1. Original Goals

Originally, The Collectors planned to work with three distinct API's: OMDb API, Spotify API and Open Trivia Database. The main goals for these APIs were to collect aggregate data that we could sufficiently use towards calculating further information on the project. For the OMDb API, conducted by Ethan Rayman, we expected to accumulate data on movie titles, years, genres, directors, actors, IMDb ratings, box office earnings and runtimes. For the Spotify API, conducted by Daniel Gampel, we expected to collect info on song track names, artist names, album names, release dates, popularity, durations and audio features. For the Open Trivia Database, conducted by Ethan Shemtov, we planned to research trivia questions, answers, incorrect answers, categories and difficulties.

2. Achieved Goals

After starting the project, we adjusted our goals based on accessibility and usability of the APIs. The only database we retained was the OMDb database, however Daniel and Ethan decided to replace their databases with the REST Countries API and Open Library API, respectively. The countries database was chosen because of initial issues using and retrieving the data from the Spotify database, and also because it provides relevant global information. Ethan Shemtov switched from the Open Trivia Database to the Open Library API because its free version had limits and didn't provide enough data. The Open Library API was chosen because it provided more detailed and diverse information, making it a better fit for the project. For OMDb, we collected all parameters expected, including movie title, year, genre, director, actors, IMDb rating, box office total and runtime. For REST Countries API, we created two tables, countries and languages. Within the countries table we recorded the country name, region, capital, continent, landlocked boolean and their currency. Within the languages table we only focused on the country ID and the language name. For the Open Library API, we built a books table and an authors table. The books table included parameters such as title, author id, first publishing year, isbn code, the language and subject. The authors table only included ids and author names to be more compact.

3. Problems Faced

OMDb Database:

For this project, there were a few issues in retrieving the full data that I wanted to capture and managing to avoid duplicates in data while adding 25 entries each run to the existing table entries. In my first attempts, I was unaware that I needed to retrieve imdb ids from each movie beforehand to get the full scope of data that I intended to accumulate, therefore it took some trial and error before researching the documentation and understanding how to retrieve it using the ids as a key. Additionally, doing research on how to accurately count existing entries in a database, limit the entries added and increase the total count of entries in data.db on each run was a significant component in reaching this final product.

Books Database:

During this project, I faced several challenges, including handling inconsistent data from the Open Library API, such as missing fields for certain books or authors, which required additional error handling and validation. Another issue was ensuring duplicate records were avoided in the database, which involved writing careful checks and queries to compare newly created data with existing entries. Debugging pagination and offset logic was also challenging, as I needed to ensure that data fetched in

batches did not overlap or miss records. Additionally, creating meaningful visualizations required experimenting with different graph types and refining the data to achieve clarity and relevance.

Countries Database:
During development of the countries database, several challenges arose. The script initially inserted duplicate data into the database each time it ran, which was resolved by adding a pre-insertion check to skip duplicates. Missing directories, such as txt files and visualizations, caused FileNotFoundError when saving files or visualizations, so code was added to create these directories automatically. Additionally, visualizations failed to display initially due to backend configuration issues with matplotlib, which was fixed by ensuring the appropriate backend was used and calling plt.show(). Finally, processing nested JSON data from the REST Countries API, especially for currencies and languages, required robust logic to handle dictionaries and lists effectively.

4. Data Calculations

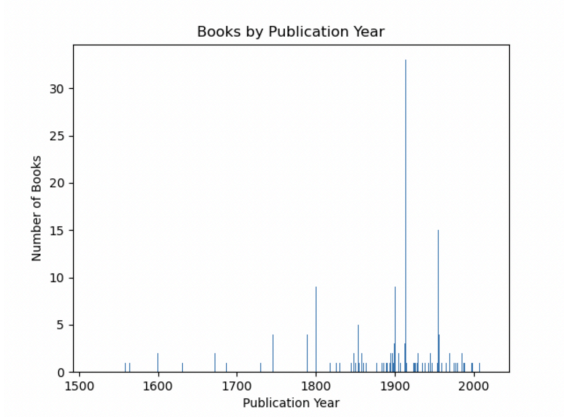
```
Genre, Average IMDb Rating, Frequency
Biography, 8.10, 1
Documentary, 7.37, 7
Western, 7.30, 1
Mystery, 7.27, 3
Thriller, 6.80, 1
Animation, 6.76, 48
Crime, 6.73, 7
Action, 6.70, 33
Drama, 6.62, 10
Sport, 6.40, 1
Adventure, 6.29, 52
Fantasy, 5.91, 8
Comedy, 5.82, 61
Sci-Fi, 5.63, 6
Horror, 5.38, 4
Family, 4.87, 9
Romance, 3.90, 2

