AJ deVaux & Siya Shah

Professor Ericson

SI 206

21 April 2023

Github Link: https://github.com/viraux/206-Final

https://github.com/siya204/206_final_API.git

Final Report

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

Our main goal for this project was to determine if the duration of popular media has gone up in the past 20 years or so. To do this, we planned to work with the IMDb-API and the spotify API. We wanted to collect the runtime or song duration for the top movies/songs (around 25) per year to compare them with one another. We also looked into collecting other data such as genre to compare with runtime as well or rating to compare with overall critical score.

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

For the IMDb-API, we were able to collect data from the top 25 movies (but up to the top 100) per year. Included in this collected data was a unique id, the movie title, runtime, release year, genre (up to three different ones per movie), age rating, and metacritic score. From this, we were able to create calculations and visualizations to show the average runtime by year and by genre along with the average score by age rating. For the main goal we started with, it showed a slight increase in duration over the past 20 years excluding the outlier of 2020. We then used the

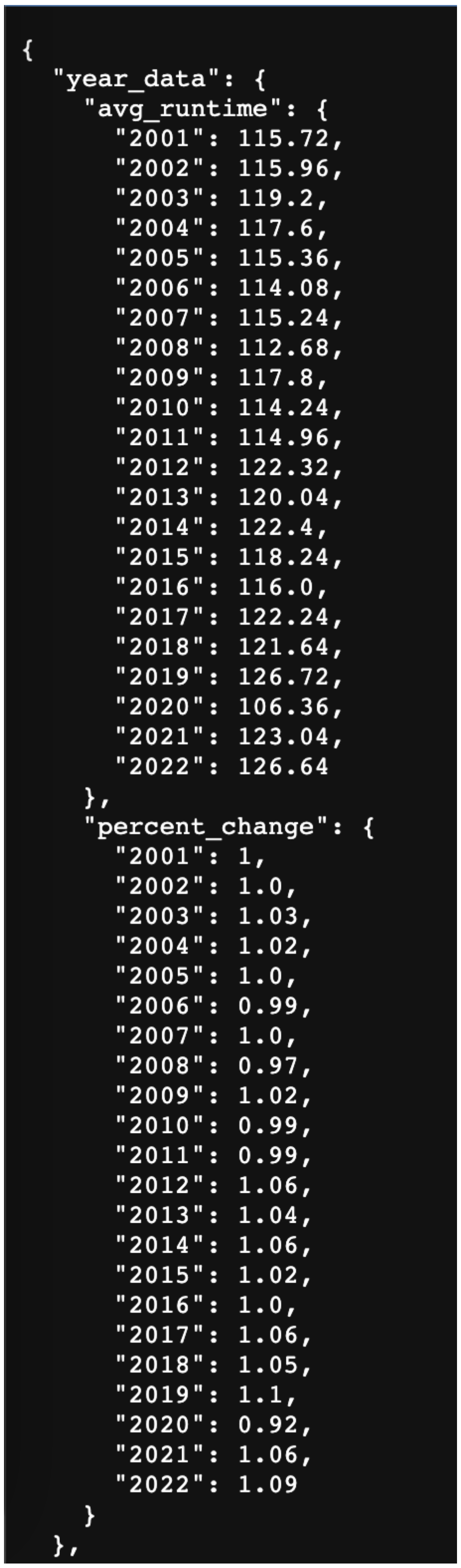
Spotipy library for the spotify api in order to get similar data from songs throughout the years of 2001-2022. Here, we only recorded the title, duration, and release year in order to compare it with the data from the IMDb-API. From all of this, we converted the data to percent differences based on the first year's value (which can be seen in the visualization). This led to us finding that song duration has actually decreased over the last 20ish years while movie runtime has increased slightly. Being able to see this was very satisfying as it was directly tied to our initial goal. Finally, we web scraped data from the Goodreads website. We decided to collect data on the ratings of top authors, as well as book ratings by top authors on the website and create visualizations based on top author ratings and book ratings. Included in this collected data was a unique id, author name, author rating, book rating and book name. From the web scraped data, we created 4 data visualizations. The first one is a bar graph for top rated authors on the website and their ratings, the second one is a scatter plot of book ratings by these top authors, the third one was taking the same data and creating a distribution plot of book ratings, and the fourth one was creating a heatmap of book ratings of book titles by top authors.

3. The problems that you faced (10 points)

A major problem we faced with collecting data on movies was finding an API that let us search for movies by year. We tried many API's, including the OMDb API and track API, but none let me conduct this search. Thankfully, the IMDb-API offered this functionality through its advanced search, which also meant we could search the top movies in each year as well. Additionally, we ran into some issues with getting our visualizations where we wanted them, and searching our issues on forums, like stackoverflow, was very useful in solving these issues.

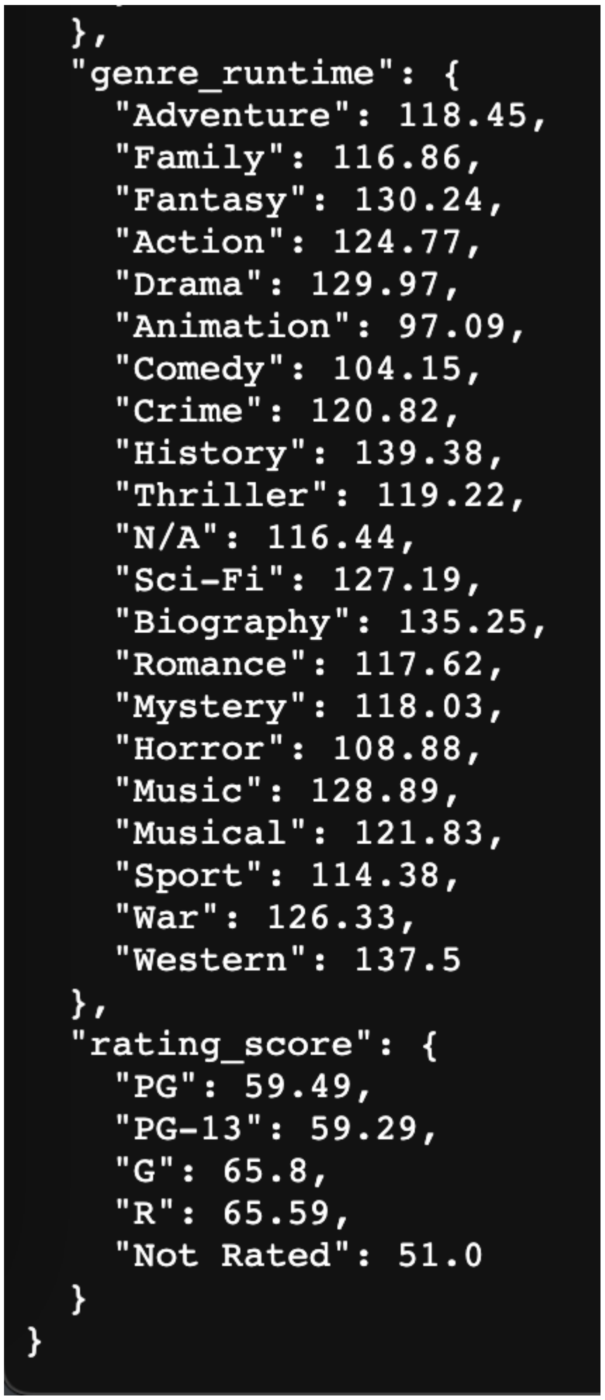
Another problem we faced was with authorization of the Spotipy API, specifically when trying to create an app for a step in the authentication process. This was able to be solved by watching some of their videos on the documentation page. But, while we were having this issue, we also decided to webscrape goodreads. This ended up working out conceptually since we wanted a similar theme between two sets of visualizations, and though our initial plan was to analyze and visualize popular media through movies and songs data, we ended up using movie, songs, and book data.

