SI 206 Final Report
Adaptation Analyzers
Sarah Kichula, Andrew Wang, Emma Johnson

Github repository: https://github.com/skichula/SI206-final-project

Planned Goals

Our planned goals were to work with the APIs from the OMBd, Google Books, and Open Library in order to collect movie and book titles and their corresponding ratings or reviews. We wanted to analyze the difference in reviews between books and their respective movie adaptations. We planned to calculate the average ratings for several books and their corresponding movies in order to determine which medium (book or movie) receives higher ratings on average. Additionally, using book data from both Google Books and Open Library, we planned to compare ratings for the same books from the different sources to see which source, if any, averaged higher ratings.

Achieved Goals

We used the same APIs that we had planned to in order to collect movie and book titles and their corresponding ratings or reviews; additionally, we collected the primary genre categorization for each movie title found to categorize them. Then, we were able to calculate and compare averages of the three movie ratings found from OMBd–IMBd, Rotten Tomatoes, and Metacritic for each respective genre. We were also able to find the book ratings from both Google Books and Open Library, comparing the differences in book ratings from the different sources. Furthermore, we were able to calculate the average of the two sources for one book rating to compare to its corresponding movie rating to see which rated higher.

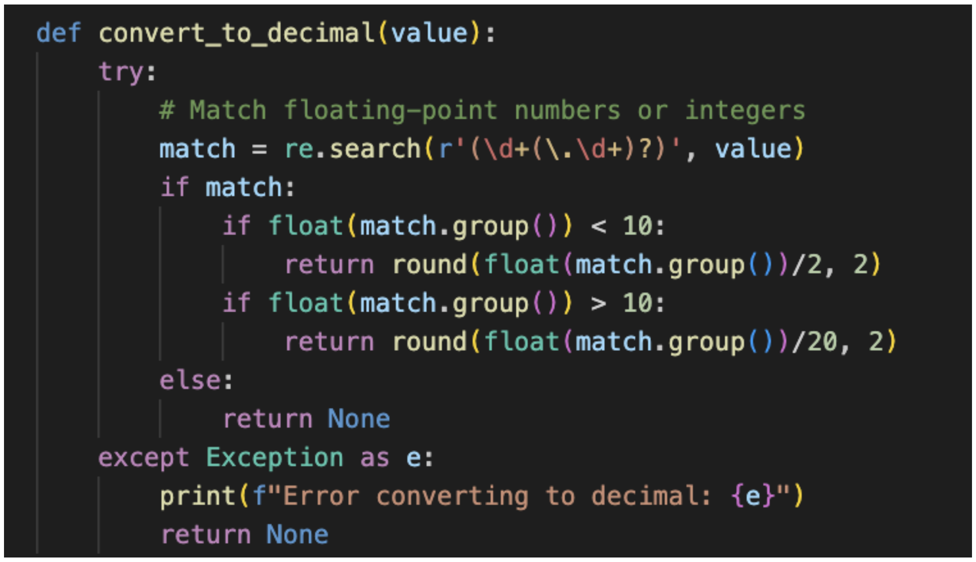
Problems Faced

One of the problems we faced when comparing and calculating the movie reviews from IMDb, Rotten Tomatoes, and Metacritic were that they all came in different formats–either as percentages, ratios with a denominator of 10, or ratios with a denominator of 100. In order to address this issue, we created the convert_to_decimal function which uses regular expressions to convert each of the different formats into floating point values. Another issue that arose was the titles of some movies that we accessed from IMDb did not align with the titles from OMDb; for example, “Dune” and “Dune: Part One” did not match. In this case, we had to add an exception that would skip the insertion into the database in order to avoid any errors. We encountered another challenge when certain movie titles didn’t perfectly align with their corresponding book titles. Consequently, when attempting to assign the same title ID for a book with a movie adaptation, only a few titles were matching exactly. In order to address this issue, we used the fuzz.partial_ratio() function from the Fuzzy Wuzzy library. This allowed us to compare the similarity between movie titles and book titles. We then selected the titles with the highest similarity ratio, facilitating the matching of title IDs for the same story without the titles having to be exactly the same.

