Candice Jones
Harsha Luitel
Kayla Ricumstrict

SI206 Final Project Report

Summary of Corrections Implemented from Grading Session Feedback

1. **Removed Duplicate String Data**
    - Duplicate string data, such as country names (e.g., "Japan") and genres (e.g., "Animation"), were removed throughout the database.
    - To achieve this, we created separate tables for Places (countries) and Genres and ensured values were normalized and stored uniquely.
2. **Corrected Year Data Type**
    - The year column was updated to store integers instead of strings. This change eliminated issues where "2024" was being counted as duplicate string data.
3. **Created a New Database**
    - A new database was created due to errors encountered after merging the final repository. One of the files retained outdated parameters, which caused the Movies table to include incorrect data (e.g., movies from 2015).
    - Troubleshooting efforts successfully removed the 2015 titles and reset the database. However, mismatched ID keys between the Movies table and related tables (e.g., Soundtracks) caused issues with visualizations.
    - After consulting with the graders, the decision was made to start a new database to resolve the ID mismatch and table reference issues completely.
4. **Converted columns with numbers stored as string in the database into integers**
    - In the Soundtracks and Songs database, the total_duration and song_length columns respectively were changed from being stored as text to integers in order to prevent duplicate strings.
5. **Split Genres into Individual Entries**
    - Genres were split when multiple genres were listed for a single movie. For example, a movie tagged as "Animation, Adventure, Comedy" now has each genre stored in its own entry in the Genres table.
6. **Ensure No Duplicate String Data Across Tables/Columns**
    - All tables and columns were reviewed to ensure no duplicate string data existed. This included fields like country names, genres, and movie titles.
    - The published_date column was converted from an ISO 8601 date-time string format to a REAL format representing the number of seconds since January 1, 1970, also known as UNIX epoch time. This conversion allowed us to not have duplicate string data for the published_date column in the Articles table.

    - Changed both Soundtracks and Songs tables in the database to reflect integers instead of strings for total_duration and song_length.
7. **Split CSV Files to Avoid Errors**
    - The CSV file was split into smaller csv files for each calculation. This change prevented errors caused by file size limits or conflicts during processing.

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

- OMDB API: http://www.omdbapi.com
- Spotify API: https://developer.spotify.com/documentation/web-api
- NewsAPI: https://newsapi.org/

The original goals for our project were finding the correlation between soundtrack popularity (from Spotify API) and media coverage (from News API). Another goal was to compare IMDb ratings (from OMDb API) with media coverage trends (from News API).

The data we planned to gather was different for all of them. For the OMDb API, we were going to get the data for the average IMDb ratings by genre or director, find the highest rated genres, find the correlation between box office revenue and IMDb ratings, and find the top 5 highest rated movies.

For the Spotify API, we initially planned to get the average song popularity for soundtracks by genre or movie, the number of songs per genre across all movies, and find the top 5 most listened to songs from a movie soundtrack. However, we realized when viewing the documentation for spotify API that the API doesn't allow for retrieving the amount of streams an album or song has. Therefore, we pivoted to retrieving the album and song lengths to complete calculations. We instead calculated the top 5 longest movie soundtracks pulled from the API.

For the News API, we were going to get the number of articles published per movie, and the top 5 movies with the highest media coverage.

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

The goals that were achieved in our project were many - our project successfully utilized multiple APIs to collect, analyze, and visualize data. These included the OMDb API, Spotify API, and the News API. For the OMDb API, we found the average film ratings with the search parameter "movie" from 2024, by different places in the world, through IMDb; additionally, we

were also able to find the average 2024 IMDb ratings by genre too. We then created a ‘Movies’ table in DB Browser for SQLite and separated the IMDb ratings by different tables called ‘Places’ and ‘Genre’, so there would be no duplicate string data. From there, we aggregated the total number of movies produced in each country in 2024, and while the range varied across different countries, the leading contributor, the United States, had a count of over 80 movies. To expand on this data, we then calculated the total number of movies per genre, where categories such as Animation and Drama lead the way. We created graphs to illustrate the data findings which are listed under the names, movies_by_country.png, and movies_by_genre.png.

For the Spotify API, we created a soundtrack table inside the existing SQLite Movies database. We then fetched movies from 2024 and searched for albums on Spotify using the movie titles, and inserted the soundtracks of those movies into the ‘soundtracks’ table. For every soundtrack, we also pulled the length (given in milliseconds) and converted it to seconds. Along with this, we fetched songs for each movie soundtrack from the ‘soundtracks’ table and inserted them into the Song table. Finally, we executed a select to join Movies and Soundtracks to find movies with soundtracks found in the Spotify API, which returned the top 5 movies (with movie in the title) in 2024 with the longest soundtracks and created a chart.

For the News API, we set up the ‘Articles’ and ‘Sources’ table in the database. We fetched and stored articles from NewsAPI related to a specific movie title, in 2024, from the News API and stored them in the ‘Articles’ table. We found the published date and the contents of the article that were related to the specific movie title and put it in the database. Also, for the ‘Sources’ table, we gathered the information about what source the article was coming from as well. More data was gathered to find the number of articles per movie and the IMDb Rating vs. the number of articles there are.

3. The problems that you faced & the lessons that were learned (10 points):

1. During our final repository merge, we encountered an issue where one of the files seemed to contain outdated parameters. As a result, the updated Movies table included movies from 2015, causing the data pull to differ from the expected results after running omdb.py. While troubleshooting, we successfully removed the 2015 titles and cleared the database. However, this created another issue where the shared integer keys had mismatched IDs. Although the tables were still linked, one of the additional visualizations became empty, and the ID keys for the Movies table no longer aligned with the Soundtracks table. We attempted multiple troubleshooting steps, but SQLite continued to confuse the table references, making the issue increasingly complex. This led us to debate whether we should start fresh with a new database across all files—generating new shared keys—or continue fixing the existing database.

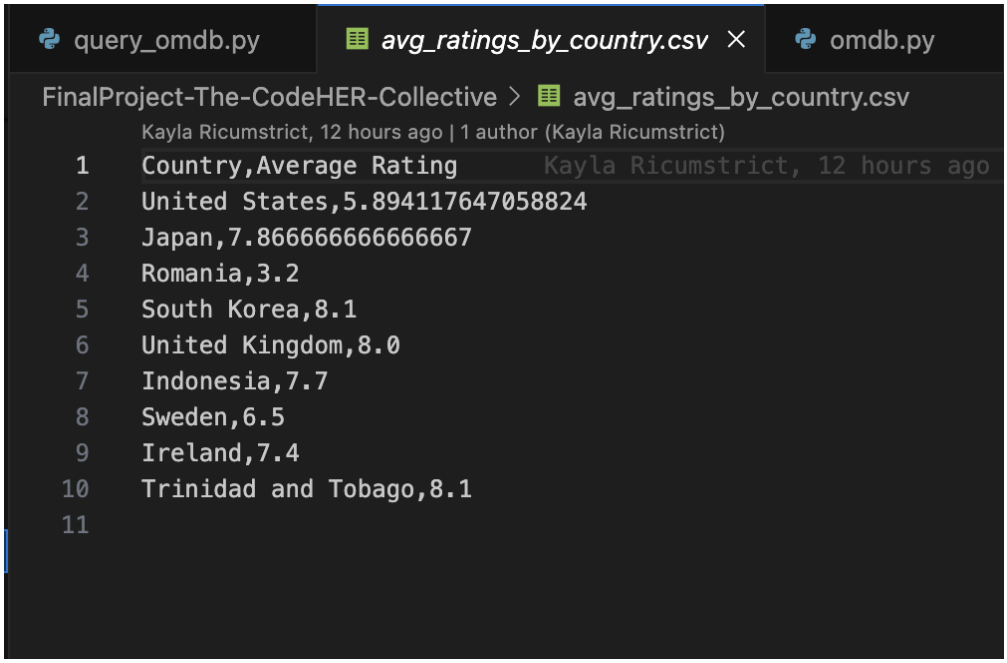
Ultimately, we consulted our graders during a grading session on Zoom. They advised us to create a new database rather than attempt to clean up the old one. Following their guidance, we rebuilt the database from scratch, ensuring all tables and keys were correctly aligned for the final submission.

