

JAS Final Project Report

Github Link: <https://github.com/aaminaah/Final-Project/tree/main>

1. Goals for the project, including what APIs you planned to work with and what data you planned to gather

We planned on using the Spotify API, Google Books API, and IMDB API. We wanted to collect movie ratings and movie genre ratings from IMDB, the number of streams by each artist and genres of music from Spotify, and book ratings and genres from Google Books. We wanted to use this data to find the average ratings of movie genres, average artist streams per music genre, and average ratings for book genres. We also planned to use Matplotlib to create bar graphs of these calculations.

2. Goals that were achieved, including what APIs/websites you actually worked with and what data you did gather

Spotify: We were able to find music genres and artist names using the Spotify API, but could not find the number of streams of a given track. This made it impossible to find the average number of artist streams per genre. Additionally, Spotify's genre assignment system is hyper-specific so few artists shared a genre to calculate an average from. Instead, I found artists' names, top tracks, and popularity numbers, then calculated the most popular artists' popularity numbers and most popular artists' top tracks.

Books:

- Successfully retrieved book titles, authors, genres, and average ratings from the Google Books API.
- Calculated average ratings for various book genres.
- Analyzed the most frequent genres and their corresponding average ratings.

TMDB: We were not able to get access to the IMDB API for free due to new regulations and IMDB's data now being stored by Amazon. The IMDB API access Data was very complicated with unclear documentation which was very confusing to use. We instead found another free API from an open source database of movies called TMDB with IMDB ratings that was much more hassle free in giving access to the API and contained all the information we needed including the IMDB ratings. From this we were successful in collecting IMDB, release year, and genre for over 100 movies (to be exact 104 movies). With this data I calculated the average rating of movies by genre. Meaning collects the average rating of a genre by utilizing all the movies that belong to that genre. I then was able to successfully create a bar graph that displayed the average rating of each genre.

3. Problems you faced

Spotify:

It was initially difficult to access the API for Spotify and determine what information came from where within its returned data. Additionally, gathering a list of artist IDs to give to Spotify's API was a time-consuming process. Making sure the database name could be changed also took time to reconfigure so all the members' functions could work at the same time.

Google Books:

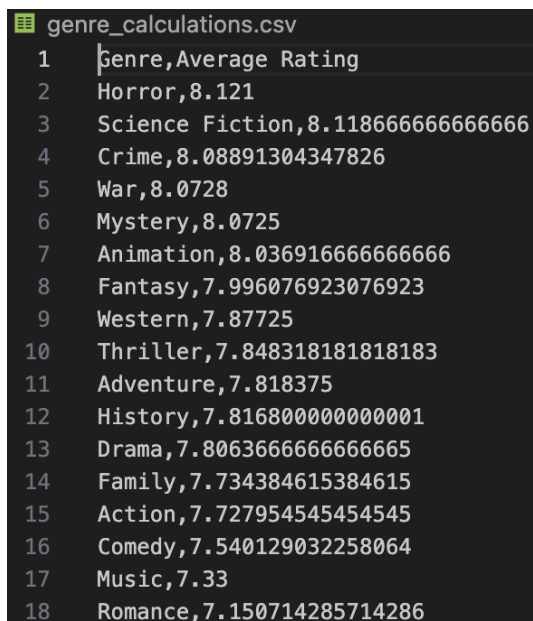
- The API returned inconsistent genre tags for some books, requiring additional data cleaning.
- Limited book entries for certain genres reduced the sample size for accurate calculations.

TMDB:

Due to the inability of accessing the IMDB API the switch was made to use the TMDB API/ Another struggle was handling duplicate string data in the initial genres column as there repeats of genres. We also struggled as a group when we combined our databases into one combined database as the Google Books API would go over the 25 limit. Another thing that was difficult when it came to the TMDB API was making sure that each time the API would insert new data into the database not just repeat the same data entries over again which is why we added the Progress table so we knew exactly what was going on easily.

4. Screenshot of the calculations from the data in the database

TMDB:



```
genre_calculations.csv
1  Genre,Average Rating
2  Horror,8.121
3  Science Fiction,8.118666666666666
4  Crime,8.08891304347826
5  War,8.0728
6  Mystery,8.0725
7  Animation,8.036916666666666
8  Fantasy,7.996076923076923
9  Western,7.87725
10 Thriller,7.848318181818183
11 Adventure,7.818375
12 History,7.816800000000001
13 Drama,7.806366666666665
14 Family,7.734384615384615
15 Action,7.727954545454545
16 Comedy,7.540129032258064
17 Music,7.33
18 Romance,7.150714285714286
```

Google Books:

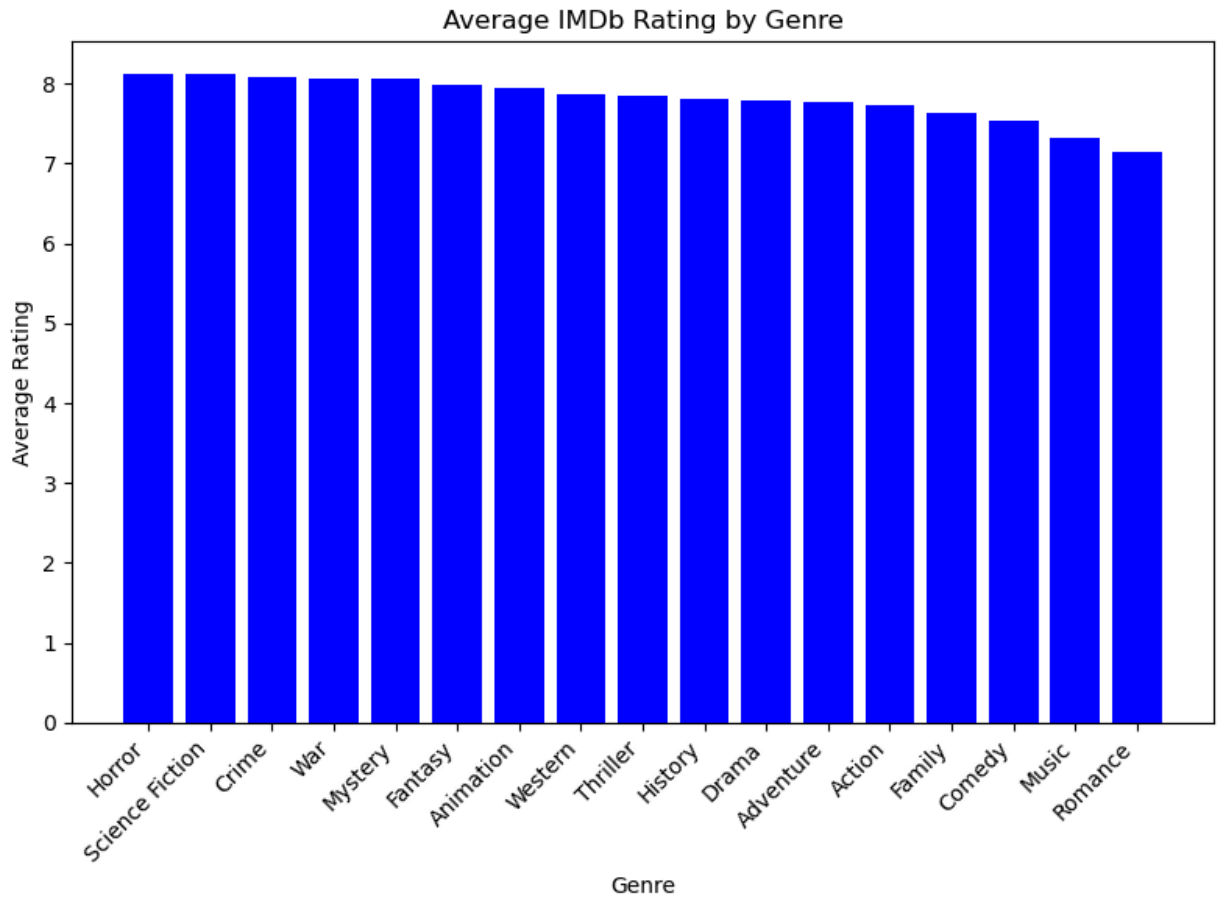
```
book_calculations.csv
1 Average Rating of All Books,4.375
2 Genre,Average Rating
3 Fiction,4.333333333333333
4 Games & Activities,4.0
5 History,4.125
6 Juvenile Fiction,4.5
7 Literary Criticism,5.0
8 Poetry,5.0
9
```

Spotify:

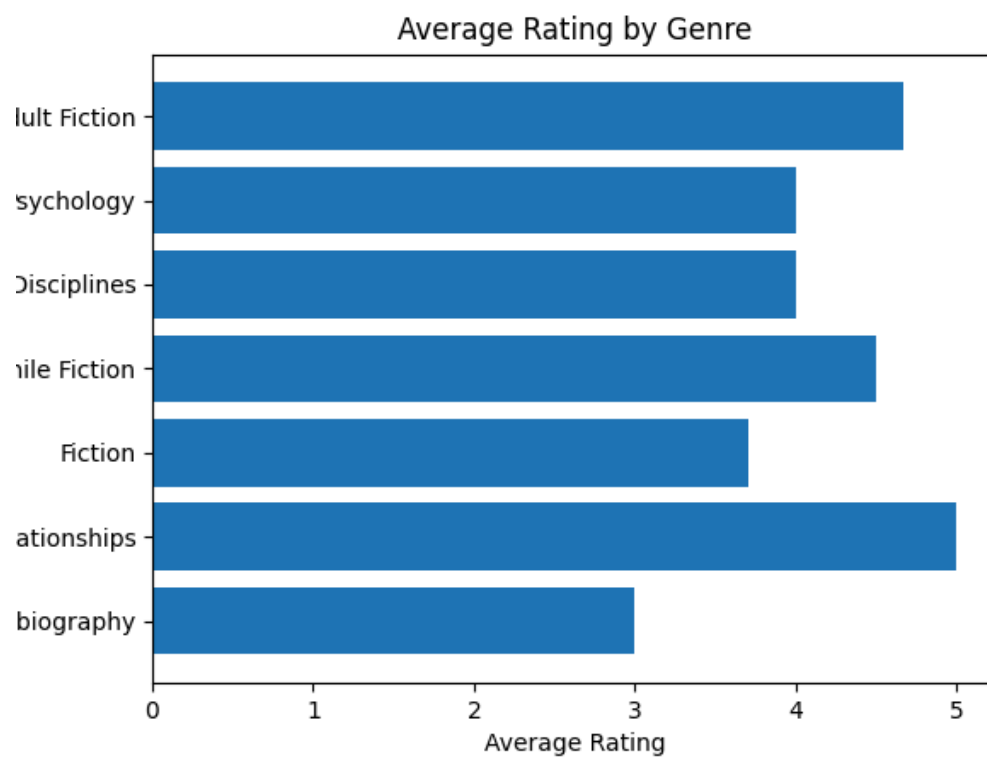
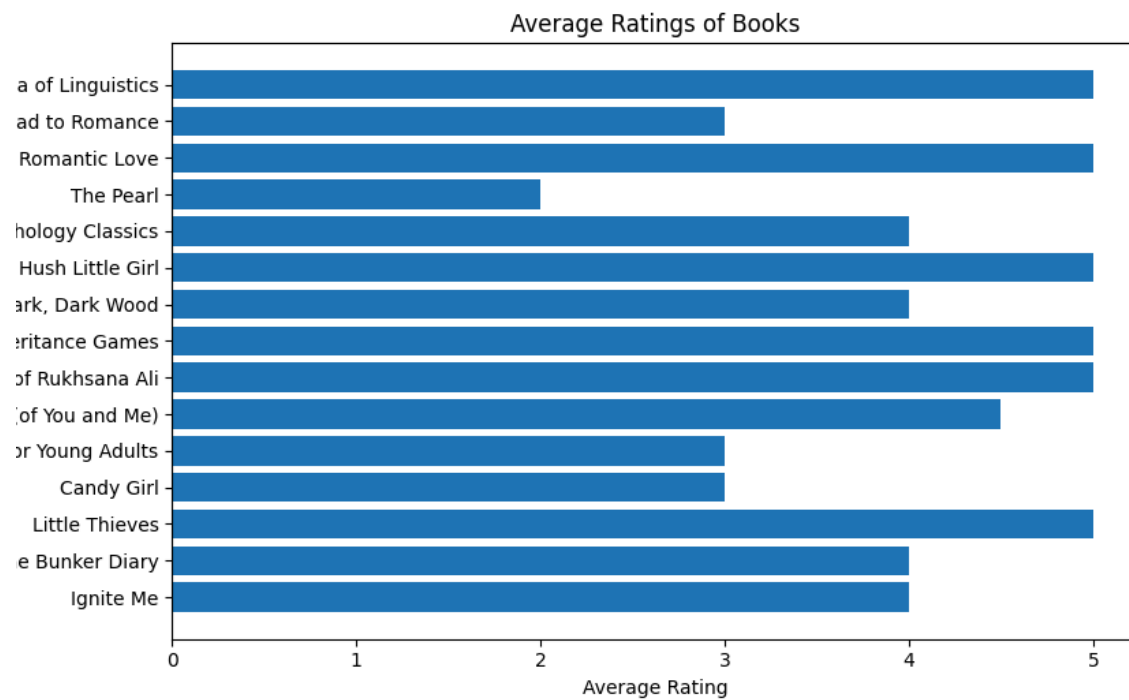
```
spotify_data.csv
1 id,name,popularity,top_track
2 06HL4z0CvFAxyc27GXpf02,Taylor Swift,100,Cruel Summer
3 2YZyLoL8N0Wb9xBt1NhZWg,Kendrick Lamar,97,Like That
4 0du5cEVh5yTK9QJze8zA0C,Bruno Mars,96,Die With A Smile
5 6qqNVTkY8uBg9cP3Jd7DAH,Billie Eilish,96,BIRDS OF A FEATHER
6 66CXWjxzNUsdJxJ2Jdwvnr,Ariana Grande,96,Santa Tell Me
7 3TVXtAsR1Inumwj472S9r4,Drake,96,One Dance
8 4q3ewBCX7sLwd24euuV69X,Bad Bunny,96,Qué Pasa...
9 74KM79TiuVKeVCqs8QtB0B,Sabrina Carpenter,94,Espresso
10 4V8LLVI7PbaPR0K2TGSxFF,"Tyler, The Creator",94,See You Again (feat. Kali Uchis)
11 7tYKF4w9nC0nq9CsPZTHyP,SZA,93,luther (with sza)
12 7dGJo4pcD2V6oG8kP0tJRR,Eminem,93,Without Me
```