Data Calculations

Movie Ratings:

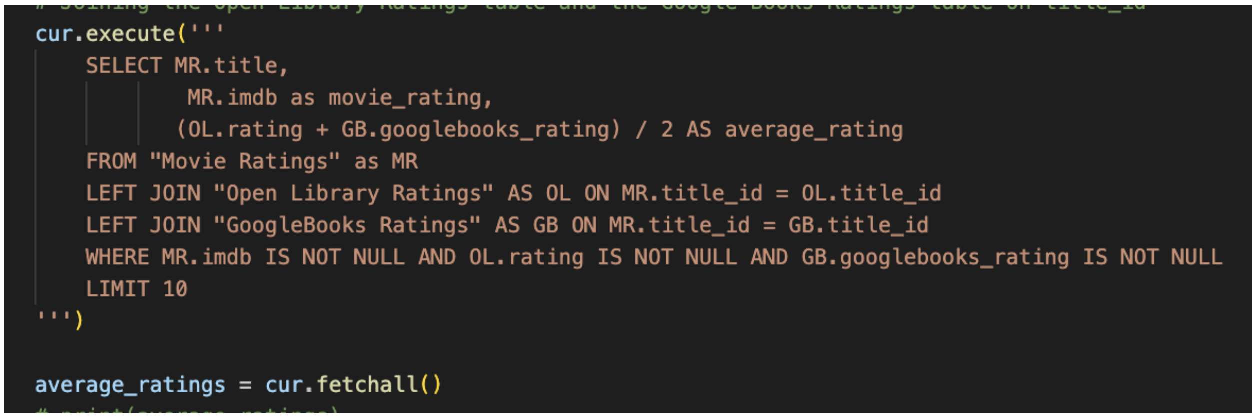
Took the movie ratings, which originally came on a scale of 1-10 or 1-100, and converted them to be on a scale of 1-5.



```python
def convert_to_decimal(value):
    try:
        # Match floating-point numbers or integers
        match = re.search(r'(\d+(\.\d+)?)', value)
        if match:
            if float(match.group()) < 10:
                return round(float(match.group())/2, 2)
            if float(match.group()) > 10:
                return round(float(match.group())/20, 2)
        else:
            return None
    except Exception as e:
        print(f"Error converting to decimal: {e}")
        return None
```

GoogleBooks Ratings and Open Library Ratings:

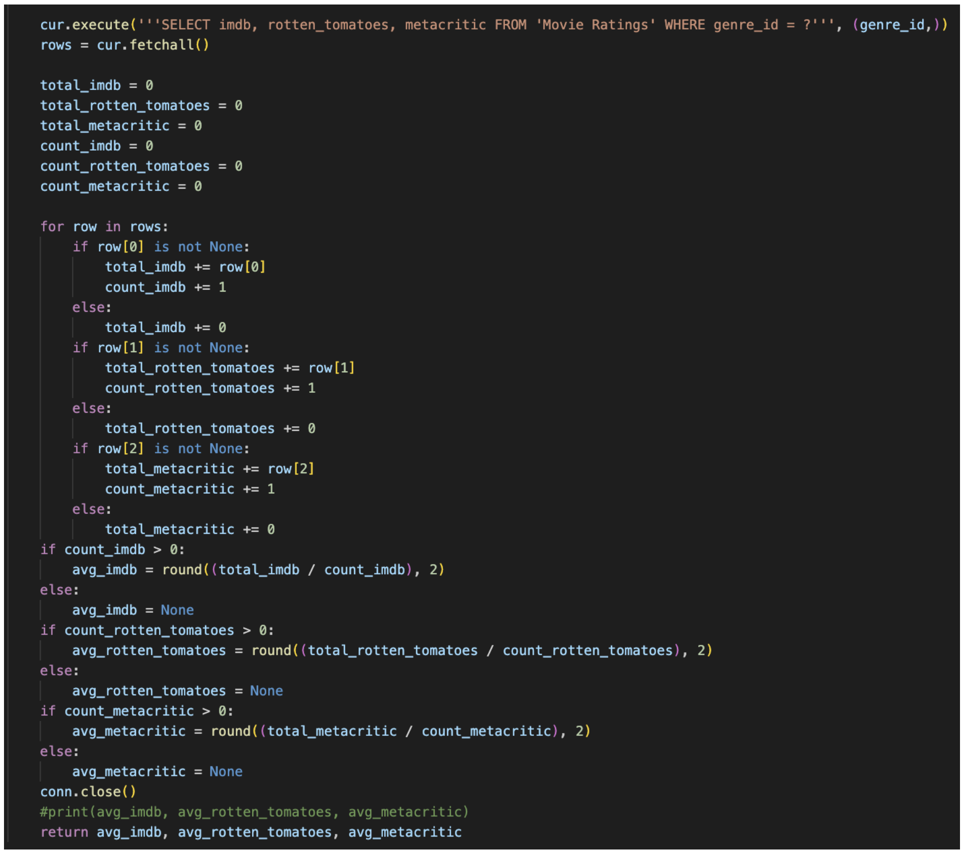
Calculated the average rating for the same book title by adding together the rating from the “GoogleBooks Ratings” table and the “Open Library Ratings” table and dividing it by two.



```python
cur.execute('''
    SELECT MR.title,
            MR.imdb as movie_rating,
           (OL.rating + GB.googlebooks_rating) / 2 AS average_rating
    FROM "Movie Ratings" as MR
    LEFT JOIN "Open Library Ratings" AS OL ON MR.title_id = OL.title_id
    LEFT JOIN "GoogleBooks Ratings" AS GB ON MR.title_id = GB.title_id
    WHERE MR.imdb IS NOT NULL AND OL.rating IS NOT NULL AND GB.googlebooks_rating IS NOT NULL
    LIMIT 10
''')

average_ratings = cur.fetchall()
```

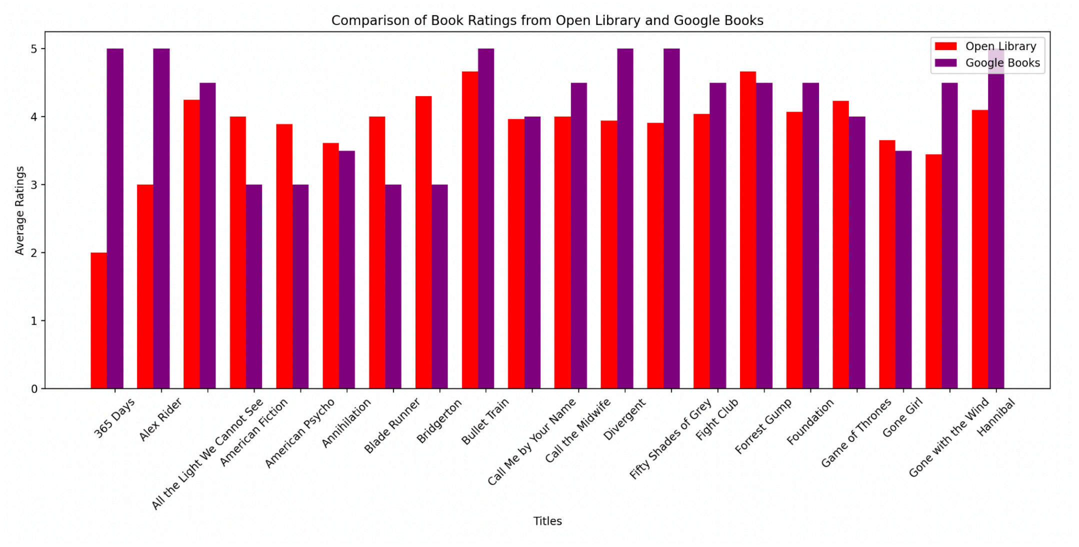
Genres:

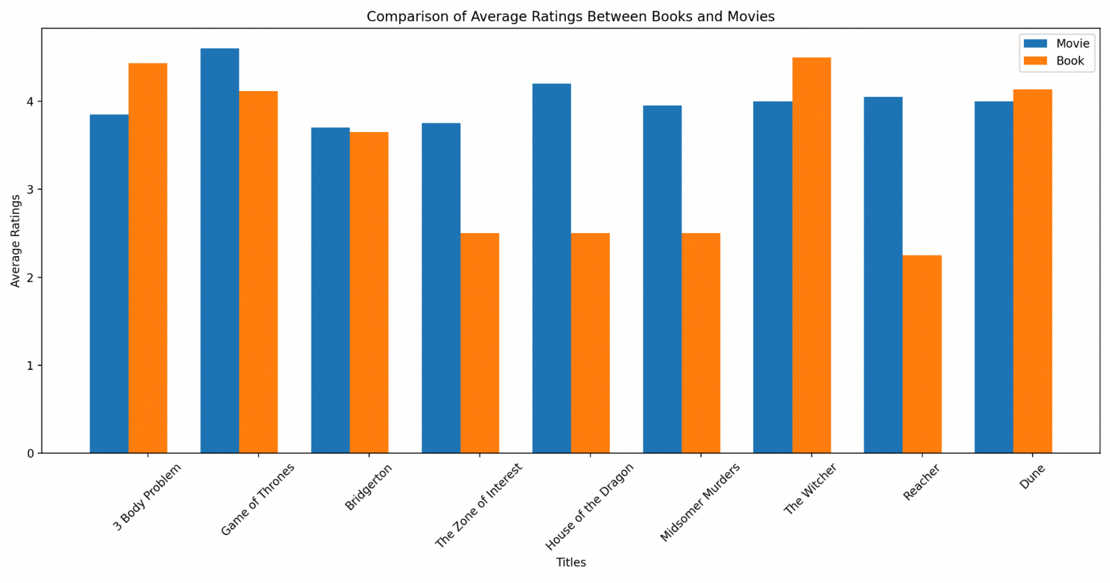
For each genre, we calculated the sum of all ratings and the number of ratings found for each of the following categories: IMDb, Rotten Tomatoes, and Metacritic. We then divided the sum of total ratings and the count of each category to calculate the average for each type of rating.