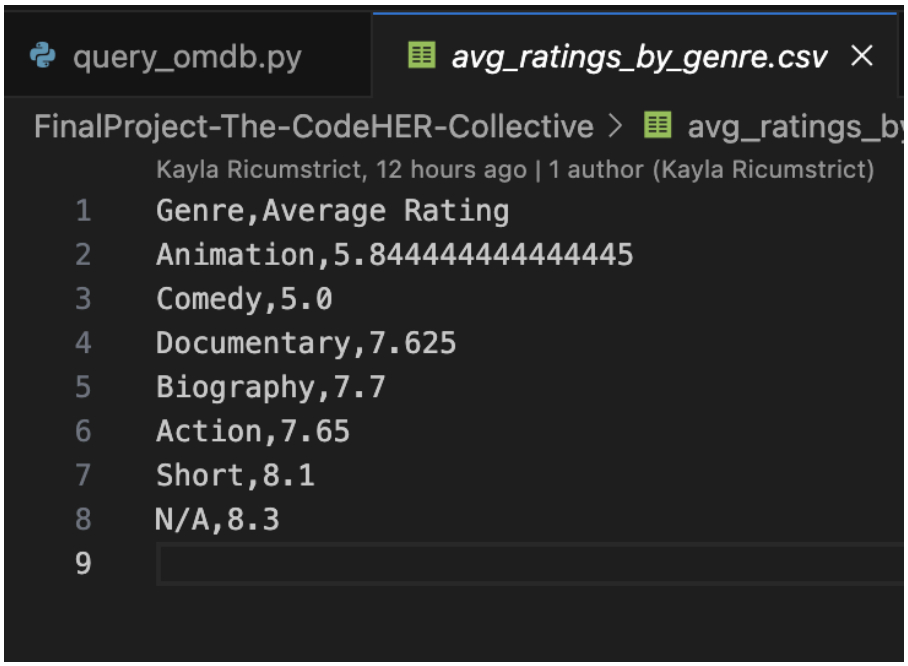
2. Ensuring there was no duplicate string data in each table or column presented a challenge. To resolve this issue, we separated the data into distinct lookup tables where necessary. For instance, tables such as ‘Places’ and ‘Genres’ were created to store unique values, which allowed us to reference them using foreign keys instead of duplicating string data across the database.

3. A significant issue we faced was exceeding the API rate limits by making too many requests in a short period, which caused the code to stop working and return errors. To address this, we initially created new API keys. However, we still encountered problems because the rate limits were repeatedly exhausted. This prompted us to implement an updated time sleep function, which introduced a delay between API calls. By adding a sleep timer, we ensured that the requests stayed within the API’s rate limits, preventing errors and allowing the code to function properly.

4. The published_date column was converted from an ISO 8601 date-time string format to a REAL format representing the number of seconds since January 1, 1970, also known as UNIX epoch time. This conversion is significant because UNIX epoch time serves as a universal reference point for timekeeping across various systems and applications, particularly within the Unix operating system. By storing dates in REAL format, we ensure consistency, enable efficient querying, and improve compatibility with time-based operations.

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





query_omdb.py | movies_articles_analysis.csv | omdb.py

FinalProject-The-CodeHER-Collective > movies_articles_analysis.csv

Kayla Ricumstrict, 6 hours ago | 1 author (Kayla Ricumstrict)

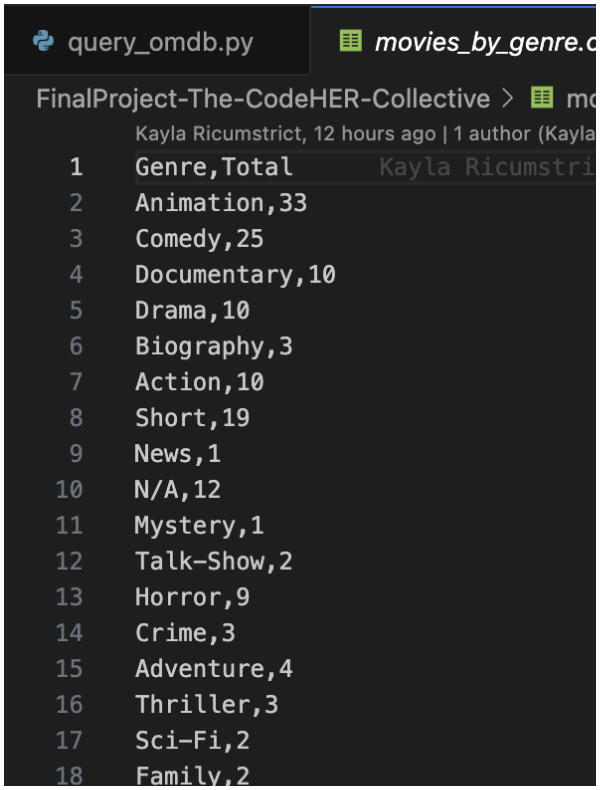
```
1   Movie Title,IMDb Rating,Article Count
2   Attack on Titan the Movie: The Last Attack,9.3,25
3   Buzz House: The Movie,3.2,25
4   Not Another Church Movie,2.2,25
5   The Garfield Movie,5.7,25
6   The Wrong Movie,0.0,19
7   The 4:30 Movie,6.0,16
8   Slave Play. Not a Movie. A Play.,5.3,7
9   The Day the Earth Blew Up: A Looney Tunes Movie,7.0,4
10  Doraemon the Movie: Nobita's Earth Symphony,6.6,2
11  Saving Bikini Bottom: The Sandy Cheeks Movie,3.7,2
12
```