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



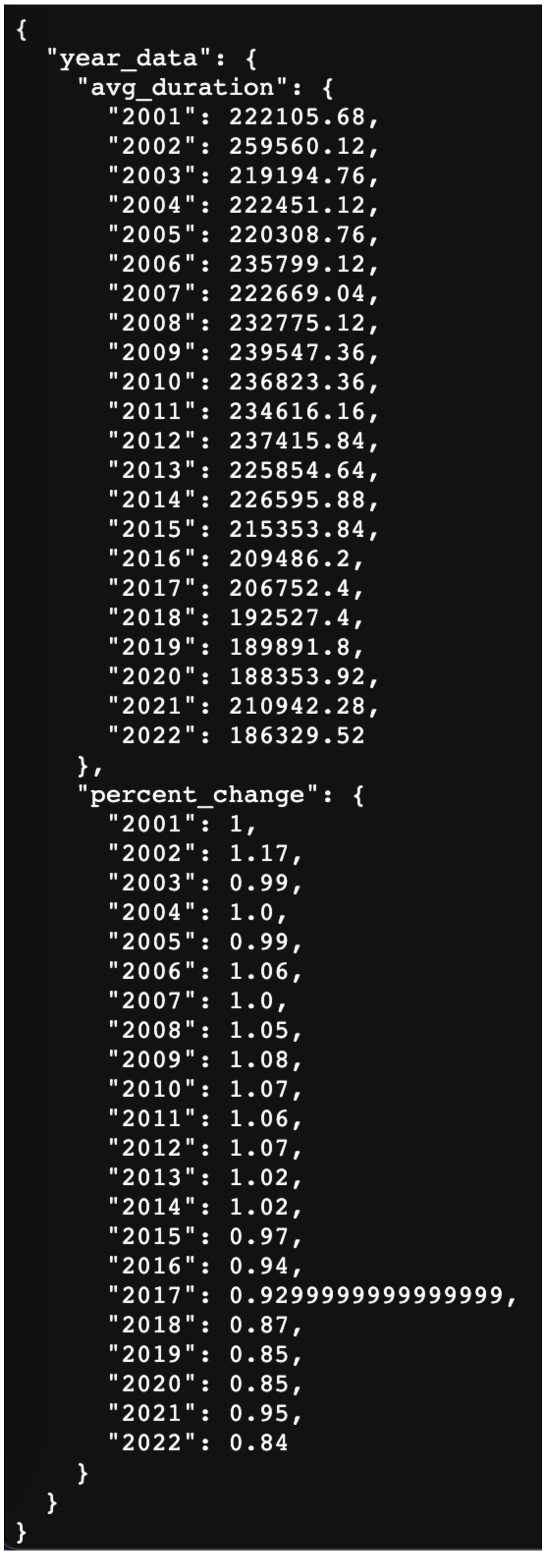
```
{
  "year_data": {
    "avg_runtime": {
      "2001": 115.72,
      "2002": 115.96,
      "2003": 119.2,
      "2004": 117.6,
      "2005": 115.36,
      "2006": 114.08,
      "2007": 115.24,
      "2008": 112.68,
      "2009": 117.8,
      "2010": 114.24,
      "2011": 114.96,
      "2012": 122.32,
      "2013": 120.04,
      "2014": 122.4,
      "2015": 118.24,
      "2016": 116.0,
      "2017": 122.24,
      "2018": 121.64,
      "2019": 126.72,
      "2020": 106.36,
      "2021": 123.04,
      "2022": 126.64
    },
    "percent_change": {
      "2001": 1,
      "2002": 1.0,
      "2003": 1.03,
      "2004": 1.02,
      "2005": 1.0,
      "2006": 0.99,
      "2007": 1.0,
      "2008": 0.97,
      "2009": 1.02,
      "2010": 0.99,
      "2011": 0.99,
      "2012": 1.06,
      "2013": 1.04,
      "2014": 1.06,
      "2015": 1.02,
      "2016": 1.0,
      "2017": 1.06,
      "2018": 1.05,
      "2019": 1.1,
      "2020": 0.92,
      "2021": 1.06,
      "2022": 1.09
    }
  },
```