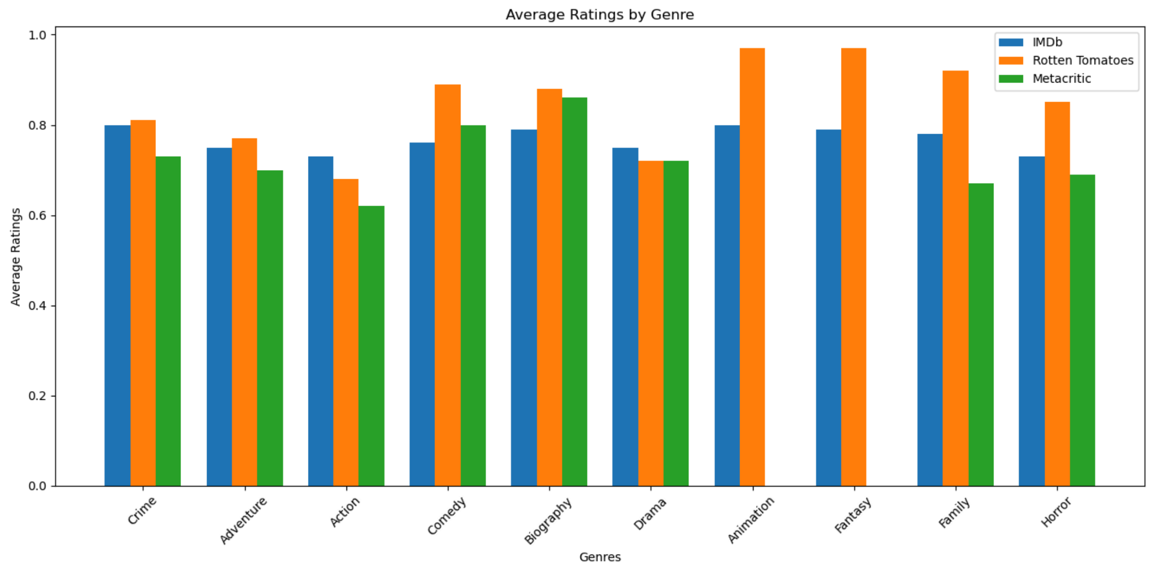


```
cur.execute('''SELECT imdb, rotten_tomatoes, metacritic FROM 'Movie Ratings' WHERE genre_id = ?''', (genre_id,))
rows = cur.fetchall()

total_imdb = 0
total_rotten_tomatoes = 0
total_metacritic = 0
count_imdb = 0
count_rotten_tomatoes = 0
count_metacritic = 0

for row in rows:
    if row[0] is not None:
        total_imdb += row[0]
        count_imdb += 1
    else:
        total_imdb += 0
    if row[1] is not None:
        total_rotten_tomatoes += row[1]
        count_rotten_tomatoes += 1
    else:
        total_rotten_tomatoes += 0
    if row[2] is not None:
        total_metacritic += row[2]
        count_metacritic += 1
    else:
        total_metacritic += 0
if count_imdb > 0:
    avg_imdb = round((total_imdb / count_imdb), 2)
else:
    avg_imdb = None
if count_rotten_tomatoes > 0:
    avg_rotten_tomatoes = round((total_rotten_tomatoes / count_rotten_tomatoes), 2)
else:
    avg_rotten_tomatoes = None
if count_metacritic > 0:
    avg_metacritic = round((total_metacritic / count_metacritic), 2)
else:
    avg_metacritic = None
conn.close()
#print(avg_imdb, avg_rotten_tomatoes, avg_metacritic)
return avg_imdb, avg_rotten_tomatoes, avg_metacritic
```

Data Visualizations


Comparison of Book Ratings from Open Library and Google Books
Open Library
Google Books
Average Ratings
Titles
365 Days
Alex Rider
All the Light We Cannot See
American Fiction
American Psycho
Annihilation
Blade Runner
Bridgerton
Bullet Train
Call Me by Your Name
Call the Midwife
Divergent
Fifty Shades of Grey
Fight Club
Forrest Gump
Foundation
Game of Thrones
Gone Girl
Gone with the Wind
Hannibal


Comparison of Average Ratings Between Books and Movies
Movie
Book
Average Ratings
Titles
3 Body Problem
Game of Thrones
Bridgerton
The Zone of Interest
House of the Dragon
Midsomer Murders
The Witcher
Reacher
Dune



Instructions to Run Code

First, install the Fuzzy Wuzzy library by using pip install fuzzywuzzy and pip install python-Levenshtein. After cloning the repository and locating the SI206-final-project folder, run the omdb.py first. This file initializes the database, so it must be run first. Next, run the open_library.py file. Afterwards, run the googbooks.py. After running the three of these once each, repeat the process until each file has been run five times. At the end, run the calculations.py file to see the plots.