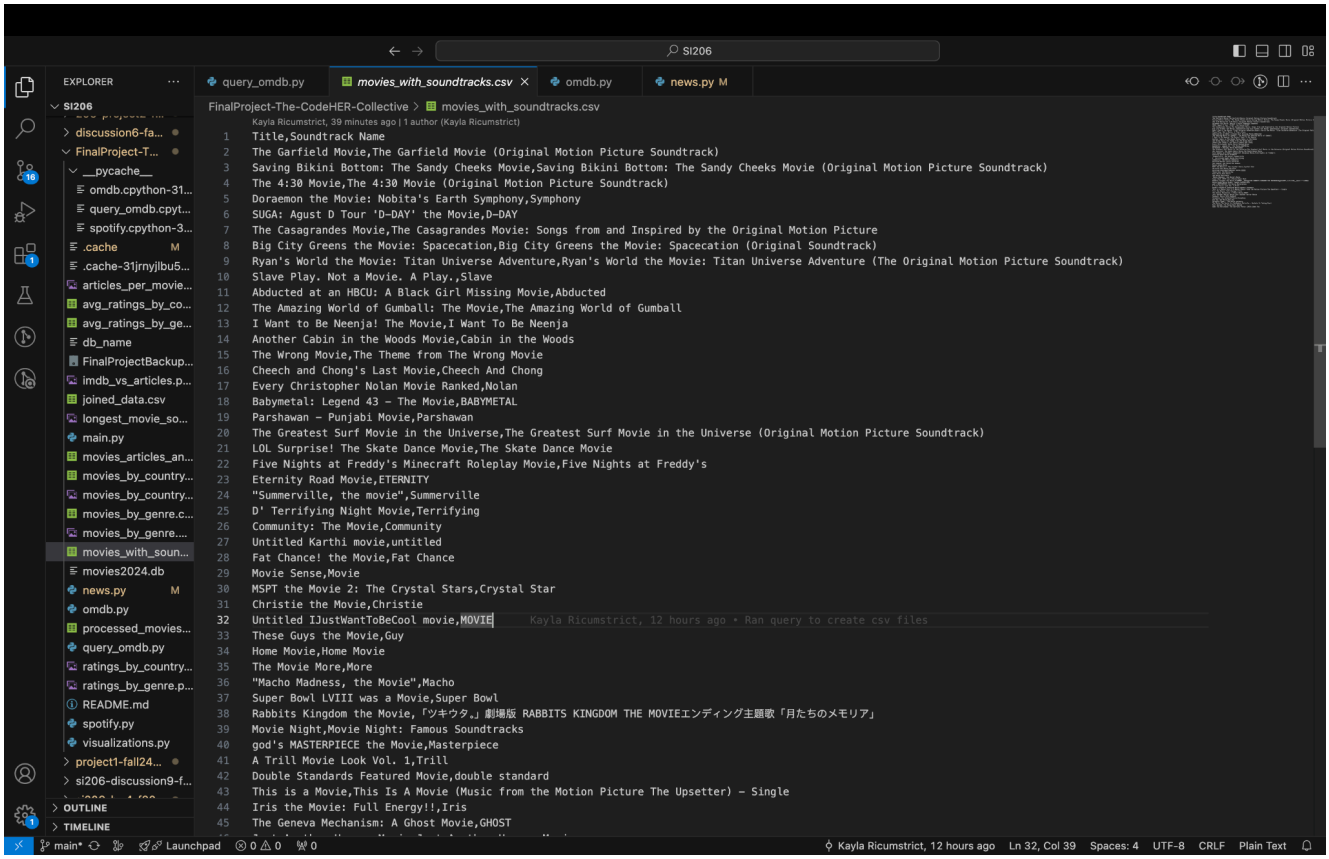
query_omdb.py | movies_by_country.csv

FinalProject-The-CodeHER-Collective > movies_by_

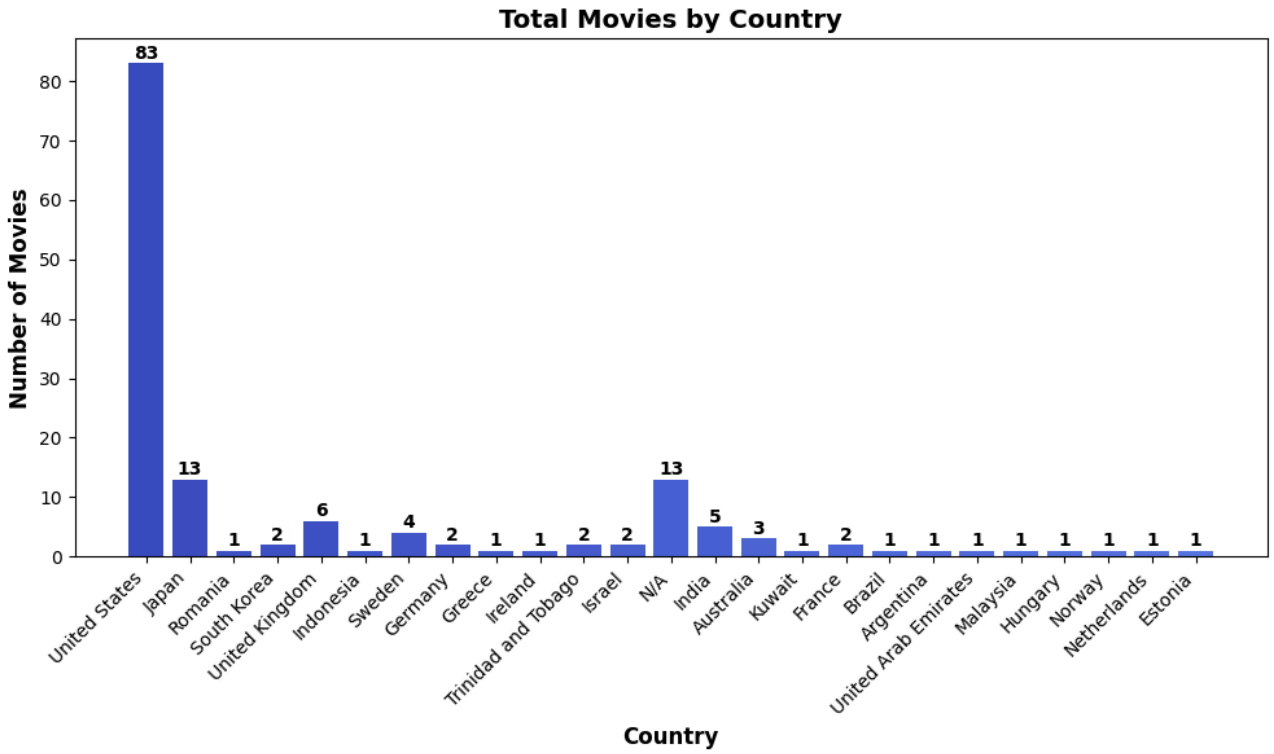
Kayla Ricumstrict, 12 hours ago | 1 author (Kayla Ricumstrict

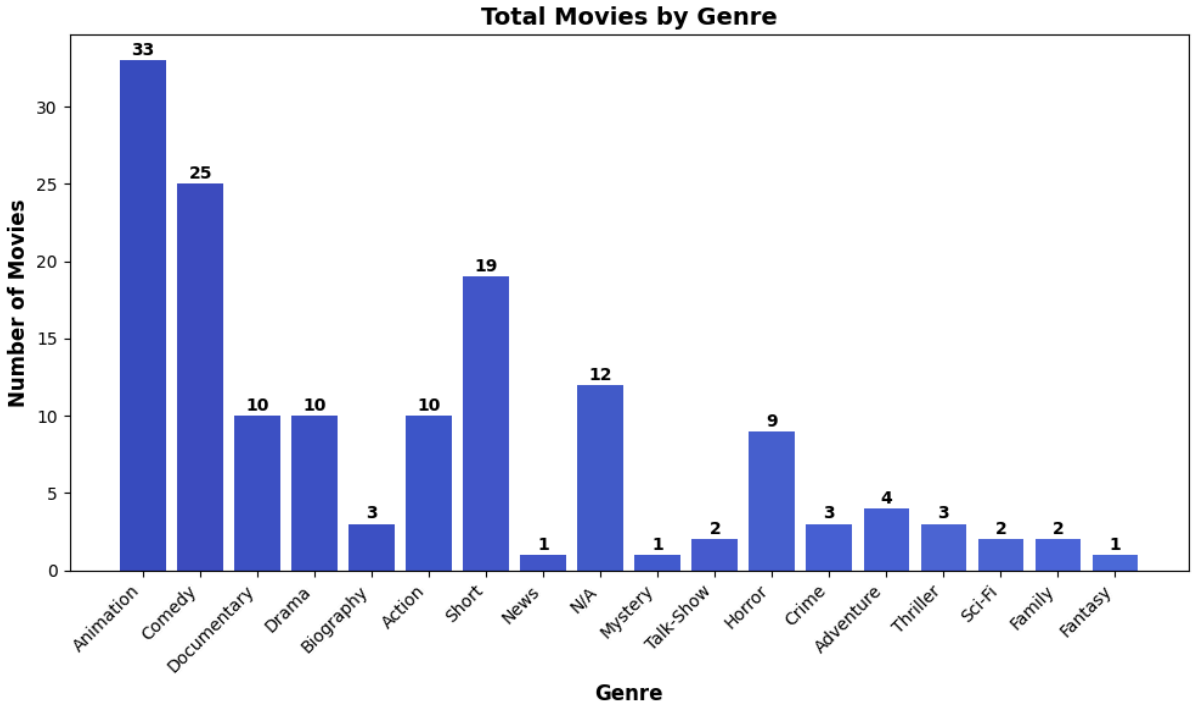
```
1   Country,Total
2   United States,83
3   Japan,13
4   Romania,1
5   South Korea,2
6   United Kingdom,6
7   Indonesia,1
8   Sweden,4
9   Germany,2
10  Greece,1
11  Ireland,1
12  Trinidad and Tobago,2
13  Israel,2
14  N/A,13
15  India,5
16  Australia,3
17  Kuwait,1
18  France,2
19  Brazil,1
20  Argentina,1
21  United Arab Emirates,1
22  Malaysia,1
23  Hungary,1
24  Norway,1
25  Netherlands,1
26  Estonia,1
27
```

