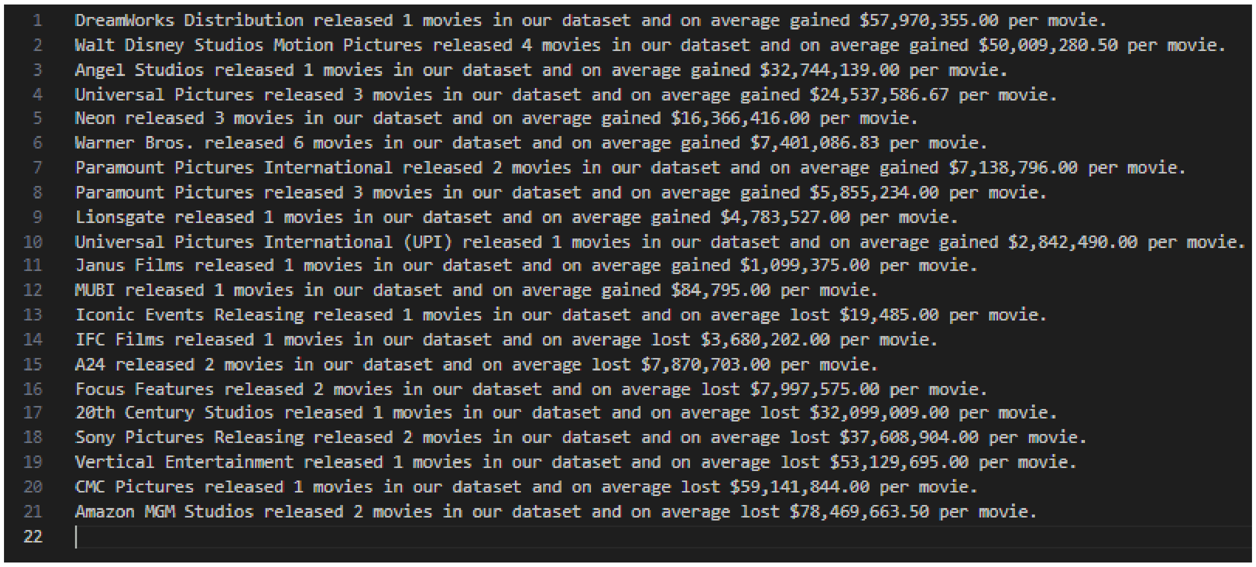
## D. Calculations (The calculations from the data in the database (i.e. a screenshot) (10 points))



```
Moana 2 had a budget of $150,000,000 and had a gross of $460,405,297, so it gained $310,405,297.
A Minecraft Movie had a budget of $150,000,000 and had a gross of $349,515,057, so it gained $199,515,057.
Sonic the Hedgehog 3 had a budget of $122,000,000 and had a gross of $236,115,100, so it gained $114,115,100.
The Wild Robot had a budget of $78,000,000 and had a gross of $143,901,945, so it gained $65,901,945.
Dog Man had a budget of $40,000,000 and had a gross of $97,970,355, so it gained $57,970,355.
Mufasa: The Lion King had a budget of $200,000,000 and had a gross of $254,567,693, so it gained $54,567,693.
Smile 2 had a budget of $28,000,000 and had a gross of $69,012,586, so it gained $41,012,586.
The King of Kings had a budget of $15,000,000 and had a gross of $47,744,139, so it gained $32,744,139.
The Monkey had a budget of $10,000,000 and had a gross of $39,724,909, so it gained $29,724,909.
Captain America: Brave New World had a budget of $180,000,000 and had a gross of $200,168,699, so it gained $20,168,699.
Venom: The Last Dance had a budget of $120,000,000 and had a gross of $139,755,882, so it gained $19,755,882.
Anora had a budget of $6,000,000 and had a gross of $20,474,295, so it gained $14,474,295.
Conclave had a budget of $20,000,000 and had a gross of $32,580,655, so it gained $12,580,655.
Heart Eyes had a budget of $18,000,000 and had a gross of $30,415,738, so it gained $12,415,738.
Companion had a budget of $10,000,000 and had a gross of $20,809,101, so it gained $10,809,101.
The Woman in the Yard had a budget of $12,000,000 and had a gross of $22,027,725, so it gained $10,027,725.
Presence had a budget of $2,000,000 and had a gross of $6,900,044, so it gained $4,900,044.
Flight Risk had a budget of $25,000,000 and had a gross of $29,783,527, so it gained $4,783,527.
Drop had a budget of $11,000,000 and had a gross of $13,842,490, so it gained $2,842,490.
Novocaine had a budget of $18,000,000 and had a gross of $19,861,854, so it gained $1,861,854.
Flow had a budget of $3,700,000 and had a gross of $4,799,375, so it gained $1,099,375.
The Substance had a budget of $17,500,000 and had a gross of $17,584,795, so it gained $84,795.
Peter Pan's Neverland Nightmare had a budget of $250,000 and had a gross of $230,515, so it lost $19,485.
Love Hurts had a budget of $18,000,000 and had a gross of $15,683,090, so it lost $2,316,910.
Memoir of a Snail had a budget of $4,350,000 and had a gross of $669,798, so it lost $3,680,202.
A Working Man had a budget of $40,000,000 and had a gross of $36,059,914, so it lost $3,940,086.
Warfare had a budget of $20,000,000 and had a gross of $12,265,197, so it lost $7,734,803.
Opus had a budget of $10,000,000 and had a gross of $1,993,397, so it lost $8,006,603.
The Lord of the Rings: The War of the Rohirrim had a budget of $30,000,000 and had a gross of $9,158,572, so it lost $20,841,428.
Black Bag had a budget of $50,000,000 and had a gross of $21,424,195, so it lost $28,575,805.
The Amateur had a budget of $60,000,000 and had a gross of $27,900,991, so it lost $32,099,009.
Sinners had a budget of $90,000,000 and had a gross of $55,807,653, so it lost $34,192,347.
The Alto Knights had a budget of $45,000,000 and had a gross of $6,103,664, so it lost $38,896,336.
In the Lost Lands had a budget of $55,000,000 and had a gross of $1,870,305, so it lost $53,129,695.
Ne Zha 2 had a budget of $80,000,000 and had a gross of $20,858,156, so it lost $59,141,844.
Mickey 17 had a budget of $118,000,000 and had a gross of $46,012,474, so it lost $71,987,526.
Kraven the Hunter had a budget of $120,000,000 and had a gross of $25,026,310, so it lost $94,973,690.
Gladiator II had a budget of $310,000,000 and had a gross of $172,438,016, so it lost $137,561,984.
Red One had a budget of $250,000,000 and had a gross of $97,000,759, so it lost $152,999,241.
Snow White had a budget of $270,000,000 and had a gross of $84,895,433, so it lost $185,104,567.
```