Function Documentation

omdb.py

read_api_key(file): This function opens a text file with the OMDb API key and reads it.
get_movie_titles_from_books(num_pages): This function retrieves movie titles based on the keyword "based-on-novel" from the IMDb database using the IMDbPy library/IMDb API. It iterates over a specified number of pages of search results (controlled by the “num_pages” parameter), collects the titles of movies found on each page, and adds them to a list. It returns the list of movie titles.
convert_to_decimal(value): This function converts the rating value to a floating point number using regular expressions for each of the three ratings collected per movie in the following function. The string rating value is the input, and the function returns a floating point value for the rating (scaled from 0-5) or None if there is no rating or there was an error converting the value.

get_movie_ratings(title): This function retrieves ratings and genre information for a movie title using the OMDB API through a request for each title that is inputted as a parameter. After accessing the title, ratings, and genres from the response data, it also calls the convert_to_decimal function to convert the rating data. The output is either a tuple with the values for title, rating_decimals, and genres, a tuple with the values "Error: Movie not found," None, None, or a tuple with the values "Error: Unable to access API", None, None.
create_database(): This function connects to the "ratings" database and creates the "Genres" table and the "Movie Ratings" table if they do not exist. There is no parameter or return value.
insert_data(title, ratings, genres): This function inserts the primary genre of a movie into the "Genres" table and it inserts the movie title and all movie ratings into "Movie Ratings" table. The parameters are a movie title string, a dictionary with the keys as strings of the type of rating and the values as floating point numbers, and a string of all the associated genres. There is no return value.
get_movies_with_genres(): This function retrieves data from the ratings database by executing an SQL query that joins the "Movie Ratings" table with the "Genres" table on the genre_id column. There is no parameter, but the function returns a list of tuples with the following values: movie title string, IMDb rating (float or None), Rotten Tomatoes rating (float or none), Metacritic rating (float or None), and the primary genre string.
main(): This collects a list of movie adaptations titles, calls the create_database() function to create a table Movie ratings, inserts new movie ratings into a database with their respective genres.

googbooks.py

read_api_key(file): This function opens a text file that contains the Google Books API key and returns the value.
create_googlebooks_ratings_table(): This function connects to the database and creates a GoogleBooks Ratings table if it doesn't already exist.
insert_googlebooks_rating(title_id, googlebooks_rating): This function takes in a title id and GoogleBooks rating, connects to the database, and inserts the GoogleBooks rating with the corresponding title id into the GoogleBooks Ratings table.
get_book_ratings(title): This function takes in a title, creates a URL to query the GoogleBooks API, iterates through all the items in the API response, searches for the titles and ratings of each item, and returns the rating of the closest match with a rating.
main(): This function calls the create_googlebooks_ratings_table() function to create a table, connects to the database, stores the titles in Movie Ratings that aren't in GoogleBooks Ratings into a list, iterates through each title, and inserts each title's ratings by using get_book_ratings() and insert_googlebooks_rating().

open_library.py

get_title(movie_titles): This function takes in a list of movie titles that are pulled from the "Movie Ratings" table in the "ratings" database. It returns a new list where each title is split by spaces and then the words are joined by a plus sign so that they are in the correct format to be added to the url.
get_book_ratings(book_list): This function iterates over each title in book_list, which is inputted as a parameter. Through a request for each title using the Open Library API, it retrieves the average rating. It accesses the title and rating for each title through the response data, and returns a list of tuples with each tuple containing the title and the corresponding Open Library rating.
create_database(): This function connects to the "ratings" database and creates the "Open Library Ratings" table if it does not exist.
find_best_match(title, cur): This function uses an SQL query to select the "title" and "title_id" columns in the "Movie Ratings" table. A loop iterates over each row fetched from the query result and the title and title_id are extracted from each row. It uses the fuzz.partial_ratio() function from the fuzzywuzzy library to calculate a similarity ratio between the title string extracted from the "Movie Ratings" table and the inputted title string as a parameter. The title from the "Movie Ratings" table, the inputted title, and the similarity ratio are all appended to a new list, "matches" as a tuple. After iterating through all the rows, the function returns the title id of the tuple with the highest similarity ratio using the max() function and a custom key that extracts the similarity ratio from each tuple. If "matches" is empty, indicating no matches were found in the database, the function returns "None".
insert_ratings(rating_list): This function takes in a list of tuples containing book titles and their corresponding ratings. It iterates over each tuple in the inputted "rating_list" and extracts each book title and the rating. Then, the function "find_best_match" is called to find the best matching title from the "Movie Ratings" title for the current title from "rating_list". If a matching title is found, then a SQL command is executed to insert or replace the title id and its corresponding rating into the "Open Library Ratings" table. If no matching title is found, then a message is printed that indicates no matching title was found.
main(): This function uses a SQL query to select titles from the "Movie Ratings" table for which we haven't retrieved ratings from Open Library yet. It then creates a list of titles that we need to retrieve the ratings for. It then runs through the "get_title", "get_book_ratings", and "insert_ratings" functions to insert the ratings into the "Open Library Ratings" table.