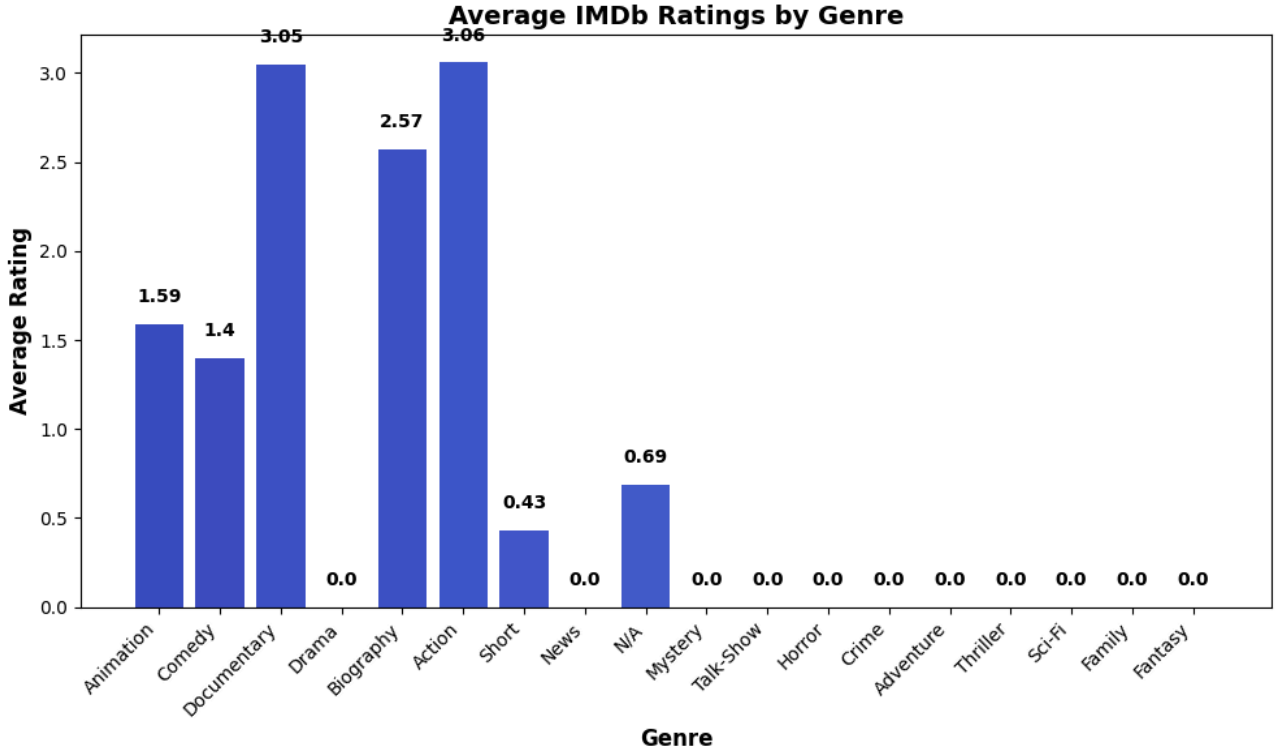

query_omdb.py
movies_by_genre.c
FinalProject-The-CodeHER-Collective > mo
Kayla Ricumstrict, 12 hours ago | 1 author (Kayla
1 Genre,Total
2 Animation,33
3 Comedy,25
4 Documentary,10
5 Drama,10
6 Biography,3
7 Action,10
8 Short,19
9 News,1
10 N/A,12
11 Mystery,1
12 Talk-Show,2
13 Horror,9
14 Crime,3
15 Adventure,4
16 Thriller,3
17 Sci-Fi,2
18 Family,2