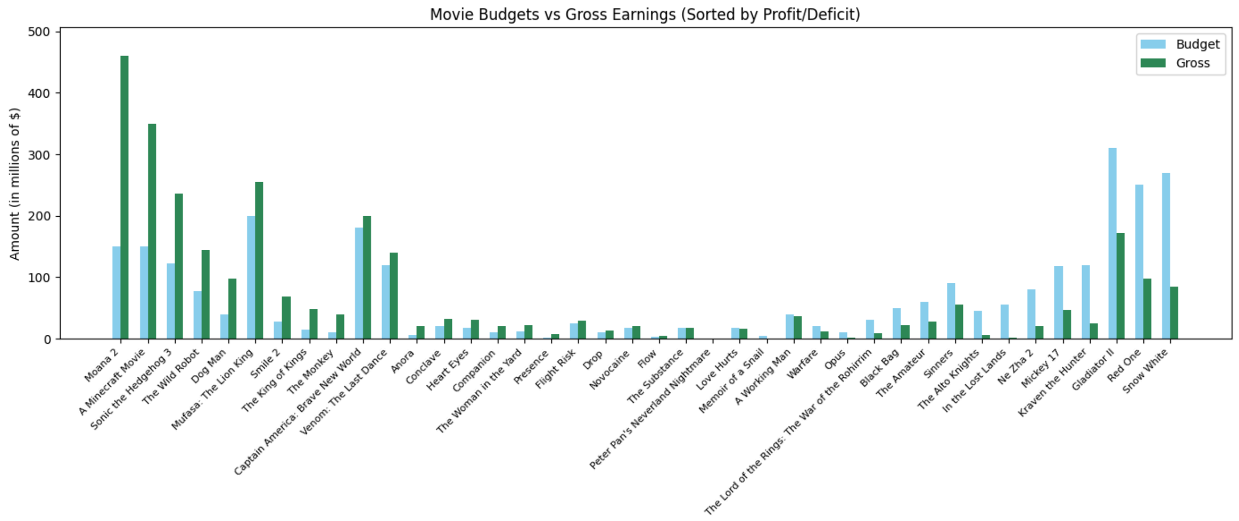


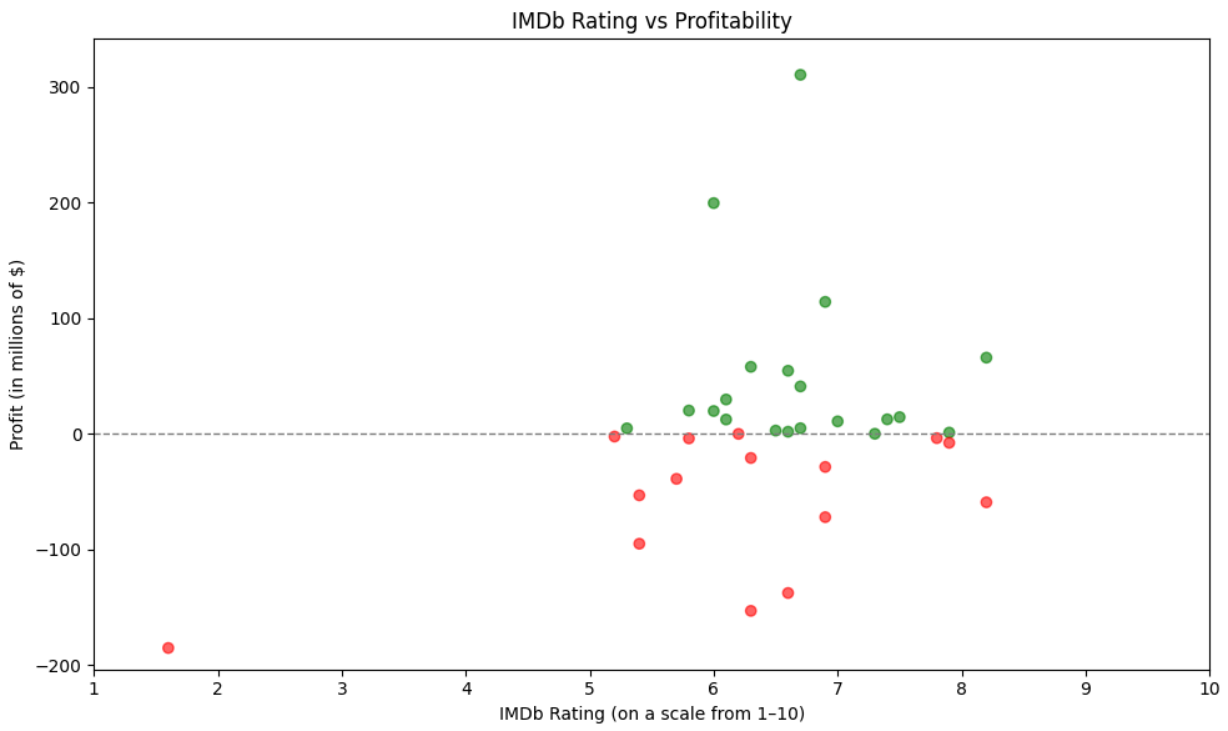
```
Snow White had a rating of 1.6 and lost $185,104,567.
Love Hurts had a rating of 5.2 and lost $2,316,910.
Flight Risk had a rating of 5.3 and gained $4,783,527.
In the Lost Lands had a rating of 5.4 and lost $53,129,695.
Kraven the Hunter had a rating of 5.4 and lost $94,973,690.
The Alto Knights had a rating of 5.7 and lost $38,896,336.
A Working Man had a rating of 5.8 and lost $3,940,086.
Captain America: Brave New World had a rating of 5.8 and gained $20,168,699.
A Minecraft Movie had a rating of 6.0 and gained $199,515,057.
Venom: The Last Dance had a rating of 6.0 and gained $19,755,882.
Heart Eyes had a rating of 6.1 and gained $12,415,738.
The Monkey had a rating of 6.1 and gained $29,724,909.
Peter Pan's Neverland Nightmare had a rating of 6.2 and lost $19,485.
Dog Man had a rating of 6.3 and gained $57,970,355.
Red One had a rating of 6.3 and lost $152,999,241.
The Lord of the Rings: The War of the Rohirrim had a rating of 6.3 and lost $20,841,428.
Drop had a rating of 6.5 and gained $2,842,490.
Novocaine had a rating of 6.6 and gained $1,861,854.
Mufasa: The Lion King had a rating of 6.6 and gained $54,567,693.
Gladiator II had a rating of 6.6 and lost $137,561,984.
Moana 2 had a rating of 6.7 and gained $310,405,297.
Presence had a rating of 6.7 and gained $4,900,044.
Smile 2 had a rating of 6.7 and gained $41,012,586.
Sonic the Hedgehog 3 had a rating of 6.9 and gained $114,115,100.
Mickey 17 had a rating of 6.9 and lost $71,987,526.
Black Bag had a rating of 6.9 and lost $28,575,805.
Companion had a rating of 7.0 and gained $10,809,101.
The Substance had a rating of 7.3 and gained $84,795.
Conclave had a rating of 7.4 and gained $12,580,655.
Anora had a rating of 7.5 and gained $14,474,295.
Memoir of a Snail had a rating of 7.8 and lost $3,680,202.
Flow had a rating of 7.9 and gained $1,099,375.
Warfare had a rating of 7.9 and lost $7,734,803.
The Wild Robot had a rating of 8.2 and gained $65,901,945.
Ne Zha 2 had a rating of 8.2 and lost $59,141,844.
```