Movies.json



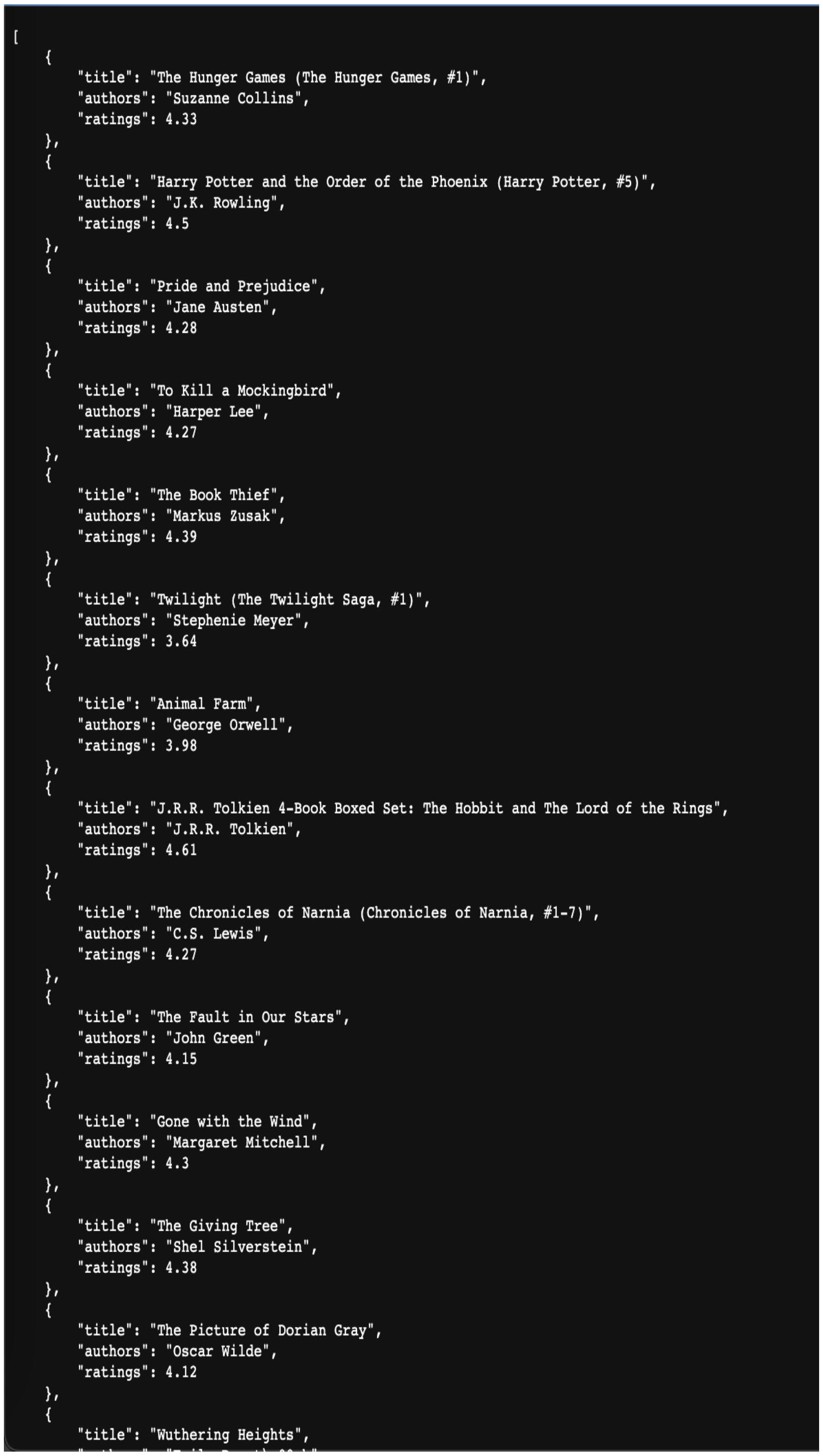
```
  },
  "genre_runtime": {
    "Adventure": 118.45,
    "Family": 116.86,
    "Fantasy": 130.24,
    "Action": 124.77,
    "Drama": 129.97,
    "Animation": 97.09,
    "Comedy": 104.15,
    "Crime": 120.82,
    "History": 139.38,
    "Thriller": 119.22,
    "N/A": 116.44,
    "Sci-Fi": 127.19,
    "Biography": 135.25,
    "Romance": 117.62,
    "Mystery": 118.03,
    "Horror": 108.88,
    "Music": 128.89,
    "Musical": 121.83,
    "Sport": 114.38,
    "War": 126.33,
    "Western": 137.5
  },
  "rating_score": {
    "PG": 59.49,
    "PG-13": 59.29,
    "G": 65.8,
    "R": 65.59,
    "Not Rated": 51.0
  }
}
```

Movies.json



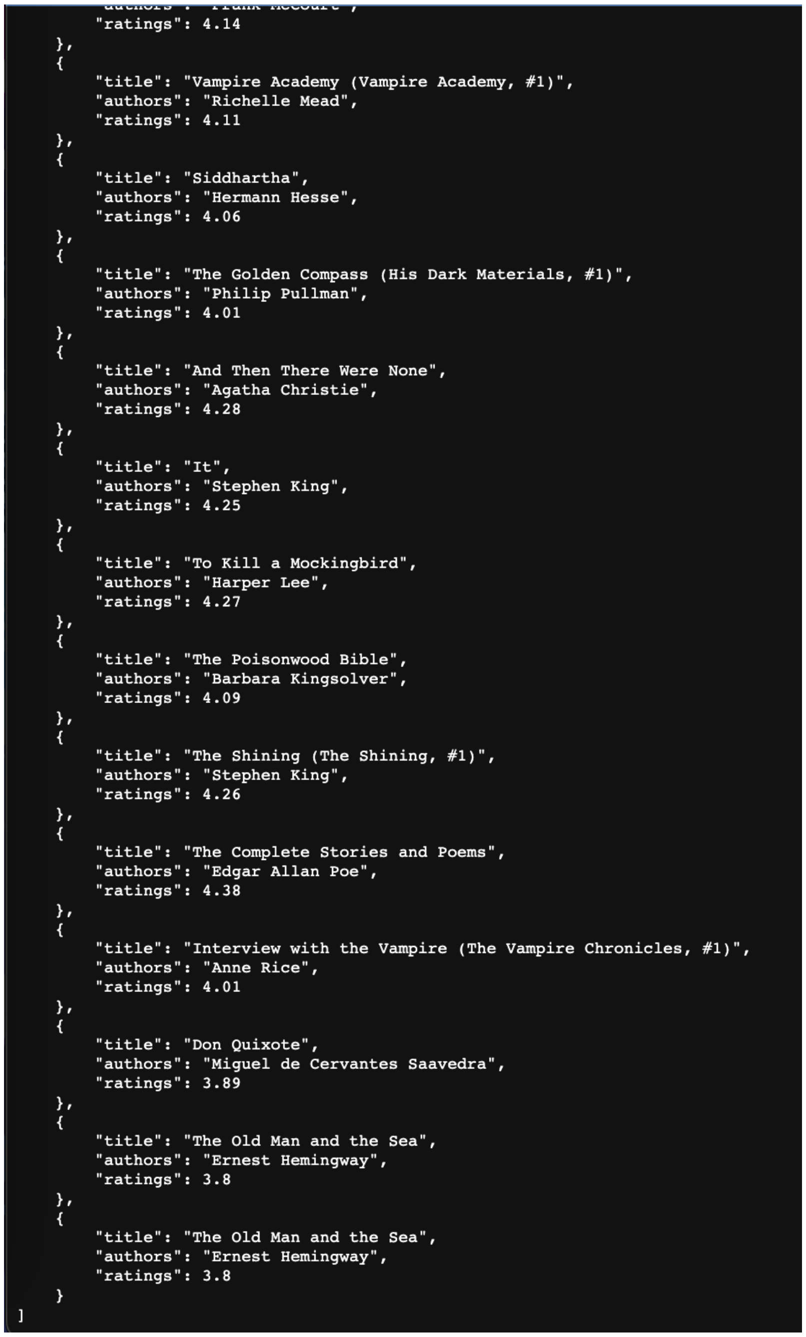
```
{
  "year_data": {
    "avg_duration": {
      "2001": 222105.68,
      "2002": 259560.12,
      "2003": 219194.76,
      "2004": 222451.12,
      "2005": 220308.76,
      "2006": 235799.12,
      "2007": 222669.04,
      "2008": 232775.12,
      "2009": 239547.36,
      "2010": 236823.36,
      "2011": 234616.16,
      "2012": 237415.84,
      "2013": 225854.64,
      "2014": 226595.88,
      "2015": 215353.84,
      "2016": 209486.2,
      "2017": 206752.4,
      "2018": 192527.4,
      "2019": 189891.8,
      "2020": 188353.92,
      "2021": 210942.28,
      "2022": 186329.52
    },
    "percent_change": {
      "2001": 1,
      "2002": 1.17,
      "2003": 0.99,
      "2004": 1.0,
      "2005": 0.99,
      "2006": 1.06,
      "2007": 1.0,
      "2008": 1.05,
      "2009": 1.08,
      "2010": 1.07,
      "2011": 1.06,
      "2012": 1.07,
      "2013": 1.02,
      "2014": 1.02,
      "2015": 0.97,
      "2016": 0.94,
      "2017": 0.9299999999999999,
      "2018": 0.87,
      "2019": 0.85,
      "2020": 0.85,
      "2021": 0.95,
      "2022": 0.84
    }
  }
}
```