5. Visualizations you created

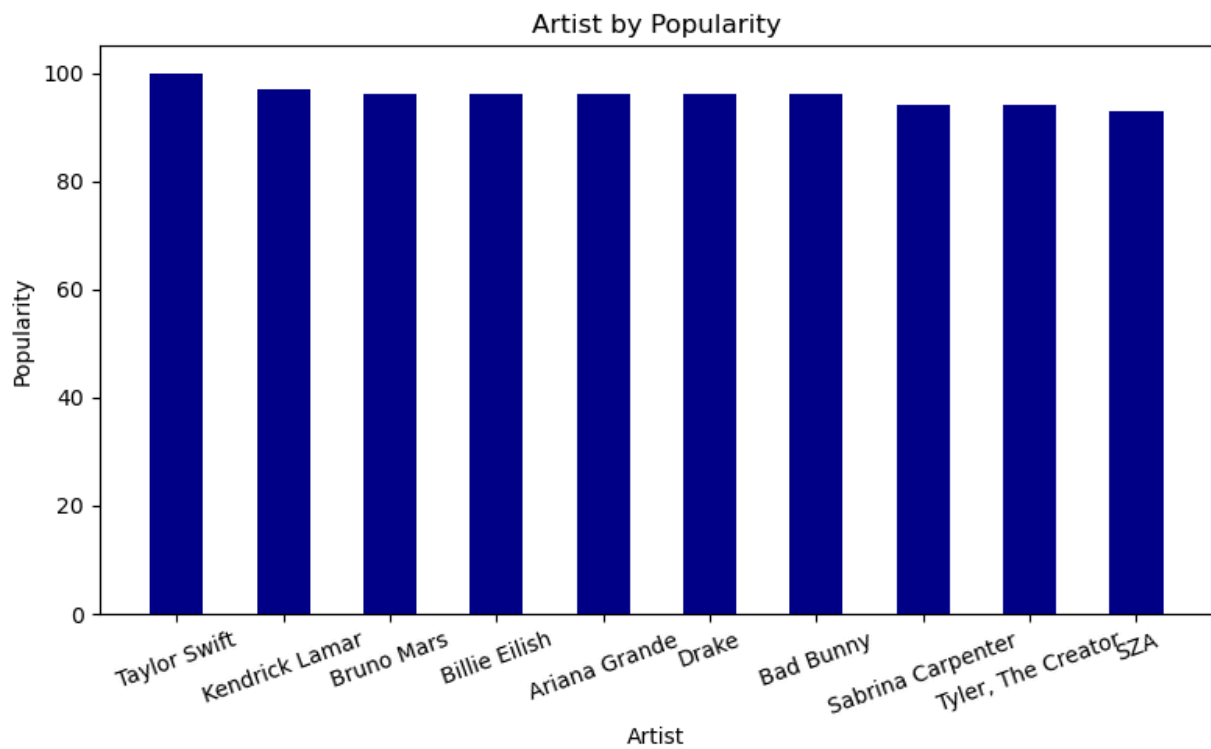
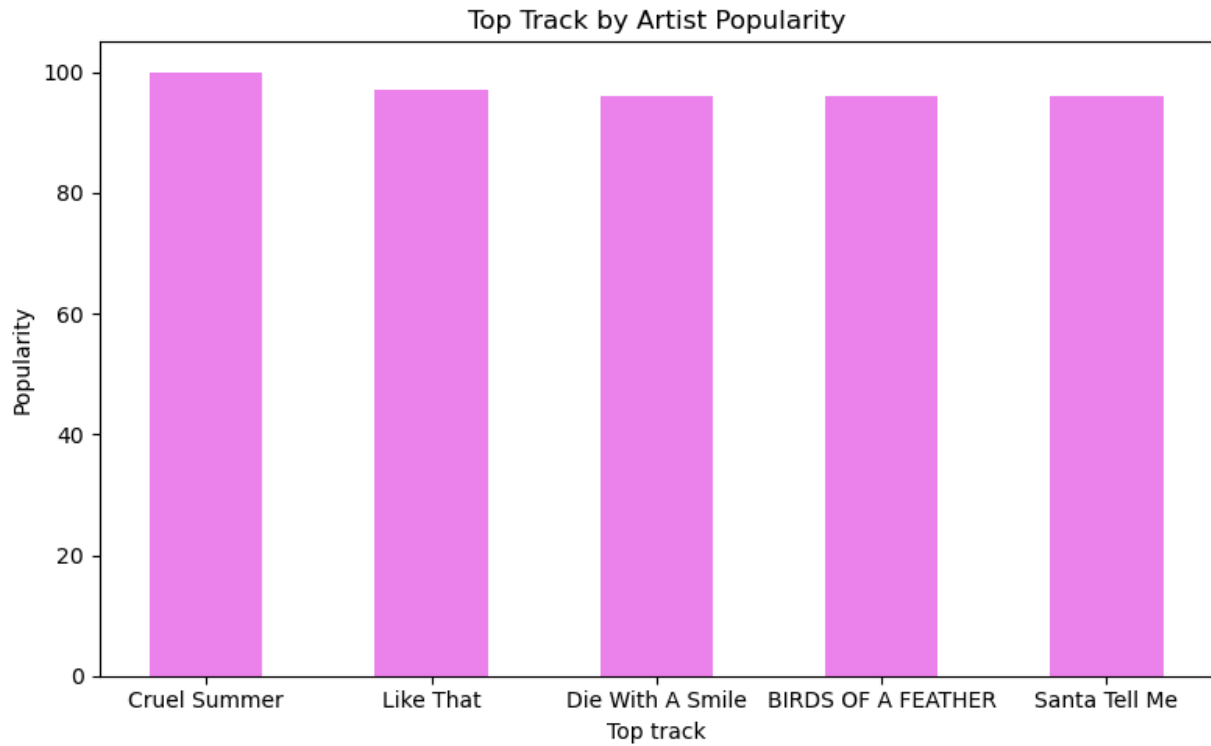
TMDB:



Google Books:



Spotify:



6. Instructions for running your code

1. Run `final_combined_database.py`:

- 1.1. This script initializes the SQLite database called combined.db and creates tables that contain all tables related to our three APIs: Google Books, Spotify, and TMDB. It processes data in batches (25 entries at a time) and writes them into the database.