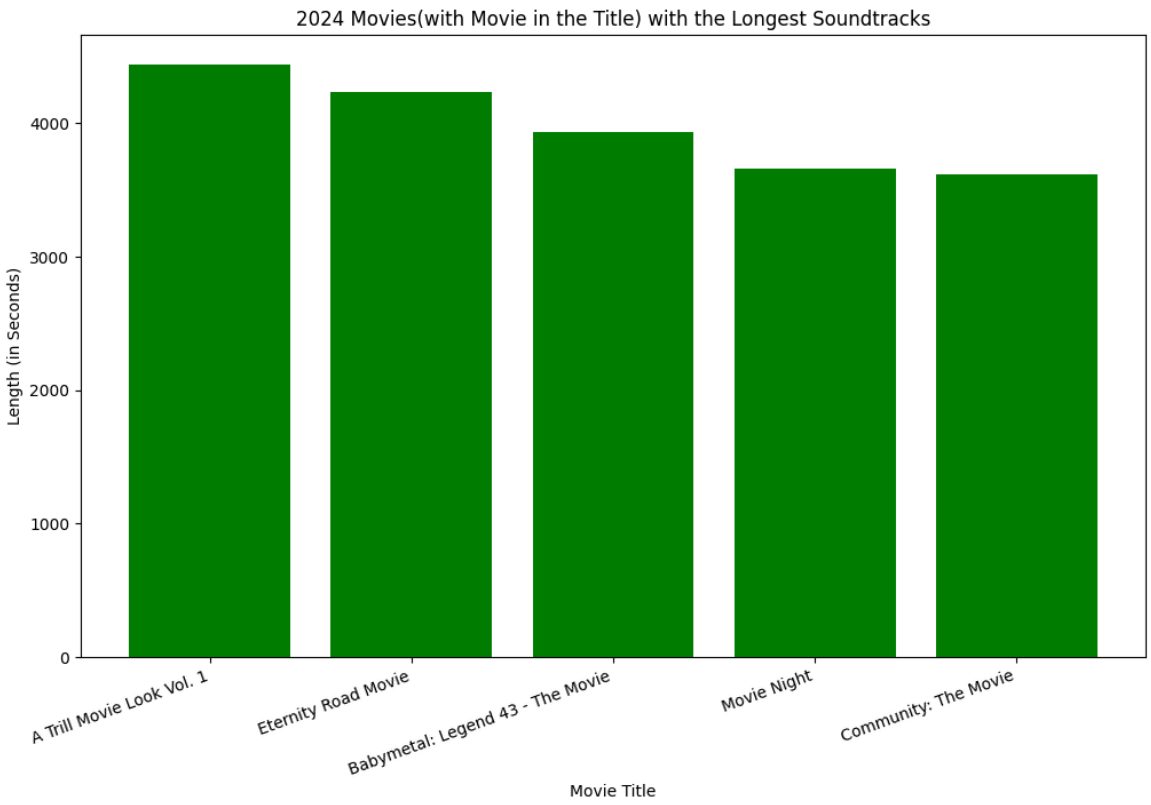

SI206
EXPLORER
query_omdb.py
movies_with_soundtracks.csv
omdb.py
news.py M
FinalProject-The-CodeHER-Collective > movies_with_soundtracks.csv
Kayla Ricumstrict, 39 minutes ago | 1 author (Kayla Ricumstrict)
1 Title,Soundtrack Name
2 The Garfield Movie,The Garfield Movie (Original Motion Picture Soundtrack)
3 Saving Bikini Bottom: The Sandy Cheeks Movie,Saving Bikini Bottom: The Sandy Cheeks Movie (Original Motion Picture Soundtrack)
4 The 4:30 Movie,The 4:30 Movie (Original Motion Picture Soundtrack)
5 Doraemon the Movie: Nobita's Earth Symphony,Symphony
6 SUGA: Agust D Tour 'D-DAY' the Movie,D-DAY
7 The Casagrandes Movie,The Casagrandes Movie: Songs from and Inspired by the Original Motion Picture
8 Big City Greens the Movie: Spacecation,Big City Greens the Movie: Spacecation (Original Soundtrack)
9 Ryan's World the Movie: Titan Universe Adventure,Ryan's World the Movie: Titan Universe Adventure (The Original Motion Picture Soundtrack)
10 Slave Play. Not a Movie. A Play.,Slave
11 Abducted at an HBCU: A Black Girl Missing Movie,Abducted
12 The Amazing World of Gumball: The Movie,The Amazing World of Gumball
13 I Want to Be Neenja! The Movie,I Want To Be Neenja
14 Another Cabin in the Woods Movie,Cabin in the Woods
15 The Wrong Movie,The Theme from The Wrong Movie
16 Cheech and Chong's Last Movie,Cheech And Chong
17 Every Christopher Nolan Movie Ranked,Nolan
18 Babymetal: Legend 43 – The Movie,BABYMETAL
19 Parshawan – Punjabi Movie,Parshawan
20 The Greatest Surf Movie in the Universe,The Greatest Surf Movie in the Universe (Original Motion Picture Soundtrack)
21 LOL Surprise! The Skate Dance Movie,The Skate Dance Movie
22 Five Nights at Freddy's Minecraft Roleplay Movie,Five Nights at Freddy's
23 Eternity Road Movie,ETERNITY
24 "Summerville, the movie",Summerville
25 D' Terrifying Night Movie,Terrifying
26 Community: The Movie,Community
27 Untitled Karthi movie,untitled
28 Fat Chance! the Movie,Fat Chance
29 Movie Sense,Movie
30 MSPT the Movie 2: The Crystal Stars,Crystal Star
31 Christie the Movie,Christie
32 Untitled IJustWantToBeCool movie,MOVIE Kayla Ricumstrict, 12 hours ago • Ran query to create csv files
33 These Guys the Movie,Guy
34 Home Movie,Home Movie
35 The Movie More,More
36 "Macho Madness, the Movie",Macho
37 Super Bowl LVIII was a Movie,Super Bowl
38 Rabbits Kingdom the Movie,「ツキウタ.」劇場版 RABBITS KINGDOM THE MOVIEエンディング主題歌「月たちのメモリア」
39 Movie Night,Movie Night: Famous Soundtracks
40 god's MASTERPIECE the Movie,Masterpiece
41 A Trill Movie Look Vol. 1,Trill
42 Double Standards Featured Movie,double standard
43 This is a Movie,This Is A Movie (Music from the Motion Picture The Upsetter) – Single
44 Iris the Movie: Full Energy!!,Iris
45 The Geneva Mechanism: A Ghost Movie,GHOST
OUTLINE
TIMELINE
main*
Launchpad
Kayla Ricumstrict, 12 hours ago
Ln 32, Col 39
Spaces: 4
UTF-8
CRLF
Plain Text

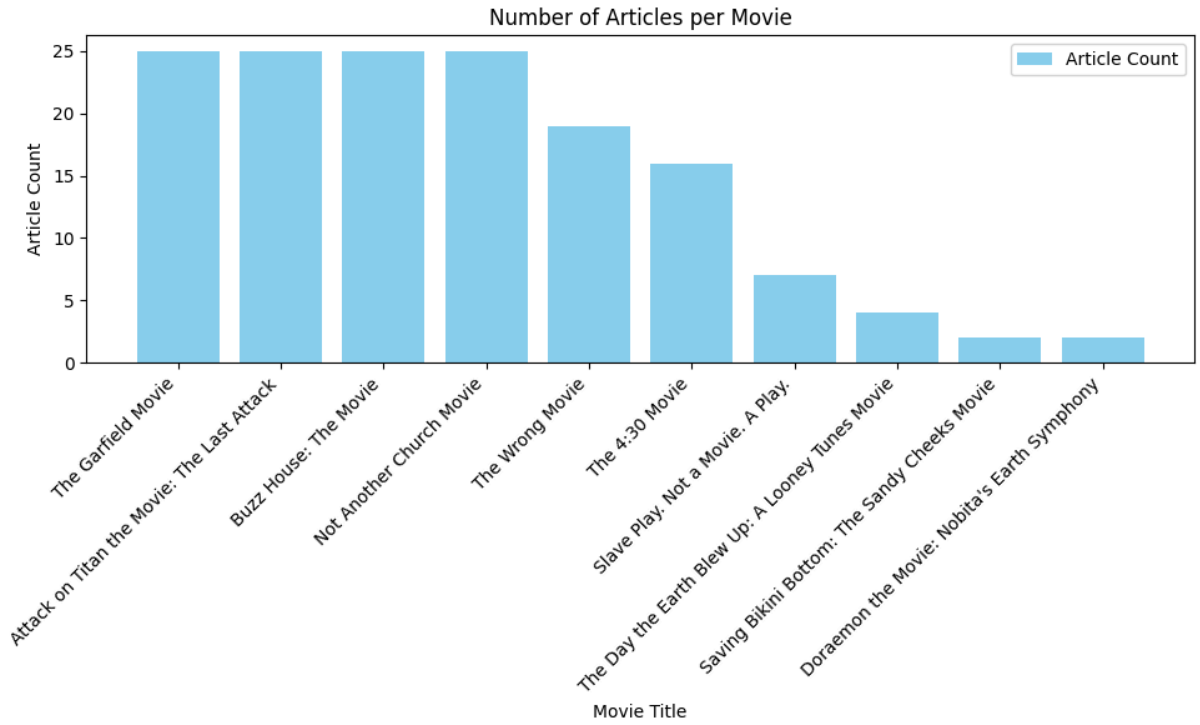
5. The visualization that you created (i.e. screenshot or image file) (10 points)