Songs.json



```
[
    {
        "title": "The Hunger Games (The Hunger Games, #1)",
        "authors": "Suzanne Collins",
        "ratings": 4.33
    },
    {
        "title": "Harry Potter and the Order of the Phoenix (Harry Potter, #5)",
        "authors": "J.K. Rowling",
        "ratings": 4.5
    },
    {
        "title": "Pride and Prejudice",
        "authors": "Jane Austen",
        "ratings": 4.28
    },
    {
        "title": "To Kill a Mockingbird",
        "authors": "Harper Lee",
        "ratings": 4.27
    },
    {
        "title": "The Book Thief",
        "authors": "Markus Zusak",
        "ratings": 4.39
    },
    {
        "title": "Twilight (The Twilight Saga, #1)",
        "authors": "Stephenie Meyer",
        "ratings": 3.64
    },
    {
        "title": "Animal Farm",
        "authors": "George Orwell",
        "ratings": 3.98
    },
    {
        "title": "J.R.R. Tolkien 4-Book Boxed Set: The Hobbit and The Lord of the Rings",
        "authors": "J.R.R. Tolkien",
        "ratings": 4.61
    },
    {
        "title": "The Chronicles of Narnia (Chronicles of Narnia, #1-7)",
        "authors": "C.S. Lewis",
        "ratings": 4.27
    },
    {
        "title": "The Fault in Our Stars",
        "authors": "John Green",
        "ratings": 4.15
    },
    {
        "title": "Gone with the Wind",
        "authors": "Margaret Mitchell",
        "ratings": 4.3
    },
    {
        "title": "The Giving Tree",
        "authors": "Shel Silverstein",
        "ratings": 4.38
    },
    {
        "title": "The Picture of Dorian Gray",
        "authors": "Oscar Wilde",
        "ratings": 4.12
    },
    {
        "title": "Wuthering Heights",
```