- 1.1.1. combined.db will contain:

- 1.1.1.1. book table: Stores book_unique_id: A unique identifier for the book, title : The title of the book, published_date : The publication date of the book, average_rating : The average user rating of the book
 - 1.1.1.2. categories table: Stores genre or category information for books. Contains: id: A unique identifier integer for the category and name: The name of the category.
 - 1.1.1.3. Book_categories table: Maps books to their respective categories or genres (many-to-many relationship). Contains book_unique_id (from book table): The unique identifier of the book and category_id (from categories table): The unique identifier of the category.
 - 1.1.1.4. Spotify_artists table: Stores information about Spotify artists retrieved from the Spotify API. Contains id: The Spotify artist's unique identifier, name: The name of the artist, and popularity: The popularity score of the artist (0–100).
 - 1.1.1.5. Top_tracks table: Stores information about the top tracks for Spotify artists. Contains id (from Spotify_artists table): The Spotify artist's unique identifier and top_track: The name of the artist's top track.
 - 1.1.1.6. movies table: Stores movie information retrieved from the TMDB API. Contains id: A unique integer identifier for the movie in the database, title: The title of the movie, year: The release year of the movie, and imdb_rating: The IMDb rating of the movie.
 - 1.1.1.7. genres table: Stores genre information for movies. Contains id : A unique integer identifier for the genre and name: The name of the genre.
 - 1.1.1.8. Movie_genres table: Maps movies to their respective genres (many-to-many relationship). Contains movie_id

(from movie table): The unique integer identifier of the movie and genre_id (from genre table): The unique integer identifier of the genre.

- 1.1.1.9. progress Table: Tracks the last processed index for each data category to enable batch processing. Contains id : A unique integer identifier for the progress entry. category: The category name (books, artists, movies, or top_tracks), and last_index : The last processed index for the category.

- 1.2. Each run of this code will populate 25 or fewer entries into combined.db run final_combined_database.py a total of **6** times to populate all the data: 100 books, 104 movies, and 126 artists and top tracks into the combined database

- 2. Run **final_combined_calculations.py**. This script calculates averages and aggregates data for books, movies, and Spotify tracks. The results are saved in CSV files for further analysis and visualization. You only need to run this code 1 time.
 - 2.1. Establishes a connection to the combined.db SQLite database by creating a cursor and connection object for executing SQL queries.
 - 2.2. Computes the average IMDb rating for each movie genre by joining the movies, genres, and movie_genres tables, Computes the average rating of all books in the books table, Calculates the average rating for each book genre by joining the books, categories, and book_categories tables, and Calculates the artist compared to their top track and popularity. Popularity is a number from 1 - 100 based on virality, following, engagement, and more factors.
 - 2.3. Writes results into CSV files:
 - 2.3.1. genre_calculations.csv: Contains The average rating of movies by genre
 - 2.3.2. Book_calculations.csv: Contains the cumulative average rating of all books, and the average rating for each book genre
 - 2.3.3. Spotify_data.csv: Contains the Spotify artist ID, name, popularity, and their top tracks

- 2.4. CSV files will be created in the same directory. To access CSVs click on the CSV files which will automatically pop up in the VS Code directory.
3. Run **final_graphs.py**. This file reads the CSV files from the final_combined_calculations file and pulls data directly from combined.db to create 5 total bar graphs. You only need to run this code 1 time.
 - 3.1. 5 total bar graphs are created from running this code visualize:
 - 3.1.1. Average IMDb rating by Genre: average IMDb rating by genre for all genres in database
 - 3.1.2. Average Rating of Books: average ratings by book title for 15 books
 - 3.1.3. Average Rating by Genre: average ratings by genre for 7 genres (randomly selected)
 - 3.1.4. Top Track by Artist Popularity: the 5 most popular tracks by artist.
 - 3.1.5. Artist By Popularity: popularity of the top 10 artists.
 - 3.2. All 5 graphs will pop up one after another after your exit each one, you may have to minimize your tabs to see them depending on your device. Adjusting the sliders to your liking on the Subplot Configuration Tool (located on the left of the graph popup, on the left of the magnifying glass icon) will provide a better visual of the full graph. There are a large number of titles for "Average IMDb Rating By Genre", and "Average Rating of Books" graph; it is strongly recommended to adjust the "bottom" slider in the Subplot Configuration Tool to properly view the entire graph.

7. Documentation for each function you wrote. This includes describing the input and output for each function

final_combined_database.py:

1. `fetch_books(query, max_results=25, start_index=0)`

- Input:
 - query (string) - The search term for books.
 - max_results (integer, default=25) - The maximum number of results to fetch.
 - start_index (integer, default=0) - The starting index for fetching data.
- Output: JSON response containing book data.

- Description: This function uses the Google Books API to fetch book data based on a query. The response contains details like title, published date, average rating, authors, and categories.