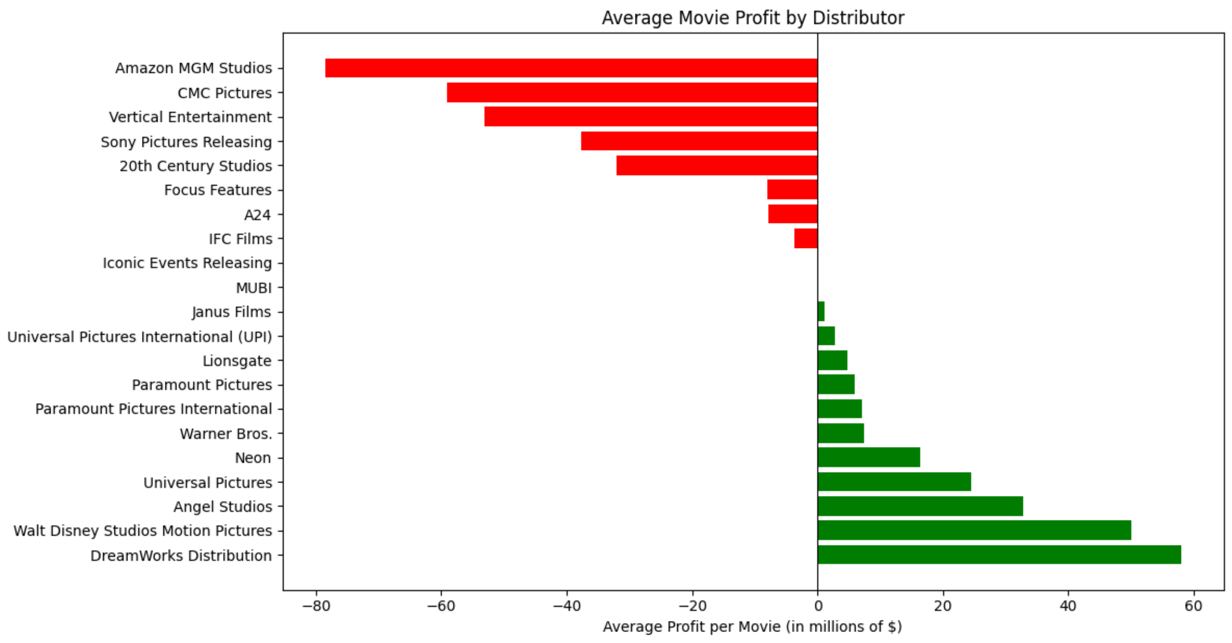


```
DreamWorks Distribution released 1 movies in our dataset and on average gained $57,970,355.00 per movie.
Walt Disney Studios Motion Pictures released 4 movies in our dataset and on average gained $50,009,280.50 per movie.
Angel Studios released 1 movies in our dataset and on average gained $32,744,139.00 per movie.
Universal Pictures released 3 movies in our dataset and on average gained $24,537,586.67 per movie.
Neon released 3 movies in our dataset and on average gained $16,366,416.00 per movie.
Warner Bros. released 6 movies in our dataset and on average gained $7,401,086.83 per movie.
Paramount Pictures International released 2 movies in our dataset and on average gained $7,138,796.00 per movie.
Paramount Pictures released 3 movies in our dataset and on average gained $5,855,234.00 per movie.
Lionsgate released 1 movies in our dataset and on average gained $4,783,527.00 per movie.
Universal Pictures International (UPI) released 1 movies in our dataset and on average gained $2,842,490.00 per movie.
Janus Films released 1 movies in our dataset and on average gained $1,099,375.00 per movie.
MUBI released 1 movies in our dataset and on average gained $84,795.00 per movie.
Iconic Events Releasing released 1 movies in our dataset and on average lost $19,485.00 per movie.
IFC Films released 1 movies in our dataset and on average lost $3,680,202.00 per movie.
A24 released 2 movies in our dataset and on average lost $7,870,703.00 per movie.
Focus Features released 2 movies in our dataset and on average lost $7,997,575.00 per movie.
20th Century Studios released 1 movies in our dataset and on average lost $32,099,009.00 per movie.
Sony Pictures Releasing released 2 movies in our dataset and on average lost $37,608,904.00 per movie.
Vertical Entertainment released 1 movies in our dataset and on average lost $53,129,695.00 per movie.
CMC Pictures released 1 movies in our dataset and on average lost $59,141,844.00 per movie.
Amazon MGM Studios released 2 movies in our dataset and on average lost $78,469,663.50 per movie.
```