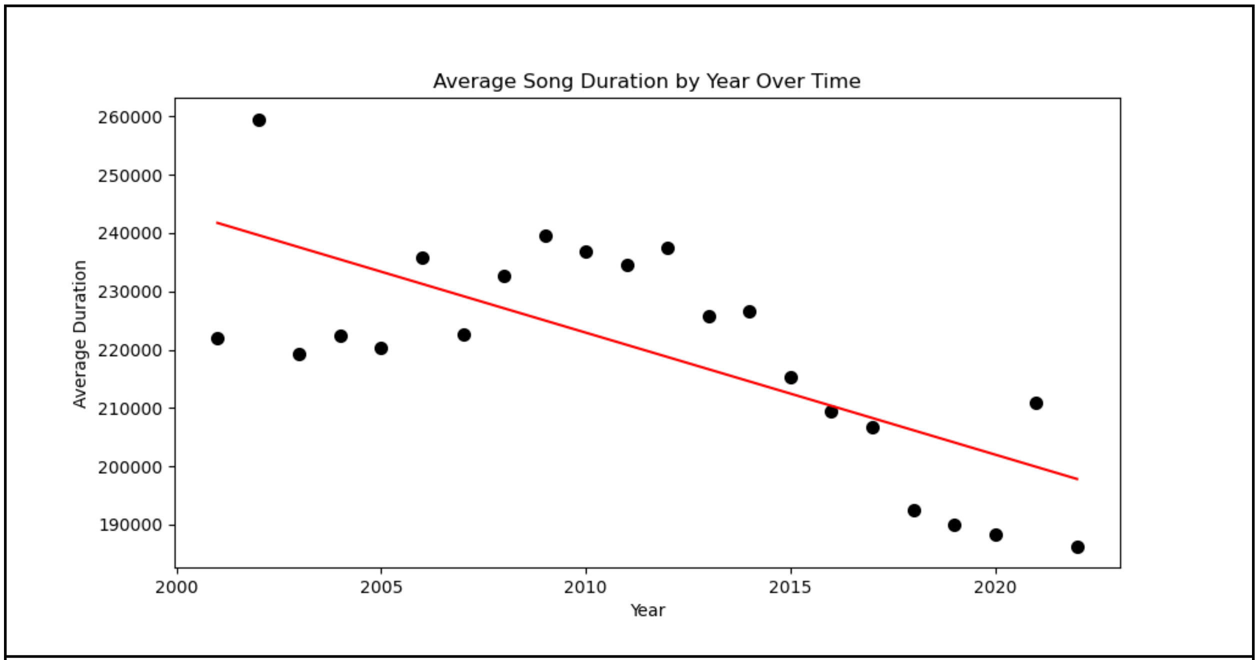
Book_ratings.json

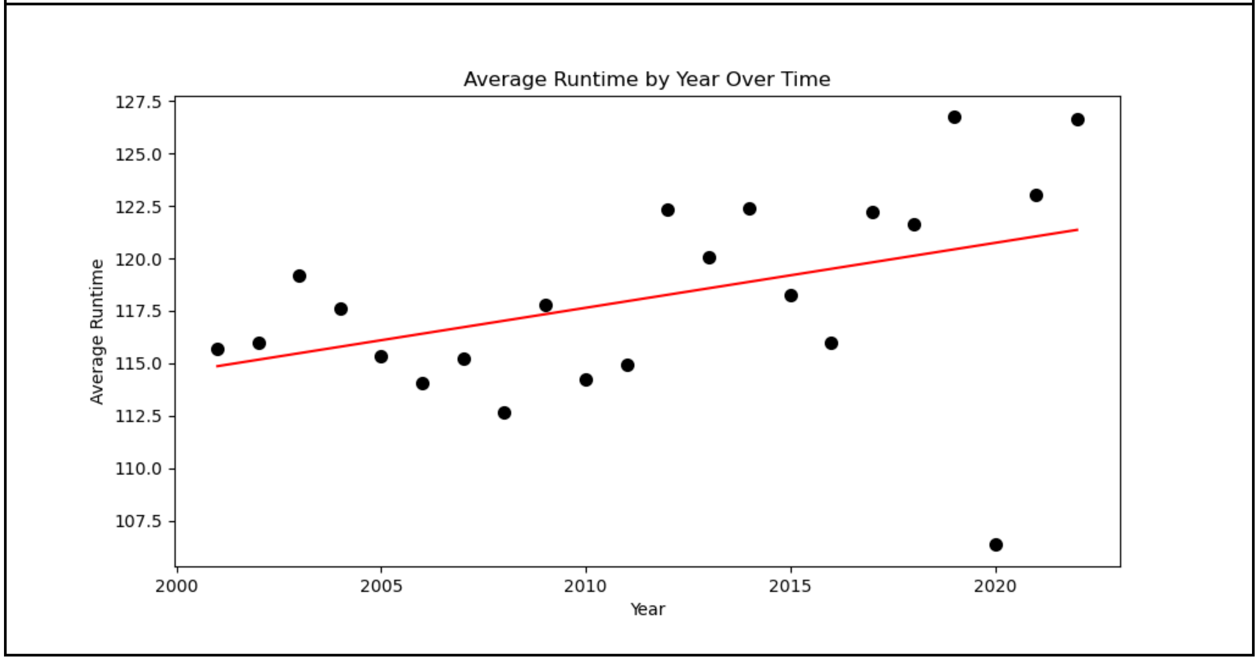


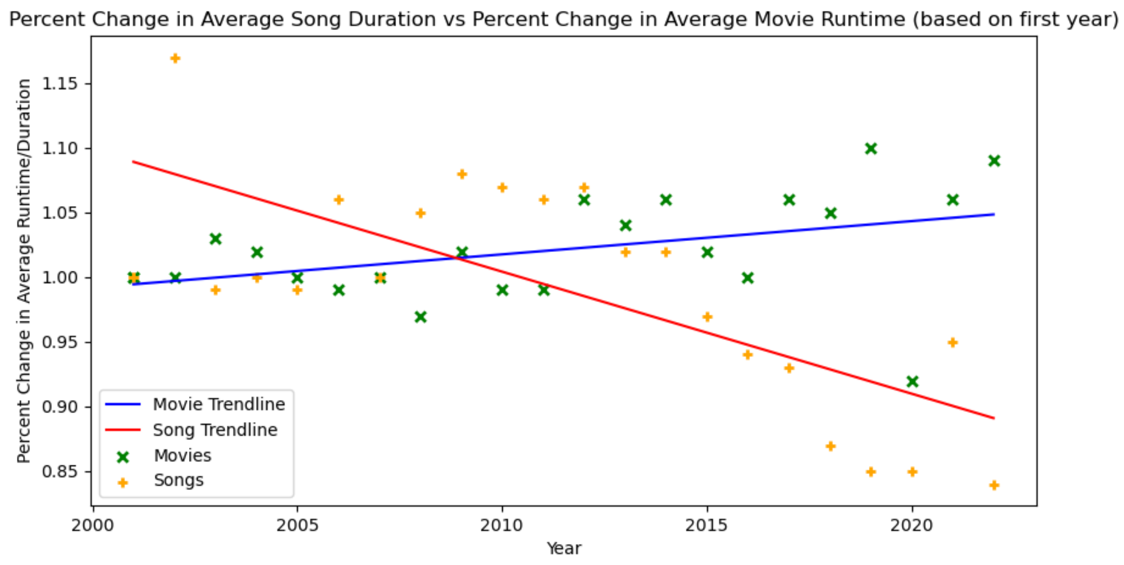
```
        "ratings": 4.14
    },
    {
        "title": "Vampire Academy (Vampire Academy, #1)",
        "authors": "Richelle Mead",
        "ratings": 4.11
    },
    {
        "title": "Siddhartha",
        "authors": "Hermann Hesse",
        "ratings": 4.06
    },
    {
        "title": "The Golden Compass (His Dark Materials, #1)",
        "authors": "Philip Pullman",
        "ratings": 4.01
    },
    {
        "title": "And Then There Were None",
        "authors": "Agatha Christie",
        "ratings": 4.28
    },
    {
        "title": "It",
        "authors": "Stephen King",
        "ratings": 4.25
    },
    {
        "title": "To Kill a Mockingbird",
        "authors": "Harper Lee",
        "ratings": 4.27
    },
    {
        "title": "The Poisonwood Bible",
        "authors": "Barbara Kingsolver",
        "ratings": 4.09
    },
    {
        "title": "The Shining (The Shining, #1)",
        "authors": "Stephen King",
        "ratings": 4.26
    },
    {
        "title": "The Complete Stories and Poems",
        "authors": "Edgar Allan Poe",
        "ratings": 4.38
    },
    {
        "title": "Interview with the Vampire (The Vampire Chronicles, #1)",
        "authors": "Anne Rice",
        "ratings": 4.01
    },
    {
        "title": "Don Quixote",
        "authors": "Miguel de Cervantes Saavedra",
        "ratings": 3.89
    },
    {
        "title": "The Old Man and the Sea",
        "authors": "Ernest Hemingway",
        "ratings": 3.8
    },
    {
        "title": "The Old Man and the Sea",
        "authors": "Ernest Hemingway",
        "ratings": 3.8
    }
]
```