2. `store_book_data_in_db(data, cursor)`

- Input:
 - data (JSON object) - The data fetched from the Google Books API.
 - cursor (SQLite cursor) - The database cursor for executing SQL queries.
- Output: The number of books successfully added to the database.
- Description: This function parses the book data from the API response and inserts it into the books, authors, categories, book_authors, and book_categories tables. It avoids duplicate entries and establishes relationships between books, authors, and categories.

3. `get_spotify_access_token(client_id, client_secret)`

- Input:
 - client_id (string) - The Spotify API client ID.
 - client_secret (string) - The Spotify API client secret.
- Output: A Spotify API access token (string).
- Description: This function retrieves an access token from the Spotify API using client credentials. It is required for further Spotify API calls.

4. `get_artist_info(client_id, client_secret, start_index, limit=25)`

- Input:
 - client_id (string) - The Spotify API client ID.
 - client_secret (string) - The Spotify API client secret.
 - start_index (integer) - The starting index for artist data retrieval.
 - limit (integer, default=25) - The maximum number of artists to retrieve.
- Output: A list of artist information dictionaries.
- Description: This function fetches details of artists (e.g., ID, name, popularity) from the Spotify API in batches. It reads artist IDs from a predefined CSV file.

5. `create_spotify_table(data, cursor)`

- Input:
 - data (list) - A list of artist dictionaries from the Spotify API.
 - cursor (SQLite cursor) - The database cursor for executing SQL queries.
- Output: None.
- Description: This function inserts Spotify artist data (ID, name, popularity) into the Spotify_artists table. Duplicate entries are skipped.

6. `fetch_genres()`

- Input: None.
- Output: A dictionary mapping genre IDs to genre names.
- Description: This function fetches all movie genres from the TMDb API and returns them as a dictionary. It is used to establish relationships between movies and genres.

7. `fetch_movie_data(movie_title)`

- Input:
 - `movie_title` (string) - The title of the movie to fetch.
- Output: A dictionary containing movie data or None if no data is found.
- Description: This function retrieves movie details (e.g., title, release year, IMDb rating) from the TMDb API using the movie title as a search parameter.

8. `insert_genres(cursor, genre_mapping)`

- Input:
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
 - `genre_mapping` (dictionary) - A dictionary of genre IDs and names.
- Output: None.
- Description: This function inserts movie genres into the genres table using the mapping from `fetch_genres()`. Duplicate entries are skipped.

9. `insert_movie_data(cursor, movie_data)`

- Input:
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
 - `movie_data` (dictionary) - Movie details retrieved from the TMDb API.
- Output: The movie ID of the inserted record or None if insertion fails.
- Description: This function inserts movie data (title, release year, IMDb rating) into the movies table.

10. `link_movie_genres(cursor, movie_id, genre_ids)`

- Input:
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
 - `movie_id` (integer) - The ID of the movie to link.
 - `genre_ids` (list) - A list of genre IDs to associate with the movie.
- Output: None.
- Description: This function establishes relationships between movies and genres by populating the `movie_genres` table.

11. `get_toptrack_info(client_id, client_secret, start_index)`

- Input:
 - `client_id` (string) - The Spotify API client ID.
 - `client_secret` (string) - The Spotify API client secret.
 - `start_index` (integer) - The starting index for top track retrieval.
- Output: A list of top track dictionaries for a batch of artists.
- Description: This function retrieves the top tracks for Spotify artists from the API.

12. `create_toptrack_table(data, cursor, limit=25)`

- Input:
 - `data` (list) - A list of top track data for Spotify artists.
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
 - `limit` (integer, default=25) - The maximum number of tracks to insert.
- Output: None.
- Description: This function inserts top track data (track name and artist ID) into the `Top_tracks` table. Duplicate entries are skipped.

13. `initialize_progress(cursor)`

- Input:
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
- Output: None.
- Description: This function initializes the progress table to track the last processed index for books, artists, movies, and top tracks.

14. `update_progress(cursor, category, index)`

- Input:
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
 - `category` (string) - The category name (e.g., books, artists).
 - `index` (integer) - The last processed index.
- Output: None.
- Description: Updates the progress for a specific category in the progress table.

15. `get_last_index(cursor, category)`

- Input:
 - `cursor` (SQLite cursor) - The database cursor for executing SQL queries.
 - `category` (string) - The category name.
- Output: The last processed index for the specified category.

- Description: Retrieves the last processed index from the progress table for the specified category.

16. `main()`

- Input: None.
- Output: combined.db database containing these tables:
 - books
 - Purpose: Stores information about books, including their titles, publication dates, and average ratings.
 - Authors (not used in calculations/visualizations)
 - Purpose: Contains a list of authors' names for books.
 - categories
 - Purpose: Stores the genres or categories associated with books.
 - Book_authors (not used in calculations/visualizations)
 - Purpose: Maps books to their respective authors (many-to-many relationship).
 - book_categories
 - Purpose: Maps books to their respective categories or genres (many-to-many relationship).
 - Spotify_artists
 - Purpose: Stores information about Spotify artists, including their IDs, names, and popularity scores.
 - Top_tracks
 - Purpose: Stores information about the top tracks of Spotify artists, including track names and associated artist IDs.
 - movies
 - Purpose: Contains movie details, including titles, release years, and IMDb ratings.
 - genres
 - Purpose: Stores genres associated with movies.
 - movie_genres
 - Purpose: Maps movies to their respective genres (many-to-many relationship).
 - progress
 - Purpose: Tracks the last processed index for each data category (books, artists, movies, top tracks), enabling batch processing.
- Description: This is the main function that:
 - Initializes the database and tables.
 - Processes batches of books, Spotify artists, movies, and top tracks (up to 25 entries per batch).
 - Updates progress in the progress table after each run.