E. Visualizations (The visualizations that you created (i.e. screenshot or image file) (10 points))







F. Instructions for running the code (Instructions for running your code (10 points))

1. Clone our repository from GitHub:
   git clone git@github.com:HotRiceEatNow/Si-206-final-project.git

2. Make sure you have the necessary modules installed (requests, BeautifulSoup, matplotlib)
3. Go into the data_gathering directory and run main.py for as many times as you like:
   ```
   ~/Si-206-final-project/data_gathering$ python3 main.py
   ```
4. Go into the data_processing directory and run main.py to create calculations and visualizations:
   ```
   ~/Si-206-final-project/data_processing$ python3 main.py
   ```
5. Look in the analysis_outputs directory to find the calculations and visualizations:
   ```
   ~/Si-206-final-project/data_processing/analysis_outputs$ ls
   ```

G. Function documentation (Documentation for each function that you wrote. This includes describing the input and output for each function (20 points))

data_gathering/

main.py

`def main():`

The main function to gather data from all the various sources and populate the SQLite database. No input is needed. Upon execution of this function, movies.db will be populated with at most 25 items/rows and the columns for each row will be filled out accordingly with information from the sources.

boxofficemojo.py

`def fetch_html(url):`

Fetches HTML content from a given url. Input is the URL. Output is the HTML from that URL.