```

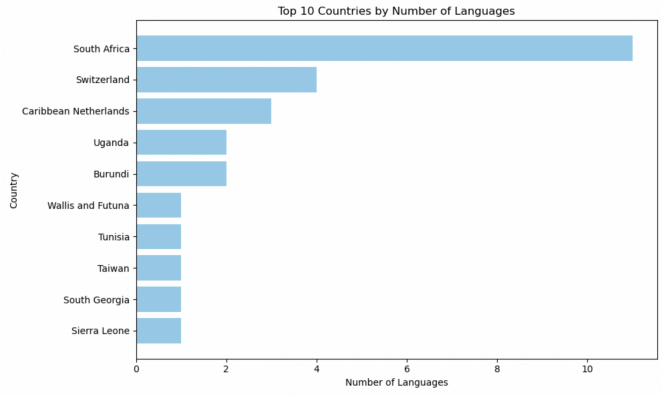
```
Region, Country Count
Africa, 9
Europe, 5
Americas, 5
Asia, 3
Oceania, 2
Antarctic, 1

```

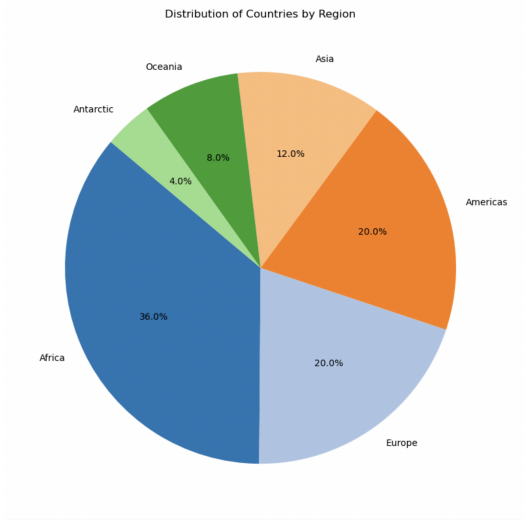
```
Year, Book Count
1518, 1
1559, 1
1564, 1
1600, 2
1631, 1
1672, 2
1687, 1
1730, 1
1746, 4
1789, 4
1800, 9
1818, 1
1826, 1
1830, 1
1845, 1
1848, 2
1850, 1
1854, 5
1855, 1
1858, 2
1860, 1
1864, 1
1877, 1
1881, 1
1883, 1
1884, 1
1886, 1
1889, 1
1890, 1
1893, 1
```

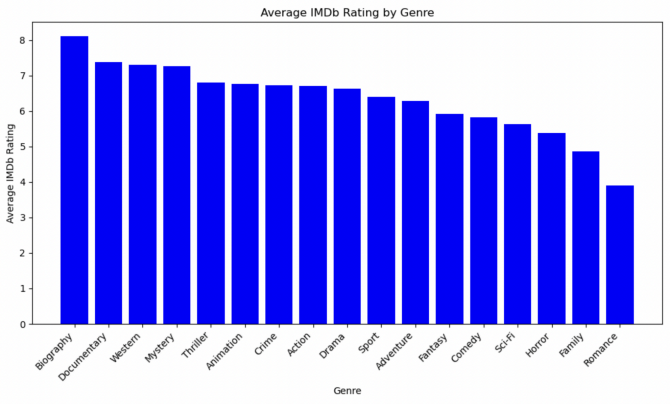
```
Country, Language Count
South Africa, 11
Switzerland, 4
Caribbean Netherlands, 3
Uganda, 2
Burundi, 2
Wallis and Futuna, 1
Tunisia, 1
Taiwan, 1
South Georgia, 1
Sierra Leone, 1
Saint Kitts and Nevis, 1
Pitcairn Islands, 1
Mexico, 1
Libya, 1
Laos, 1
Ivory Coast, 1
Italy, 1
Indonesia, 1
Hungary, 1
Grenada, 1
France, 1
Cape Verde, 1
Benin, 1
Barbados, 1
Andorra, 1