final_combined_calculations.py

- This is the

1. set_up_database(DB_NAME)

- Input:
 - DB_NAME (str): The name of the SQLite database file to connect to.
- Outputs:
 - Tuple: Contains the SQLite cursor (sqlite cursor) and connection object (sqlite connection).
- Description: This function sets up a connection to the specified SQLite database and returns both the connection and cursor for further database operations.

2. list_tables()

- Inputs:
 - None
- Outputs:
 - tables (list of tuples): A list of table names in the connected SQLite database.
- Description: This function connects to the SQLite database, queries the sqlite_master table to retrieve all table names, and returns them. It closes the connection after execution.

3. join_two_tables(cur)

- Inputs:
 - cur (sqlite3.Cursor): The cursor object to execute SQL queries.
- Outputs:
 - data (list of tuples): A list containing combined rows from the Spotify_artists and Top_tracks tables. Each tuple contains:
 - id (int): Artist ID.
 - name (str): Artist name.
 - popularity (int): Popularity score.
 - top_track (str): The top track associated with the artist.
- Description: Does SQL JOIN between the Spotify_artists and Top_tracks tables based on their id fields. The results are ordered by the artist's popularity in descending order.

4. calculate_average_imdb_rating()

- Inputs:
 - None
- Outputs:
 - `average_rating` (float): The average IMDb rating of all movies.
- Description: This function calculates the overall average IMDb rating by querying the movies table. It converts the rating to a REAL type to ensure numeric calculation.

5. `calculate_average_rating_by_genre()`

- Inputs:
 - None
- Outputs:
 - `average_rating_by_genre` (list of tuples): A list where each tuple contains:
 - `genre` (str): Genre name.
 - `average_rating` (float): The average IMDb rating for the genre.
- Description: This function calculates the average IMDb rating grouped by movie genres by joining the movies, movie_genres, and genres tables. Results are sorted by average rating in descending order.

6. `calculate_average_ratings()`

- Inputs:
 - None
- Outputs:
 - `average_rating` (float): The overall average rating of all books in the books table.
- Description: This function calculates the average book rating from the books table, excluding rows where the rating is NULL.

7. `average_rating_by_genre()`

- Inputs:
 - None
- Outputs:
 - `genres` (list of str): A list of genre/category names.
 - `avg_ratings` (list of float): A list of average ratings corresponding to each genre.

- Description: This function calculates the average book rating for each genre by performing joins between the categories, book_categories, and books tables. Results are grouped by genre names.

8. write_to_bookcsv(genres, avg_ratings, average_rating)

- Inputs:
 - genres (list of str): List of genre/category names.
 - avg_ratings (list of float): List of average ratings for each genre.
 - average_rating (float): Overall average rating of all books.
- Outputs:
 - None (creates a CSV file named book_calculations.csv)
- Description: This function writes the overall book average rating and the average ratings per genre to a CSV file.

9. write_to_csv(data, filename, headers)

- Inputs:
 - data (list of tuples): A list of rows to write into the CSV file.
 - filename (str): The name of the CSV file to create.
 - headers (list of str): Column headers for the CSV file.
- Outputs:
 - None (creates a CSV file with the given name and data)
- Description: This generic function writes data into a CSV file, including headers and rows.

10. main_calculations()

- Inputs:
 - None
- Outputs:
 - None (calls other functions and writes a CSV file)
- Description: This function performs the book-related calculations:
 - Calculates the overall book average rating
 - Calculates average ratings by genre
 - Writes the results to a CSV file using write_to_bookcsv

11. main()

- Inputs:
 - None.

- Outputs:
 - None (calls other functions, prints results, and creates CSV files)
- Description: This is the main function for orchestrating all operations:
 - Sets up combined database
 - Calculates the average IMDb rating and writes it to a CSV file
 - Calculates average IMDb ratings by genre and writes the results to a CSV file
 - Performs a join between Spotify_artists and Top_tracks tables, saving the results in a CSV file

Final_graphs.py

1. `read_csv_data(filename, usecols=None)`

Input:

- filename (string): The name of the CSV file to be read.
- usecols (list, default=None): A list of column names to load from the CSV.

Output:

- Returns a Pandas DataFrame containing the requested data from the CSV file.

Description:

This function reads data from a specified CSV file, optionally filtering the columns to include only those specified in usecols. It returns the data as a DataFrame.

2. `get_data_from_db(query)`

Input:

- query (string): The SQL query to execute on the SQLite database.

Output:

- Returns a list of tuples, where each tuple represents a row fetched by the query.

Description:

This function connects to the SQLite database (combined.db), executes the given SQL query, fetches the results, and closes the connection.

3. `visualize_book_data()`

Input:

- None.

Output:

- Displays a horizontal bar chart of book titles against their average ratings.