`def parse_html(html):`

Parses HTML using BeautifulSoup. Input is the HTML. Output is the BeautifulSoup object.

`def extract_table(soup):`

Extracts the movie table from the Soup object. Input is the Soup object. Output is the table object that is present on the BoxOfficeMojo website.

`def normalize_cell(text):`

Performs data cleaning on a cell. Input is a given cell in the table on the BoxOfficeMojo website. Output is either the value itself or None if it is originally “-”.

`def parse_movie_row(row):`

Parses a single row of movie data and returns it
as a dictionary. Input is the row in the BoxOfficeMojo
table. Output is a dictionary of all the column values.

def fetch_bom_data_for_title(title):

Returns the scraped data for a given movie title. Input
is a title of a movie. Output is a 4-tuple of gross,
theaters, total_gross, and distributor.

db.py

def create_database():

Creates the SQLite database, movies.db. It creates
the necessary tables within this database. No input
is needed.

def get_or_create_genre_id(genre_name):

Fetches the Genre ID for an associated genre name.
If that genre name does not exist yet in
the table, create a new
row for this genre. Input is the genre name. Output is
the associated Genre ID.

def get_or_create_distributor_id(name):

Fetches the Distributor ID for an associated
distributor name. If that distributor name does not
exist yet in the table, create a new row for this
distributor. Input is the distributor name. Output is
the associated Distributor ID.

```
def insert_or_update_movie( title,
                            release_year=None,
                            genre_id=None,
                            tmdb_id=None,
                            imdb_id=None,
                            popularity=None,
                            vote_count=None,
                            average_vote=None,
                            budget=None,
                            imdb_rating=None,
                            imdb_votes=None,
                            gross=None,
                            theaters=None,
                            total_gross=None,
```

distributor=None):

Inserts a new movie or updates an existing movie in the database Movies table. Input is the movie that is being inserted along with its relevant details. Output is the ID of the movie that was just modified.

def insert_showtimes_data(movie_id, slots_count):

Inserts a movie_id along with the showtimes slots it has into the database table. Input is the movie_id and the count of its showtime slots.

def print_database_state():

A helper function for debugging. Prints the state of the entire database. No input needed.

omdb.py

def fetch_omdb_data(imdb_id):

Uses the OMDb API endpoint to fetch data about a single movie, represented by imdb_id. Input is the movie's imdb_id. Output is the "Genre", "imdbRating", and "imdbVotes" fields obtained from the OMDb API.

tmdb.py

def get_last_page_retrieved():

Reads from a file called "last_tmdb_page.txt" in order to find out the previous page of TMDb's API endpoint the script left off at. No input needed. Output is the previous page read from the API, or 0 if no pages read. (the API starts at page=1)

def set_last_page_retrieved(page_num):

Stores the current page number locally so the next run will pick up from there. The input is the page we just finished reading the API from.

def fetch_tmdb_popular_movies(page):

Fetches popular movies from TMDb's API endpoint at a particular page number. Input is the page to read from. Output is the list of movie dictionaries from the JSON response.

def get_tmdb_movie_details(tmdb_id):

Fetches more details from a different TMDb API endpoint about a certain movie. Input is the tmdb_id for that movie. Output is the imdb_id and the budget associated with that movie.

data_processing/

main.py

def clean_gross(gross_str):

Cleans a given string that represents the gross amount a movie made. Input is the gross amount retrieved from the database. Output is a cleaned-up version of this string as an integer or None if it could not be cleaned.

def fetch_data():

Fetches the relevant data from the database. No input needed. Output is the rows of title, release_year, budget, total_gross, and imdb_rating from the Movies table.

def save_profitability_txt(data):

Runs calculations about the profitability of each movie. Input is the data from fetch_data(). Outputs a txt file that displays how much each movie made or lost.

def save_rating_vs_profit_txt(data):