```

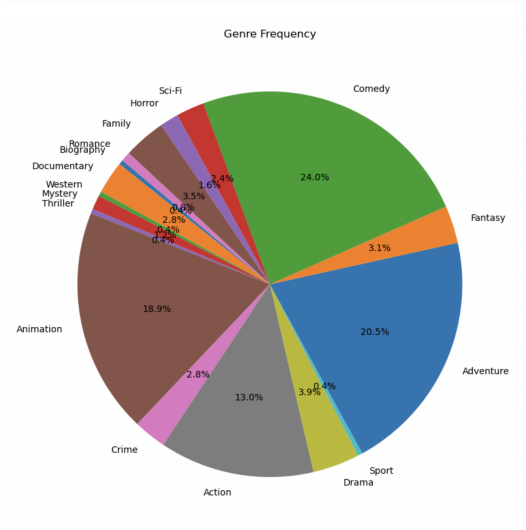
5. Data Visualizations











6. Instructions to Run

To run our project, we ensured Python 3.x was installed along with the required libraries like SQLite3, requests, and JSON. Each team member worked on their own script for a specific database: a countries database, a books database using the Open Library API, and a movies database using the OMDb API. Each script was saved as a separate file (countries_project.py, books_project.py, and movies_project.py) and executed individually. The scripts are designed to gather data in batches of 25, requiring multiple runs to gather at least 100 records for each dataset without modifying the source code. The data is stored in separate SQLite databases (countries_data.db, books_data.db, and movies_data.db), and JSON files are generated (countries_data.json, books_data.json, and movies_data.json) in the same directory as the scripts. Running the scripts required an active internet connection for the API requests, and we used tools like DB Browser for SQLite to verify the stored data. This approach ensured that we each contributed equally to the project while maintaining separate databases. Please make sure to run the OMDb, REST Countries and Open Library API files at least four times each to accumulate the necessary data into data.db, and feel free to use the viewdatabase.py file to review what data exists within data.db from each table.

7. Documentation of Each Function

OMDb Database:
The code for the OMDb API is split up into each of its functions and for easier navigation within the code files. The first of these functions is the makedb() function, which initializes the sqlite database and a table named ‘movies’ if it does not already exist. There is no input for this function, and the only output is the creation of the table within the sqlite database, returning None. The next is the getAPImovies(page) function, which serves the purpose of pulling lists of movies from the OMDb API based on the page provided for the API. The input is an integer representing the page number, and the output is a list of dictionaries derived from the JSON data output. After getAPImovies, there is getAPImoviedetails(imdb_id) that utilizes imdb ids from the initial getAPImovies results to find more details on each movie, inputting a string as imdb_id and outputting a dictionary containing more intricate details stored on that movie. Next, savetodb(movies) specific movie data into the existing sqlite database after retrieving more detailed information on each of the inputted movies. The input for this is another list of dictionaries including that detailed movie information, and the output is None, as all information is saved into the local db. For genrestats(), we calculate the average imdb rating per genre and frequency of each genre from the existing movies stored in that data.db database. There is no input for this, and the output includes a comprehensive list of tuples that include the genre, average rating and count of that genre. Then, the writetotxt(data) function wrote the calculated genre statistics to a text file as required by the assignment. The input of this was the comprehensive list of tuples output from the genrestats() function, which wrote information on the genre, rating and count for each genre into the text file. The output was None, however the file was created or overwritten with the new data. One of the last was the makebarchart()/makepiechart() functions, which each utilized the data stored from genrestats() to build visualizations. The bar chart creates a png file and displays the result with matplotlib, showing the average ratings of each genre in bar chart format. The pie chart also creates a png file and displays with matplotlib, however this one shows the distribution of frequencies across each genre, showing which are more prevalent within the existing data set. Lastly, main() puts together all of the functions, making the

database, fetching and storing data from the OMDb API, and generating statistics that are written to a txt file and then applied towards visualizations.

Countries Database:
The code consists of multiple functions, each serving a distinct purpose within the project. The setup_database() function initializes the SQLite database by creating two tables, countries and languages, if they do not already exist. It takes no input and ensures the database structure is ready for data insertion. The fetch_data() function retrieves data from the REST Countries API and returns a list of JSON objects. It handles potential HTTP errors gracefully. The insert_data() function processes the API data, inserting country and language information into the database while avoiding duplicate entries by performing a pre-insertion check. The calculate_languages_per_country() function performs a database join between the countries and languages tables to calculate the number of languages spoken in each country. It writes the results to a file named languages_per_country.txt and returns the data as a list of tuples. Similarly, the calculate_countries_per_region() function groups countries by region, counts how many countries are in each region, and writes the results to a file named countries_per_region.txt. The visualize_languages_per_country() function takes the results of the language calculation and generates a bar chart of the top 10 countries by language count, saving the visualization as languages_per_country.png. Finally, the visualize_countries_per_region() function creates a pie chart representing the distribution of countries across different regions, saving it as countries_per_region.png. Together, these functions ensure a comprehensive flow of data collection, processing, and visualization.

Books Database:
Each function in the code is documented to clearly outline its purpose, inputs, and outputs. The setup_database() function initializes the SQLite database by creating books and authors tables if they do not already exist; it takes no inputs and returns nothing. The fetch_data() function retrieves data from the Open Library API based on a query and an offset for listing, and it provides a list of book entries as JSON objects. The insert_data() function processes the fetched data, inserts authors into the authors table, and associates books with the correct author in the books table; it takes a list of book data as input and returns nothing. The export_to_json() function extracts all book and author details from the database, combines them, and writes the data to a JSON file. This function has no input or output aside from the JSON file creation. Finally, the create_visualizations() function generates a bar chart visualizing the number of books by publication year using data retrieved from the database, taking no input and returning nothing but displaying the visualization.