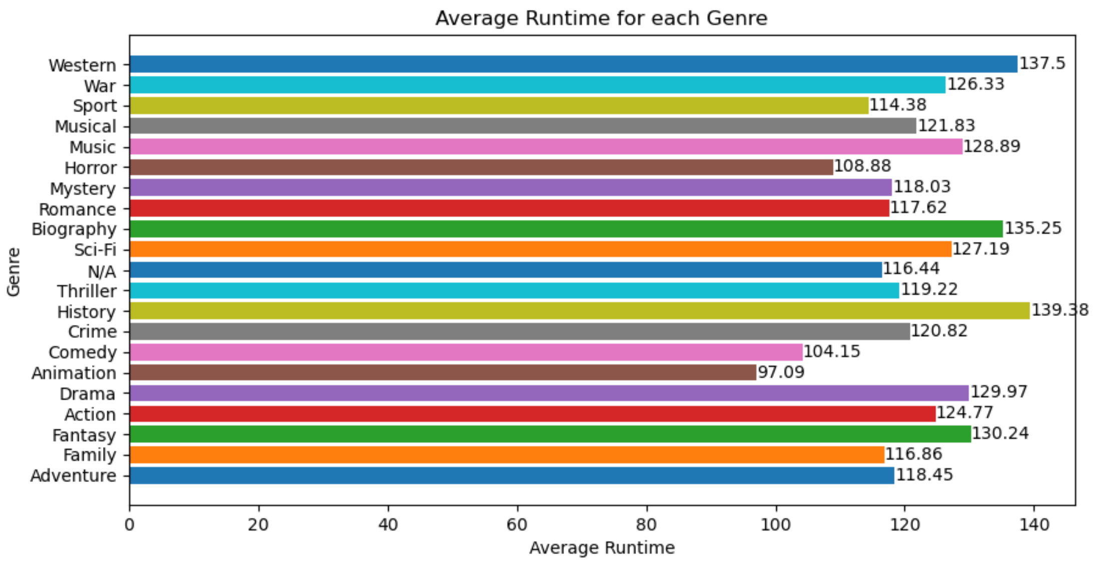
Book_ratings.json

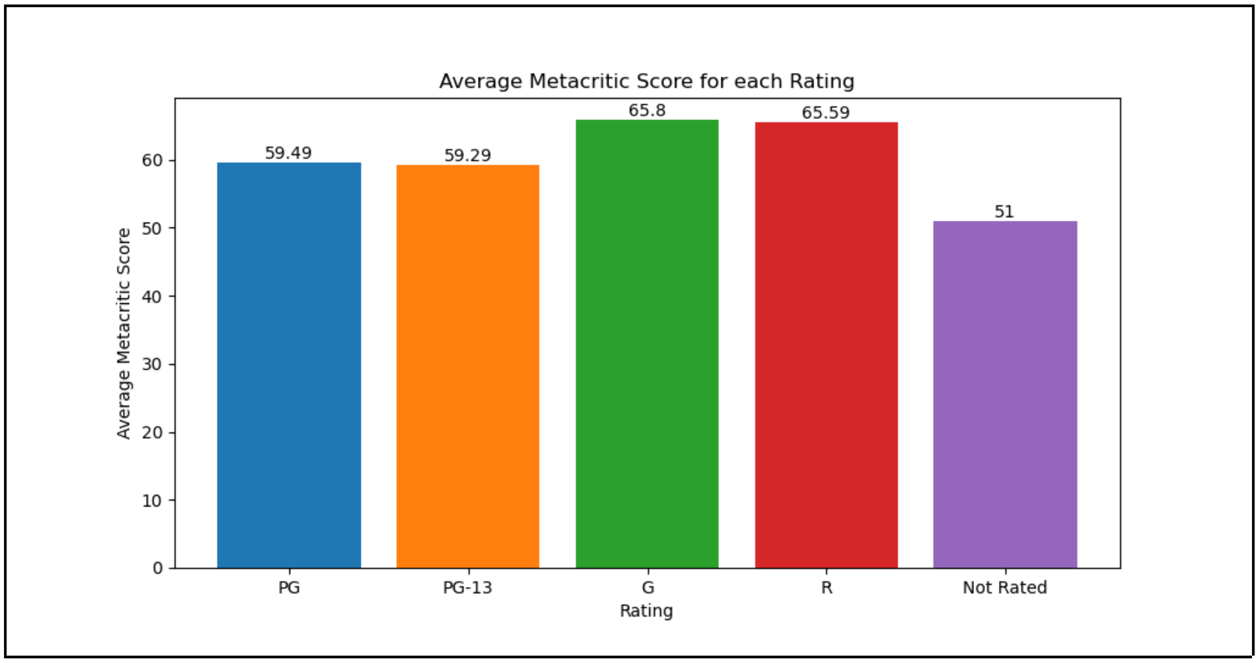
5. The visualization that you created (i.e. screenshot or image file) (10 points)

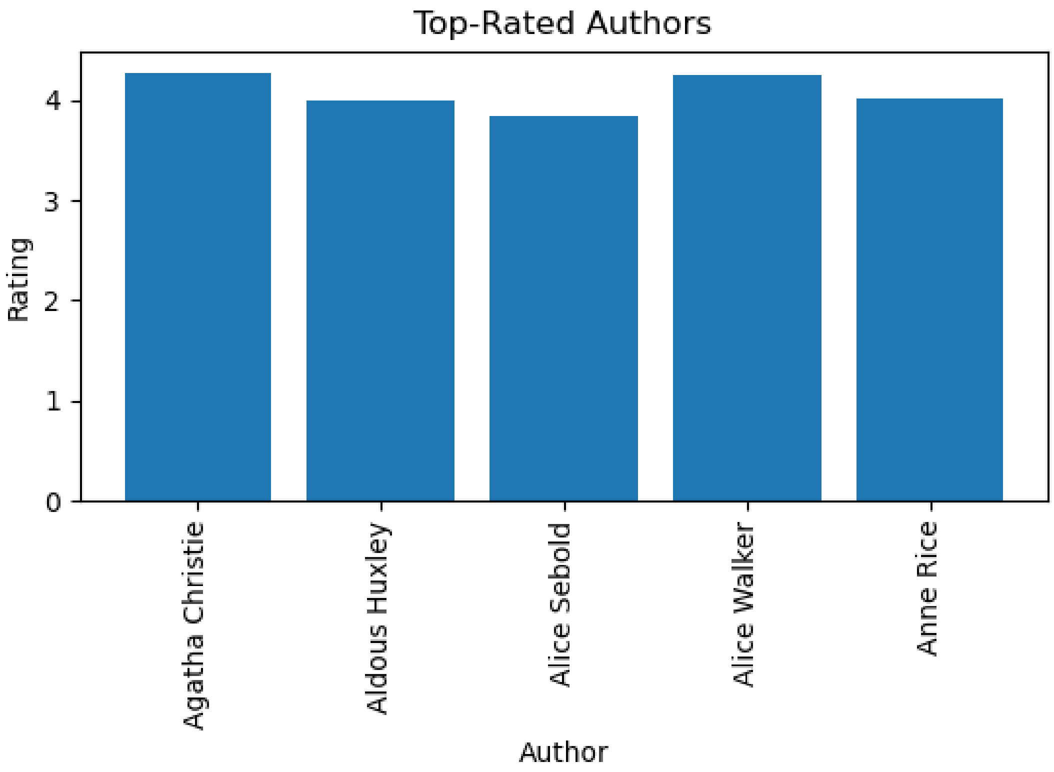

Average Song Duration by Year Over Time
Average Duration
Year
260000
250000
240000
230000
220000
210000
200000
190000
2000
2005
2010
2015
2020