Runs calculations about the relationship between the imdb_rating a movie received and the profit it earned. Input is the data from fetch_data(). Outputs a txt file that talks about this relationship.

def save_average_profitability_per_distributor_txt():

Runs calculations about on average, how much profit a distributor received per movie. No input needed. Outputs a txt file that displays the average profit per movie for each distributor.

def plot_profitability_bar_chart(data):

Makes visualization about the profitability of each movie. Input is the data from fetch_data(). Outputs a png file that is a bar chart showing the budget vs the revenue per movie.

def plot_rating_vs_profit_scatter(data):

Makes visualization about the relationship between the imdb_rating a movie received and the profit it earned. Input is the data from fetch_data(). Outputs a png file that is a scatter plot showing this relationship.

def plot_distributor_avg_profitability_bar_chart():

Makes visualization about the average profit a distributor received per movie it made. No input needed. Outputs a png file that is a bar chart showing this calculation.

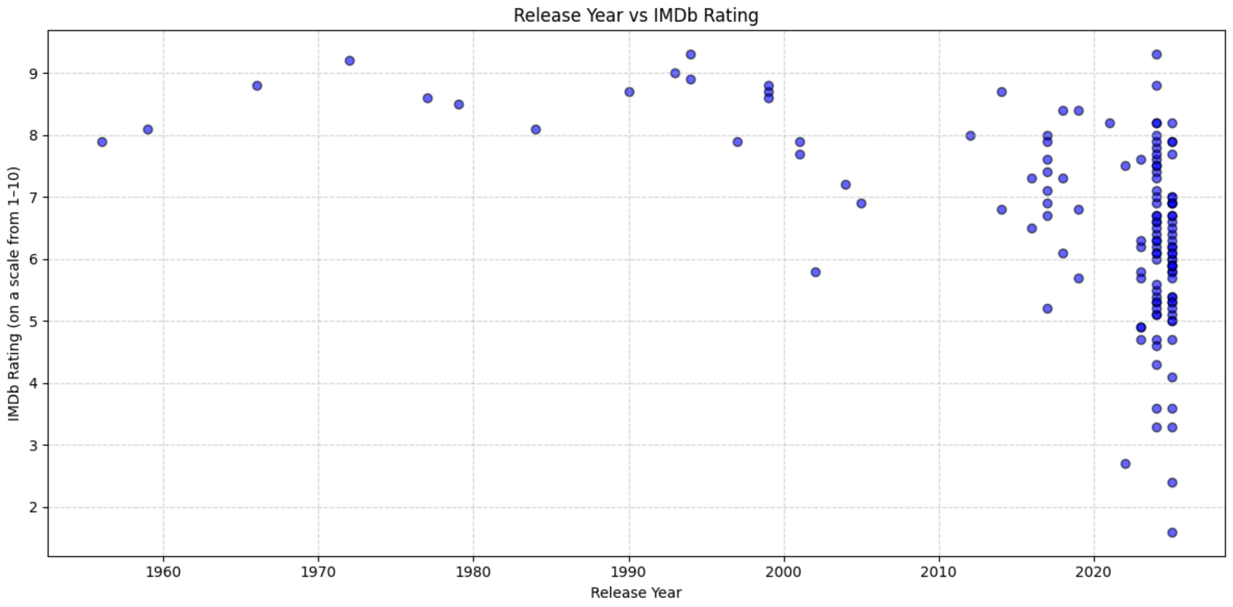
def main():