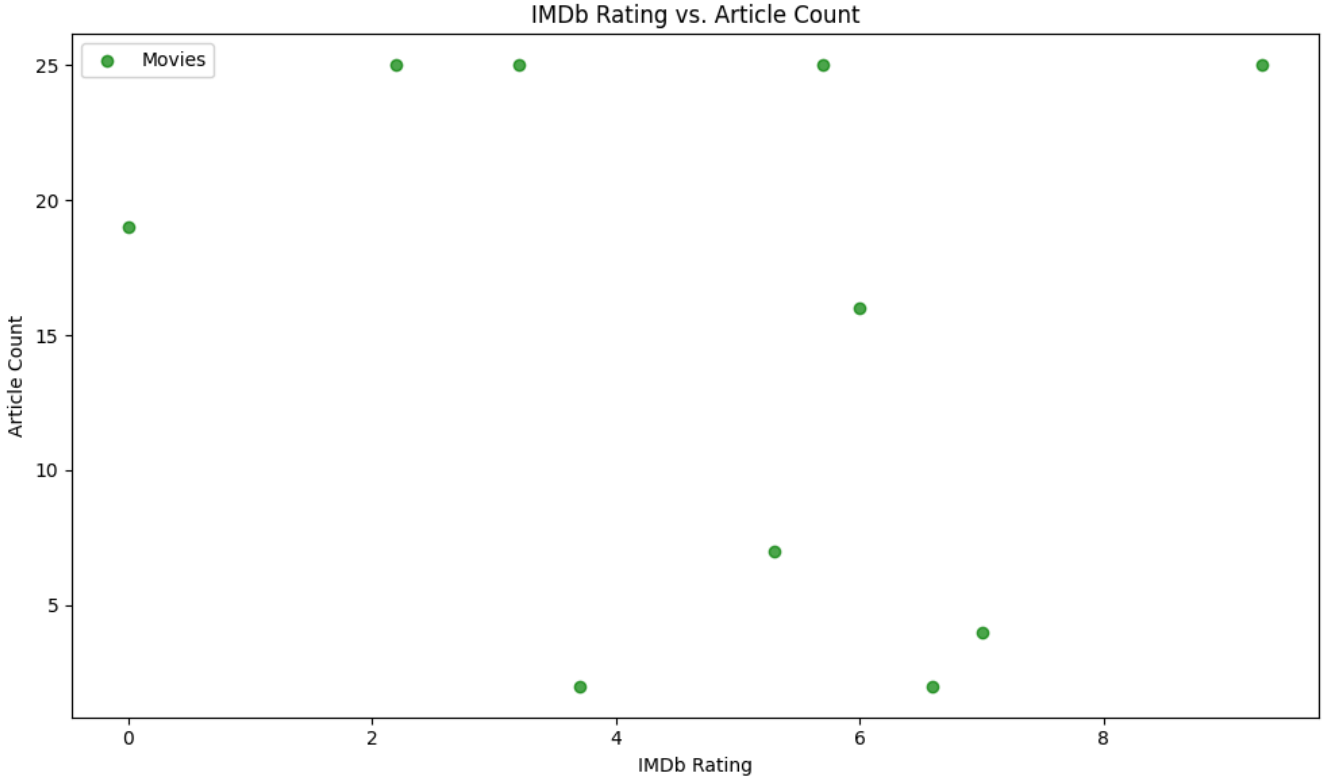





Average IMDb Ratings by Genre
Average Rating
Genre
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1.59
1.4
3.05
0.0
2.57
3.06
0.43
0.0
0.69
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
Animation
Comedy
Documentary
Drama
Biography
Action
Short
News
N/A
Mystery
Talk-Show
Horror
Crime
Adventure
Thriller
Sci-Fi
Family
Fantasy


2024 Movies(with Movie in the Title) with the Longest Soundtracks
Length (in Seconds)
0
1000
2000
3000
4000
A Trill Movie Look Vol. 1
Eternity Road Movie
Babymetal: Legend 43 - The Movie
Movie Night
Community: The Movie
Movie Title


Number of Articles per Movie
Article Count
Article Count
0
5
10
15
20
25
The Garfield Movie
Attack on Titan the Movie: The Last Attack
Buzz House: The Movie
Not Another Church Movie
The Wrong Movie
The 4:30 Movie
Slave Play. Not a Movie. A Play.
The Day the Earth Blew Up: A Looney Tunes Movie
Saving Bikini Bottom: The Sandy Cheeks Movie
Doraemon the Movie: Nobita's Earth Symphony
Movie Title


IMDb Rating vs. Article Count
Movies
Article Count
5
10
15
20
25
0
2
4
6
8
IMDb Rating

Project Overview:
The project integrates movie data, soundtracks, and news articles to perform analysis and generate visualizations. It uses SQLite for the database, OMDB API for movies, Spotify API for soundtracks, and NewsAPI for articles.

Setup Requirements

1. Python (version 3.8+ recommended).
2. Install Required Libraries:
    - Open a terminal or command prompt and run:
    - pip install requests spotipy matplotlib
3. APIs:
    - OMDB API Key: http://www.omdbapi.com/apikey.aspx
    - Spotify Client ID & Secret: Spotify Developer Dashboard
    - NewsAPI Key: https://newsapi.org
4. Database:
    - SQLite is used for the movies2024.db database.

Project File Structure
Here is how the project is organized:

Python Files:
omdb.py # Fetches movies from OMDB API and sets up the database.
spotify.py # Fetches movie soundtracks and song data from Spotify.
news.py # Fetches related news articles for movies.
query_omdb.py # Analyzes data and exports to CSV files.
visualizations.py # Generates graphs and visualizations.
main.py # Runs the overall project pipeline.
movies2024.db # SQLite database file (created automatically).

CSV's:
avg_ratings_by_genre.csv
avg_ratings_by_country.csv
movies_by_genre.csv
movies_by_country.csv
movies_with_soundtracks.csv
Movies_articles_analysis.csv

PNG FILES
articles_per_movie.png
imdb_vs_articles.png

longest_movie_soundtracks.png
movies_by_country.png
movies_by_genre.png
ratings_by_country.png
ratings_by_genre.png

Instructions to Run the Project

Step 1: Fetch Movie Data (OMDB API)

1. Run omdb.py at least four times to set up the database, and fetch movies.
    - What it does:
        - Sets up the database and creates tables (Movies, Places, Genres).
        - Fetches up to 25 movies from OMDB API for the year 2024.

Step 2: Fetch Soundtracks (Spotify API)

1. Run spotify.py at least four times to fetch soundtracks and songs.
    - What it does:
        - Creates the Soundtracks and Soundtrack_Songs tables.
        - Fetches soundtracks for movies and inserts them into the database.
        - Limits to 25 soundtracks and 25 songs per run.

Step 3: Fetch News Articles (NewsAPI)

1. Run news.py at least four times to fetch articles related to movies.
    - What it does:
        - Creates the Articles and Sources tables.
        - Fetches up to 25 articles per run for each movie title.
        - Implements rate limiting to avoid exceeding the API limit.

Step 4: Analyze Data and Export to CSV Files

1. Run query_omdb.py to analyze data and export results to csv files.
    - What it does:
        - Calculates and exports:
            - Average IMDb ratings by genre and country.
            - Total movie counts by genre and country.
            - Movies with soundtracks.
            - Articles found per movie.
        - Outputs the results as CSV files in the CSV/ folder.

Step 5: Generate Visualizations

1. Run visualizations.py to create visualizations.

- What it does:
  - Generates bar charts and scatter plots for:
    - Total movies by genre and country.
    - Average IMDb ratings by genre and country.
    - Number of articles per movie.
    - IMDb ratings vs. article count.
    - Top 5 movies with the longest soundtracks.
  - Saves the graphs as PNG files in the project directory.

Step 6: Verify Outputs
After confirming the above scripts, confirm the following:

1. Database
   - Open the movies2024.db file using DB Browser for SQLite
   - Verify that all tables (Movies, Soundtracks, Articles, etc.) are populated with data.
2. CSV Files:
   - Check the CSV files generated (e.g., movies_by_genre.csv, avg_ratings_by_country.csv, etc.) in the project folder.
3. Visualizations:
   - Verify that the PNG files are saved correctly in the project directory.

Troubleshooting Tips

API Rate Limits:
- OMDB API: Limited to 1000 requests/day.
- NewsAPI: 100 requests/day for free accounts. Use the sleep function (time.sleep(60)).
- Spotify API: Ensure token validity with get_token().