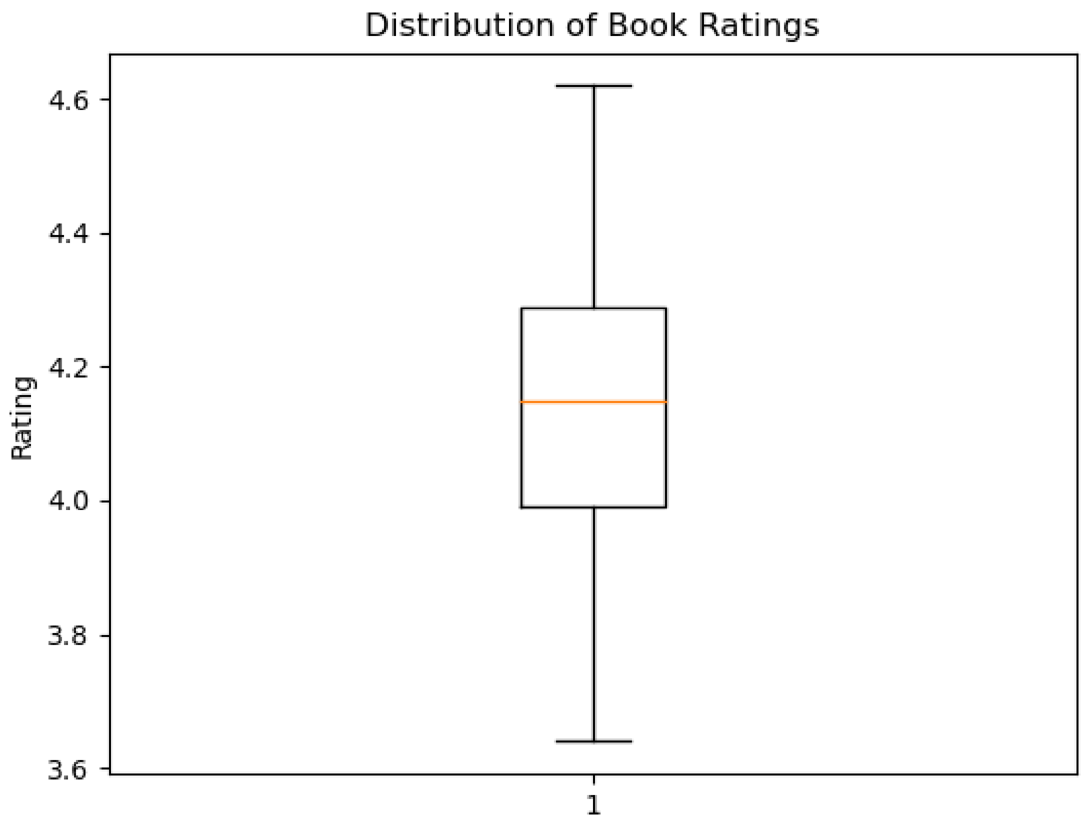

Average Runtime by Year Over Time
Average Runtime
Year
127.5
125.0
122.5
120.0
117.5
115.0
112.5
110.0
107.5
2000
2005
2010
2015
2020

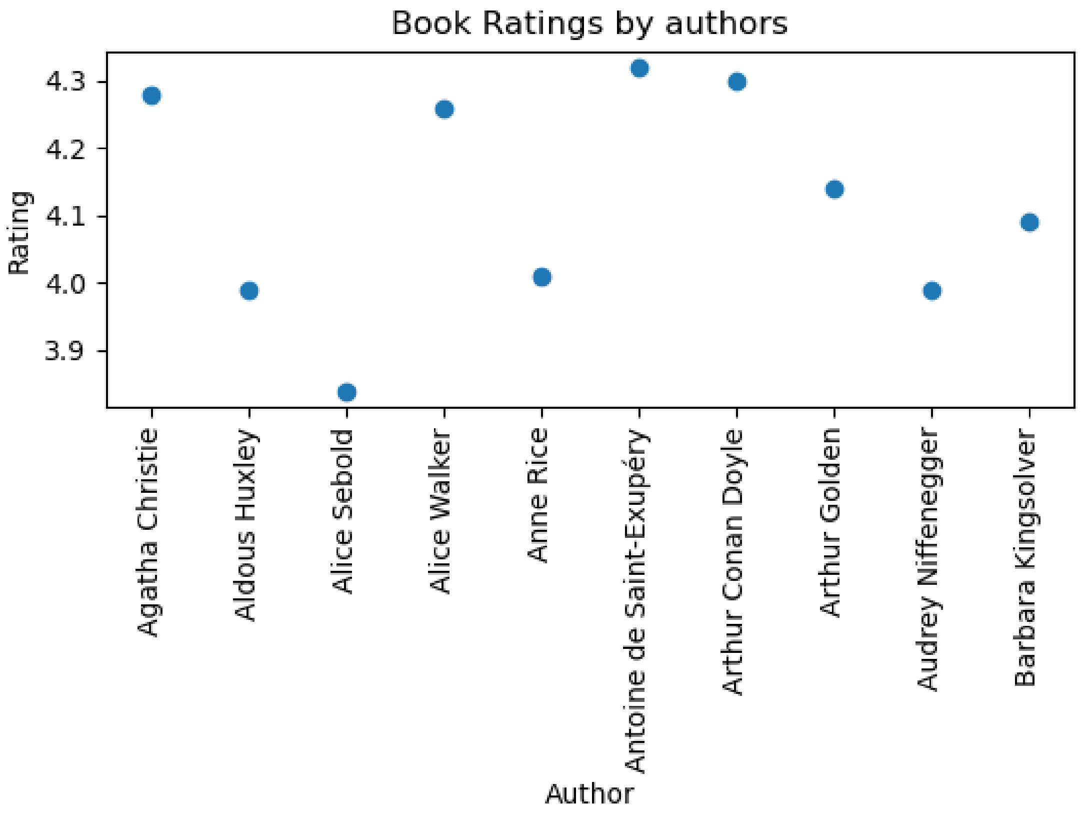

Percent Change in Average Song Duration vs Percent Change in Average Movie Runtime (based on first year)
Percent Change in Average Runtime/Duration
1.15
1.10
1.05
1.00
0.95
0.90
0.85
Year
2000
2005
2010
2015
2020
Movie Trendline
Song Trendline
Movies
Songs