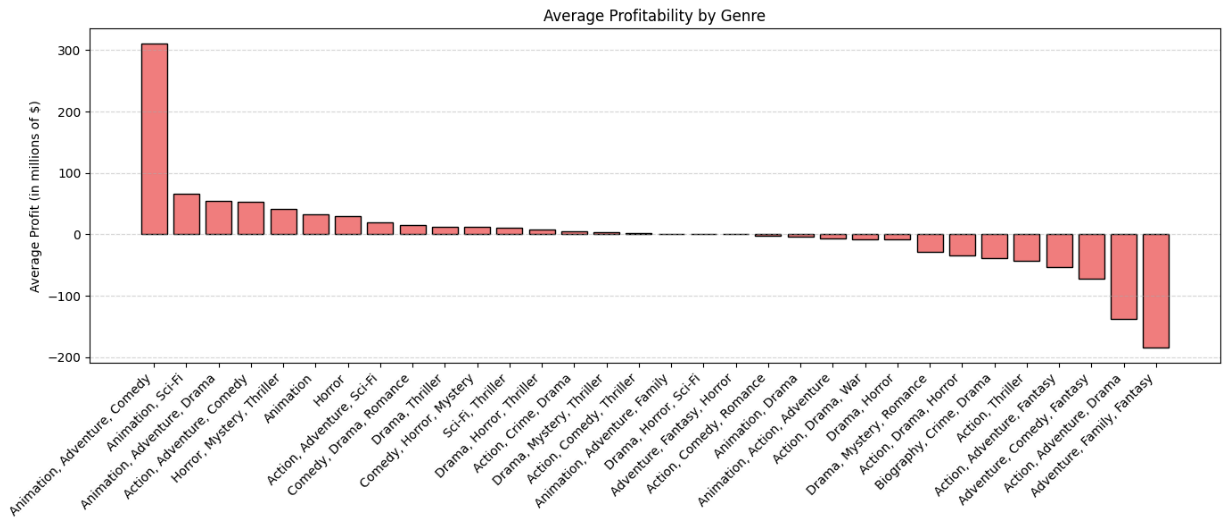
Runs the data processing script to make calculations and visualizations. No input needed. Outputs txt files and png files.

H. Resource documentation (You must also clearly document all resources you used. The documentation should be of the following form (20 points))

| Date | Issue Description | Location of Resource | Result |
|---|---|---|---|
| April 6th | how to properly make API calls to the API endpoints according to their documentation | TMDb + OMDb API Documentation and ChatGPT | The resources were detailed in telling us exactly what parameters we needed and how to make the API calls |
| April 9th | how to handle dirty data when web scraping the table from BoxOfficeMojo when a cell returned "-" | ChatGPT | ChatGPT gave helpful advice on parsing the rest of the row as normally as possible and then substituting the "-" value with None |
| April 19th | how to cleanly separate files into different modules while still having the code work | ChatGPT | ChatGPT gave helpful advice and insight on how to separate the data_gathering files cleanly and effectively |
| April 19th | how to make sure the database doesn't have any duplicated strings and duplicated tables | ChatGPT | ChatGPT gave helpful advice on how to best structure our database schema so that there weren't duplicated strings and merging redundant tables into one |
| April 21st | how to use matplotlib for the various graphs we wanted (horizontal bar graph, double bar graph, etc) | ChatGPT | ChatGPT gave code on how to represent the collected data into the graphs we wanted and formatted them accordingly to our instructions |

EXTRA CREDIT:


Release Year vs IMDb Rating
IMDb Rating (on a scale from 1–10)
Release Year


Average Profitability by Genre
Average Profit (in millions of $)
Animation, Adventure, Comedy
Animation, Sci-Fi
Animation, Adventure, Drama
Action, Adventure, Comedy
Horror, Mystery, Thriller
Animation
Horror
Action, Adventure, Sci-Fi
Comedy, Drama, Romance
Drama, Thriller
Comedy, Horror, Mystery
Sci-Fi, Thriller
Drama, Horror, Thriller
Action, Crime, Drama
Drama, Mystery, Thriller
Action, Comedy, Thriller
Animation, Adventure, Family
Drama, Horror, Sci-Fi
Adventure, Fantasy, Horror
Action, Comedy, Romance
Animation, Drama
Animation, Action, Adventure
Action, Drama, War
Drama, Horror
Drama, Mystery, Romance
Action, Drama, Horror
Biography, Crime, Drama
Action, Thriller
Action, Adventure, Fantasy
Adventure, Comedy, Fantasy
Action, Adventure, Drama
Adventure, Family, Fantasy