calculations.py

get_genres(): This function retrieves genres from the database "ratings.db" in order to extract and return a list of genres that were collected in the movie_ratings table. There is no parameter.
calcualte_average_ratings(genre): This function first attempts to match the parameter–the genre–to its genre id. If its id is not found, the function returns None. If it is found, the function calculates the sum of all ratings and the number of ratings found for each of the following categories: IMDb, Rotten Tomatoes, and Metacritic. It then divides the sum of total ratings and

the count of each category to calculate the average for each type of rating. The function then returns the calculated average ratings (IMDb, Rotten Tomatoes, Metacritic) as a tuple (avg_imdb, avg_rotten_tomatoes, avg_metacritic).
plot_movie_ratings_by_genre(): This function generates a bar chart visualizing the average ratings for each movie genre across IMDb, Rotten Tomatoes, and Metacritic. It iterates over each genre in the genres list and calls the calculate_average_ratings() function to obtain the average ratings for the current genre. There is no parameter and no return value.
calculate_average_book_rating(): This function uses a SQL query to select the title and IMDB rating from the "Movie Ratings" table, along with the averaged rating from the "Open Library Ratings" table and the "Google Books Ratings" table when the title_id matched across all three tables. The function skips over the selection if any of the ratings are null and limits the selection to only 10 titles. It returns a list of tuples containing the title, the movie rating, and the average book rating.
plot_compare_ratings(): This function calls the "calculate_average_book_rating" function to get a list of tuples containing a title, the movie rating, and the average book rating. The titles, movie ratings, and average book ratings are separated into three separate lists by the use of the zip function. Then, a bar graph is created using matplotlib. Two bars are created for each title, one representing the movie rating and the other representing the average book rating.
plot_book_ratings(): This function generates a bar chart comparing the ratings of books from Open Library and Google Books, joining both tables on title_id from the "Movie Ratings" Database; it limits the graph to plot 20 different titles. There is no parameter or return value.

Resource Documentation

04/03/2024	We needed a list of movie titles that were based on novels in order to have titles to add to our database.	IMDb API	After accessing the API, we were able to get access to thousands of up-to-date movie and show titles that are based on novels.
04/18/2024	Our movie titles and their corresponding book titles did not match exactly many times.	FuzzyWuzzy Python Library	After accessing the library, we were able to see many more matches between the titles in the movies table and the titles in Open Library and Google Books.
04/18/2024	The plot "Average Ratings by Genre" was not formatting well with text overlapping.	ChatGPT	After asking to help format the data, we were able to adjust the layout and tick label positioning so the chart looks more appealing and is readable.

04/18/2024	The book titles were not matching their corresponding movie adaptation title exactly.	ChatGPT	After asking to help match book titles to movie titles when the titles weren’t exactly the same, we found that we could use the Fuzzy Wuzzy library and match strings through the fuzz.partial_ratio() function.
04/23/2024	The SQL query in googbooks.py’s main() wasn’t joining the tables correctly.	ChatGPT	After asking to help debug the code, we found that we needed to use LEFT JOIN instead of just JOIN to correctly get the titles we wanted.