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

<u>OMDb.py:</u>

1. set_up_database(db_name):

Purpose:
- Initializes and sets up the SQLite database. This function creates tables for storing movies, places, and genres if they do not already exist in the database.

Inputs:

- db_name (string): The name of the database file where the data will be stored.

Outputs:

- Returns a tuple containing the cursor (cur) and connection (conn) to the SQLite database. The cursor is used to execute SQL commands, and the connection is used to manage the database connection.

2. get_or_create_lookup_id(cur, table, column, value)

Purpose:

- Retrieves the ID for a given value from a specified lookup table. If the value does not exist, it inserts the value into the table and then retrieves the new ID.

Inputs:

- cur (sqlite3.Cursor): Database cursor to execute SQL commands.
- table (string): Name of the database table.
- column (string): Column name in the table where the lookup should happen.
- value (string): The value to lookup or insert in the table.

Outputs:

- Returns the ID (integer) of the inserted or existing value in the lookup table.

3. fetch_movies_2024(cur, conn, max_insert=25)

Purpose: Fetches movie data from the OMDb API for movies released in 2024 and inserts the data into the database until the maximum specified inserts are reached.

Inputs:

- cur (sqlite3.Cursor): Database cursor to execute SQL commands.
- conn (sqlite3.Connection): Database connection to commit the transactions.
- max_insert (int, optional): Maximum number of movie records to insert. Default is 25.

Outputs:

- This function does not return any value but outputs information about the process via print statements, including any errors and the number of movies added.

4. main()

Purpose: Main function that orchestrates the database setup and the fetching of movie data. It sets up the database, fetches movie data, and closes the database connection.

Inputs:

- None

Outputs:

- None: This function controls the flow of the program and outputs process information to the console.

Spotify.py:

1. get_token()

Purpose: Establishes a connection to the Spotify API by obtaining an access token using the client credentials flow.

Inputs: None.

Outputs:

- token_info (str): An access token that can be used to authenticate requests to the Spotify API.

2. create_soundtrack_table(db_name)

Purpose: Creates two tables in the SQLite database: soundtracks to store movie soundtracks and soundtrack_songs to store individual songs in those soundtracks.

Inputs:

- db_name (str): The name of the database file to connect to.

Outputs:

- cur (sqlite3.Cursor): The cursor object to execute SQL commands.

- conn (sqlite3.Connection): The connection object to the database for committing transactions.

3. fetch_soundtrack_data(cur, conn, token)

Purpose: Fetches soundtrack data for movies released in 2024 from Spotify based on the movie titles stored in the database, and inserts this data into the soundtracks table.

Inputs:

- cur (sqlite3.Cursor): Cursor for executing SQL commands.
- conn (sqlite3.Connection): Connection to the database for committing transactions.
- token (str): Spotify API access token.

Outputs: None. However, the function modifies the database by inserting new records into the soundtracks table.

4. fetch_soundtrack_songs_data(cur, conn, token)

Purpose: Retrieves songs for each soundtrack from Spotify and stores them in the soundtrack_songs table in the database.

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- conn (sqlite3.Connection): Database connection.
- token (str): Spotify API access token.

Outputs: None. The function updates the database by inserting song details into the soundtrack_songs table.

5. main()

Purpose: The main function that orchestrates the connection setup, data fetching, and database updates.

Inputs: None.

Outputs: None. Mainly handles the control flow of the program and closes the database connection after operations.

News.py

1. setup_articles_table(db_name)

Purpose: Sets up the Articles and Sources tables in the database to store news articles and their associated sources.
Inputs:
    - db_name (str): Name of the SQLite database file.

Outputs: Returns a tuple containing:
    - conn (sqlite3.Connection): Database connection object.
    - cur (sqlite3.Cursor): Cursor object for executing SQL commands.

2. fetch_articles(cur, conn, fetch_limit=25)

Purpose:
Fetches articles from the NewsAPI related to movies released in 2024, and inserts the data (title, source, published date, and content) into the database. Implements rate-limiting to avoid exceeding the API request limit.

Inputs:
    - cur (sqlite3.Cursor): Cursor for executing SQL commands.
    - conn (sqlite3.Connection): Database connection for committing transactions.
    - fetch_limit (int, optional): The maximum number of articles to fetch. Default is 25.

Outputs: None. Updates the database by inserting records into the Articles table and prints progress information, such as successful insertions and rate-limit handling.

3. main()

Purpose: Coordinates the setup of the database tables and initiates the fetching of articles from NewsAPI.

Inputs: None.

Outputs: None. This function serves as the entry point, managing database setup, data fetching, and connection cleanup.

Query_omdb.py:

1. calculate_avg_rating_by_table(cur, lookup_table, fk_column, file, header_list)

Purpose: Calculates the average IMDb rating for movies grouped by a specified table (e.g., genres or countries) and exports the results to a CSV file.

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- lookup_table (str): The name of the lookup table (e.g., Genres or Places).
- fk_column (str): Foreign key column in the Movies table for the lookup table.
- file (str): Name of the CSV file where results will be saved.
- header_list (list): List of column headers for the CSV file.

Outputs: None. Writes the calculated average ratings into the specified CSV file.

2. total_movies_by_table(cur, lookup_table, fk_column, file, header_list)

Purpose: Counts the total number of movies for each value in a specified lookup table (e.g., genres or countries) and exports the results to a CSV file.

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- lookup_table (str): The name of the lookup table (e.g., Genres or Places).
- fk_column (str): Foreign key column in the Movies table.
- file (str): Name of the CSV file where results will be saved.
- header_list (list): List of column headers for the CSV file.

Outputs: None. Writes the total movie counts into the specified CSV file.

3. join_movies_and_soundtracks(cur, file)

Purpose:
Retrieves and joins movie titles with their associated soundtracks from the database, then exports the results to a CSV file.

Inputs:

- cur (sqlite3.Cursor): Database cursor.

- file (str): Name of the CSV file where results will be saved.

Outputs: None. Writes the joined movie and soundtrack data into the specified CSV file.

4. articles_and_movies(cur, file)

Purpose: Analyzes and joins movie titles, IMDb ratings, and the number of related articles. Outputs this data into a CSV file.

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- file (str): Name of the CSV file where results will be saved.

Outputs: None. Writes the analysis of articles and movie data into the specified CSV file.

5. main()

Purpose: Coordinates the execution of all data analysis functions and exports results to CSV files.

Inputs: None.

Outputs: None. Writes multiple outputs to CSV files and closes the database connection.

Visualizations.py

1. calculate_aggregate_by_table(cur, lookup_table, fk_column, aggregate="COUNT(*)")

Purpose: Calculates an aggregate value (e.g., count or average) for movies grouped by a lookup table (e.g., genres or countries).

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- lookup_table (str): The name of the lookup table (e.g., Genres or Places).
- fk_column (str): Foreign key column in the Movies table.
- aggregate (str, optional): The SQL aggregate function to use. Default is COUNT(*).

Outputs: Returns a dictionary where the keys are lookup table values (e.g., genre names) and the values are the calculated aggregates.

2. plot_bar_chart(data, title, xlabel, ylabel, fig_name)

Purpose: Generates a bar chart from the provided data with labels and annotations.

Inputs:

- data (dict): Dictionary containing labels (keys) and values for the chart.
- title (str): Title of the chart.
- xlabel (str): Label for the x-axis.
- ylabel (str): Label for the y-axis.
- fig_name (str): Filename for saving the chart as a PNG.

Outputs: None. Displays and saves the chart as a PNG file.