Description:

This function retrieves book titles and their average ratings from the database and visualizes the data using a horizontal bar chart. It shows how books are rated on average.

4. `average_rating_by_genre_graph()`

Input:

- None.

Output:

- Displays a horizontal bar chart showing genres and their average book ratings.

Description:

This function calculates the average rating of books by genre, using data from multiple tables in the database. The results are visualized in a bar chart.

5. `genre_imdb()`

Input:

- None.

Output:

- Displays a bar chart of genres against their average IMDb ratings.

Description:

This function reads IMDb genre and rating data from a CSV file (`genre_calculations.csv`) and creates a bar chart to show the average rating for each genre.

6. `plot_by_top_song(data)`

Input:

- data (list of lists): Spotify data where each inner list represents a row with song popularity and metadata.

Output:

- Displays a bar chart of top tracks against their popularity scores.

Description:

This function visualizes the popularity of the top five Spotify tracks from the provided data using a bar chart.

7. `plot_by_top_artist(data)`

Input:

- data (list of lists): Spotify data where each inner list represents a row with artist popularity and metadata.

Output:

- Displays a bar chart of artists against their popularity scores.

Description:

This function visualizes the popularity of the top ten artists from the provided data using a bar chart.

8. `main_graphs()`

Input:

- None.

Output:

- Calls the other functions in the script to generate and display multiple visualizations.

Description:

This function orchestrates the generation of various graphs, including book ratings, IMDb genre ratings, and Spotify data visualizations. It serves as the entry point for running the script.

8. Clearly document all resources used.

Spotify:

Date	Issue Description	Resource Location	Result (did it solve the issue?)
12/9/24	Needed to understand how to use APIs in Python	https://www.dataquest.io/blog/api-in-python/	Solved
12/9/24	Understanding Oauth	https://stackoverflow.com/questions/63720756/importerror-no-module-named-requests-oauthlib-error	Solved
12/9/24	Requesting API key, how to use Spotify's API	https://developer.spotify.com/documentation/web-api/concepts/api-calls	Solved and referenced multiple times
12/9/24	How to get artist IDs to input into API?	https://www.reddit.com/r/spotify/comments/gx2a1h/spotify_artist_id_system_is_a_fucking_mess/	Solved
12/9/24	How to find unique artists to get IDs?	https://chartmasters.org/most-streamed-artists-ever-on-spotify/	Solved
12/9/24	How to clean up artist URIs to get IDs to input into API?	https://www.ablebits.com/office-addins-blog/remove-characters-from-string-excel/	Solved
12/9/24	What does the API return/how do I format it?	https://stackoverflow.com/questions/6260457/using-headers-with-the-python-requests-librarys-get-method	Solved
12/9/24	Accessing/iterating through CSV	https://www.geeksforgeeks.org/reading-cs	Solved

		v-files-in-python/	
12/10/24	Understanding JSON result	https://jsonformatter.org/json-pretty-print , chatGPT	Solved
12/10/24	Creating database	https://www.w3schools.com/sql/ https://www.w3schools.com/sql/sql_create_db.asp	Solved
12/10/24	What is the UnicodeDecodeError?	https://wiki.python.org/moin/UnicodeDecodeError	One of the top tracks used an emoji in the title, had to skip it
12/10/24	SQL JOIN vs INNER JOIN	https://www.w3schools.com/sql/sql_join.asp , chatGPT	Solved

TMDB:

Date	Issue Description	Resource Location	Result (did it solve the issue?)
12/9/24	Understanding how to get access/ use TMDB API	https://developer.themoviedb.org/reference/intro/getting-started	solved
12/9/2024	Creating database	https://www.w3schools.com/sql/ , https://www.w3schools.com/sql/sql_create_db.asp	solved
12/9/2024	Limiting calls to 25 per run	https://stackoverflow.com/questions/40748687/python-api-rate-limiting-how-to-limit-api-calls-globally	solved
12/9/2024	How to join tables and remove duplicate strings - prompt: "does my code contain duplicate strings. What strategies can I take to get rid of duplicate strings? Don't give	UMGPT	solved

	me the corrected code”		
12/10/2024	Debugging the entire project for almost every function. Alot of missing quotes, indentation errors, had to learn how to write sql. Helped debug the sql titles and add color to graphs. Fixed a lot of errors, identified missing quotes,indentation and how to add color to graphs	UMGPT	solved

Google Books:

Date	Issue Description	Resource Location	Result (did it solve the issue?)
12/2/24	Understanding how to use Google Books API	https://developers.google.com/books	Solved
12/2/24	Handling inconsistent genre tags	https://docs.python.org/3/library/text.html	Solved
12/9/24	Creating database and storing data	https://www.w3schools.com/sql/ , https://www.w3schools.com/sql/sql_create_db.asp	Solved
12/10/24	Handling missing or null values in API responses	https://stackoverflow.com/questions/642154/how-to-find-out-if-a-key-exists-in-a-dictionary	Solved
12/12/24	Ensuring consistent data formatting	https://www.geeksforgeeks.org/reading-csv-files-in-python/	Solved
12/12/24	SQL JOIN vs INNER JOIN	https://www.w3schools.com/sql/sql_join.asp	Solved

		p	
--	--	-------------------	--