3. longest_movie_soundtracks_chart(cur)

Purpose: Creates a bar chart displaying the top 5 movies (with “movie” in the title) that have the longest soundtracks.

Inputs:

- cur (sqlite3.Cursor): Database cursor.

Outputs: None. Generates and saves the chart as a PNG file named longest_movie_soundtracks.png.

4. articles_per_movie_chart(cur)

Purpose: Generates a bar chart showing the number of articles per movie.

Inputs:

- cur (sqlite3.Cursor): Database cursor.

Outputs: None. Displays and saves the chart as a PNG file named articles_per_movie.png.

5. imdb_ratings_and_articles(cur)

Purpose: Creates a scatter plot showing the relationship between IMDb ratings and the number of articles for each movie.

Inputs:
- cur (sqlite3.Cursor): Database cursor.

Outputs: None. Displays and saves the scatter plot as a PNG file named imdb_vs_articles.png.

6. main()

Purpose: Orchestrates the execution of visualization functions to generate charts for movie data analysis.

Inputs: None.

Outputs: None. Generates multiple visualizations and saves them as PNG files.

Main.py

1. set_up_database(db_name)

Purpose:
Sets up the SQLite database and creates the necessary tables (Movies, Places, Genres, Soundtracks, Articles, etc.).

Inputs:
- db_name (str): Name of the SQLite database file.

Outputs: Returns a tuple containing:
- cur (sqlite3.Cursor): Cursor for executing SQL commands.
- conn (sqlite3.Connection): Database connection object.

2. fetch_movies_data(cur, conn, max_insert=25)

Purpose: Fetches movie data for the year 2024 from the OMDb API and inserts the data into the database.

Inputs:

- cur (sqlite3.Cursor): Cursor for executing SQL commands.
- conn (sqlite3.Connection): Database connection to commit changes.
- max_insert (int, optional): Maximum number of movies to fetch. Default is 25.

Outputs:

None. Updates the Movies, Places, and Genres tables in the database.

3. fetch_soundtracks_data(cur, conn, token)

Purpose: Fetches movie soundtracks from the Spotify API and inserts the data into the database.

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- conn (sqlite3.Connection): Database connection.
- token (str): Access token for the Spotify API.

Outputs: None. Updates the soundtracks and soundtrack_songs tables.

4. fetch_news_articles(cur, conn, fetch_limit=25)

Purpose: Fetches articles related to movie titles from the NewsAPI and stores them in the database.

Inputs:

- cur (sqlite3.Cursor): Database cursor.
- conn (sqlite3.Connection): Database connection.
- fetch_limit (int, optional): Maximum number of articles to fetch. Default is 25.

Outputs: None. Updates the Articles and Sources tables in the database.

5. generate_visualizations(cur)

Purpose:
Generates visualizations based on the aggregated data stored in the database, including bar charts and scatter plots.

Inputs:
- cur (sqlite3.Cursor): Database cursor.

Outputs: None. Saves visualizations as PNG files in the project directory.

6. analyze_and_export_data(cur)

Purpose: Performs data analysis to calculate and export results such as average IMDb ratings, total movies, and joined data for soundtracks and articles. Results are saved to CSV files.
Inputs:
- cur (sqlite3.Cursor): Database cursor.

Outputs: None. Generates CSV files containing the results.

7. main()

Purpose: Orchestrates the entire workflow of the project, including database setup, data fetching, analysis, and visualization generation.

Inputs: None.

Outputs: None. Executes all steps of the project and prints progress updates to the console.

8. All resources you used. The documentation should be of the following form (20 points)

Date	Issue	Description	Location of Resource	Result (did it solve the issue?)
11/22/2024	None	Used ChatGPT to construct the structure of the project including all files and some of the functions.	chatgpt.com	Chat was able to make suggestions on how to structure each file based on the purpose of our project and the API's we selected. This allowed us to build the project from the "ground up".
11/23/2024	Issue with Using Spotipy	Followed the spotipy documentation to set up connect to Spotipy API. The get_access_token function kept giving a deprecation warning saying that as_dict is set to True and support for it will be removed.	Stackoverflow, Spotipy documentation	We set as_dict to false after looking at the documentation and researching stack overflow. Instead of accessing the token by retrieving the first item in the dictionary, it returned the token string by itself.
11/29/2024	Updating fetch and store movie functions to allow at least 100 entries	Used StackOverflow initially to structure the fetch and store movie functions, but there were still errors. ChatGPT was able to help navigate out of issues	stackoverflow.com and chatgpt.com	This allowed a fetch of at least 100 entries.
11/29/2024	Updated code and added movies.db	We covered this in Week 12 during Lecture and during our Discussion Section	Discussion 12.pptx	Was able to successfully create database in DB Browser SQLite

Date	Issue	Description	Location of Resource	Result (did it solve the issue?)
12/2/2024	“Drop Table” function removal	Experienced challenges with altering code because we had to remove the drop table function out of the python files. We needed the help of chatgpt to help navigate the errors.	chatgpt.com	We were able to successfully drop the tables, and keep the overall structure and functionality of the code intact.
12/2/2024	Limited counter to only 25 items per script run	Our functions to limit each run to 25 items kept failing, so we needed assistance from chatgpt to help us correct.	chatgpt.com	We were able to understand the implementation of the added counter into the fetch function, as well as the main function for each python file.
12/10/2024	We had our calculations in their perspective python files and not separate as the rubric outlined. Needed assistance restructuring the project to adhere to these guidelines.	Used chat.gpt to help us figure out to include a query file which we used for visualizations and such, in an effort to follow the rubric	chatgpt.com	This helped us separate the pertinent files and solved the issue.
12/12/2024	shared integer keys had mismatched ID numbers.	They were still sharing but one of the extra visualizations became empty, and the id keys for the movies table were not matching with the soundtracks table. We tried different troubleshooting steps but SQLite	Graders in the grading session zoom meeting	Ian said it is easier to create a new database then to try and troubleshoot the current, so we changed our database. It solved our main issue we were having. The solution did take some time to do though because

Date	Issue	Description	Location of Resource	Result (did it solve the issue?)
		was still confusing the table references. This made us question if we should just start a new database across all files, for final submission which would create new shared keys and such? Or keep trying to fix the old one?		we had to make sure our code was correct.
12/14/2024	Calculations were all in the same csv file, which would lead to errors f exported	All of the calculations in quesry_omdb.py were in one csv file although they were structured differently with different column names. We were told by the graders during our zoom grading session to separate them.	Graders in the grading session zoom meeting, Runstone Academy Reading- Files	The calculations were broken up into csv files by writing the code to create and write to the csv file with each function completing a calculation instead of a separate function for writing.
12/14/2024	Verification of duplicate strings	In order to verify if we had any duplicate strings, we used chatgpt to help us determine if there were any in our tables in the movies2024.db. We were able to ascertain that they were, so we had to rename the tables, update the schema in SQLite(DB Browser), drop the old tables, and then update the function in the	We started off using stackoverflow but this left a little confusion, so was able to utilize chatgpt to provide step by step instructions.	This resolved our issue. We were able to rid all tables of duplicate strings, and used integers instead.

Date	Issue	Description	Location of Resource	Result (did it solve the issue?)
		python files so that integers are loaded and not strings		
12/15/2024	The News API kept outputting the 429 error which meant that we reached our data request limit. Every data run resulted in the error messages, and when a new API key was used it outputted no data at all, and then reverted back to 429 error code.	Was able to navigate out of this issue by referring to the Lecture slides from Week 14 in which Dr. Ericson went over what to do if there was a data request limit.	ProjectExWithVis-v11.pdf	We were able to implement the time sleep function into the news.py, use a new API key, and it corrected the issue.