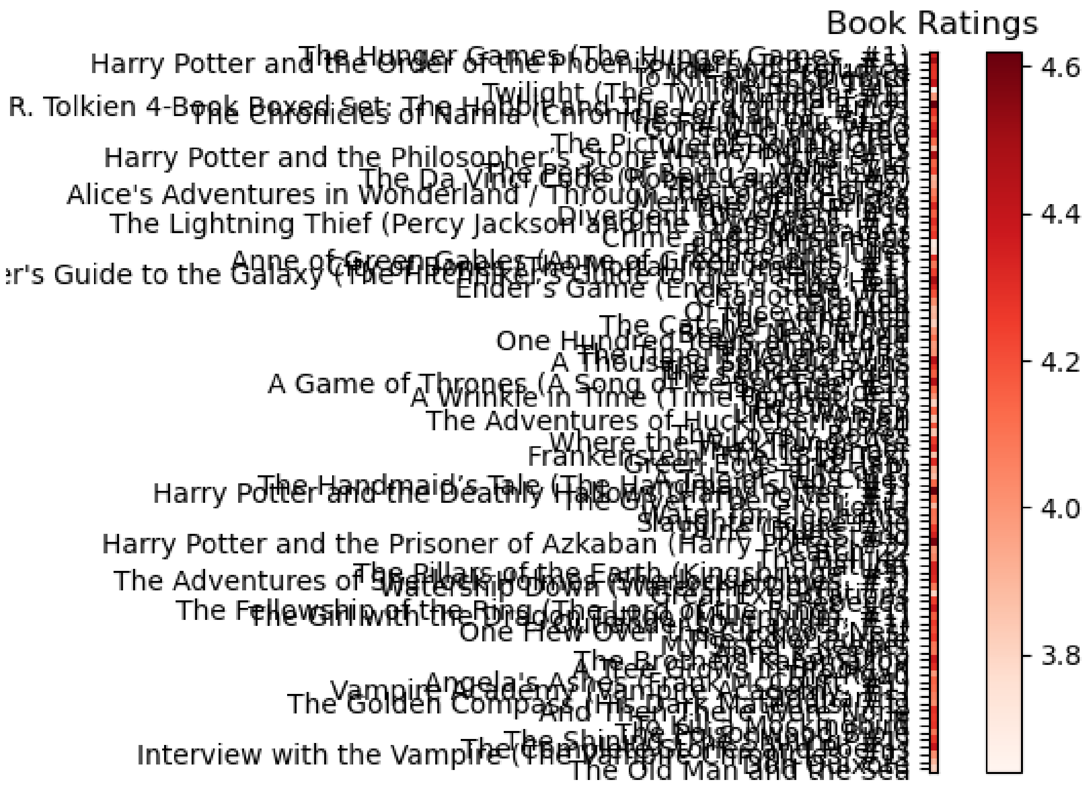

Average Runtime for each Genre
Genre
Western
War
Sport
Musical
Music
Horror
Mystery
Romance
Biography
Sci-Fi
N/A
Thriller
History
Crime
Comedy
Animation
Drama
Action
Fantasy
Family
Adventure
137.5
126.33
114.38
121.83
128.89
108.88
118.03
117.62
135.25
127.19
116.44
119.22
139.38
120.82
104.15
97.09
129.97
124.77
130.24
116.86
118.45
Average Runtime
0
20
40
60
80
100
120
140


Average Metacritic Score for each Rating
Average Metacritic Score
Rating
PG
PG-13
G
R
Not Rated
59.49
59.29
65.8
65.59
51
0
10
20
30
40
50
60


Top-Rated Authors
Rating
Author
Agatha Christie
Aldous Huxley
Alice Sebold
Alice Walker
Anne Rice
0
1
2
3
4


Distribution of Book Ratings
Rating
4.6
4.4
4.2
4.0
3.8
3.6
1


Book Ratings by authors
Rating
4.3
4.2
4.1
4.0
3.9
Agatha Christie
Aldous Huxley
Alice Sebold
Alice Walker
Anne Rice
Antoine de Saint-Exupéry
Arthur Conan Doyle
Arthur Golden
Audrey Niffenegger
Barbara Kingsolver
Author



6. Instructions for running your code (10 points)

1. The first step is to run either one of or both movie_data.py and/or music_data.py the desired amount of times to receive enough data for the database. For example, if you wanted to collect 50 songs and movies from 2001 to 2005, you would run both movie_data.py and music_data.py 10 times, inputting each of the years twice (in any order).
    a. Note, for the music_data.py file, you must pip install Spotify using the following instructions.

2. After collecting all the data from movie_data.py and/or music_data.py, you then need to run movie_calc.py and/or music_calc.py respectively. This will do all the calculations with the given data in the database for each category respectively and create all the individual visualizations with the new calculations.
3. Finally, if you have run both movie_data.py and music_data.py to input data in the database for each category and also movie_calc.py and music_cacl.py to create the data files, then you can run percent_graph.py. This will use the data files to read in the data and create a percent difference graph that includes the change in duration/runtime for both songs and movies.

Instructions for 'book_data.py':

This code file is a web scraper that extracts book data from the Goodreads website and stores it in an SQLite database. To run this code, follow these steps:

Step 1: Install Dependencies

Make sure you have the following Python packages installed:

sqlite3

json

os

matplotlib

re

requests

BeautifulSoup

You can install these packages using pip, the Python package manager, by running the following commands:

```
pip install sqlite3

pip install json

pip install matplotlib

pip install requests

pip install BeautifulSoup4
```

Step 2: Set up the Database

Before running the code, make sure you have an SQLite database file (e.g., final.db) created in the same directory as the code file. You can create an SQLite database using SQLite command-line tools or an SQLite client, or you can use a library like sqlite3 in Python to create the database programmatically.

Step 3: Run the Code

Open a command-line or terminal window, navigate to the directory where the goodreads_scraper.py file is located, and run the following command:

```
python book_data.py
```

The code will connect to the SQLite database, create a table named "book_ratings" if it doesn't exist, scrape data from the Goodreads website, and store it in the database. The scraped data includes book titles, authors, and ratings. The code will write to the database every 25 entries and close the connection after all the data has been processed.

Instruction for ‘books_visualizations.py’:

This code file retrieves book data from an SQLite database (e.g., final.db) created by the goodreads_scraper.py code and visualizes it using matplotlib. To run this code, follow these steps:

Step 1: Install Dependencies

Make sure you have the following Python packages installed:

sqlite3

matplotlib

You can install these packages using pip, the Python package manager, by running the following commands:

```
pip install sqlite3
pip install matplotlib
```

Step 2: Run the Code

Open a command-line or terminal window, navigate to the directory where the books_visualizations.py file is located, and run the following command:

python books_visualizations.py

The code will connect to the SQLite database, retrieve book data from the "book_ratings" table, and visualize it using matplotlib. The visualization includes a bar chart showing the top 10 books based on their average ratings, and a scatter plot showing the relationship between ratings and the number of books by an author. The generated plots will be displayed on the screen.

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

- ➔ setUpDatabase(db_name) -> cur, conn
    - ◆ Takes in the name of the database as a string that is being created/added to/taken from in the file. Returns a cursor and connection to the database.
- ➔ createTables(cur,conn) -> None
    - ◆ Takes a cursor and connection to the database as parameters and creates the necessary tables for the data that will be collected in the given file. Returns None.
- ➔ collectData(key, year, index) [For movies] -> list
    - ◆ Takes in the API key (string), the desired year(int), and the current index (int) to use when collecting data from the API. Collects desired information from the API call made from the parameters and returns a list of 25 tuples that represent rows of data.
- ➔ collectData(sp, year, index) [For songs] -> list
    - ◆ Takes in a connection to the spotify api, the desired year(int), and the current index (int) to use when collecting data from the API. Collects desired information

from the API call made from the spotipy connection and year and returns a list of 25 tuples that represent rows of data.

- ➔ updateSideTables(cur, conn, data_l) -> None
    - ◆ Takes the database cursor and connection along with the list of data from collectData to update the tables with shared integer keys (Genres and Ratings). Inserts any new genres or ratings found and creates a key for them, returns None.
- ➔ updateMainTable(cur,conn,data_l) -> None
    - ◆ Takes the database cursor and connection along with the list of data from collectData to update the main table (Movies or Songs) with the newly collected data. Commits these changes and returns None
- ➔ main() [For movie_data.py and music_data.py] -> returns None
    - ◆ Opens desired database and sets up cur and conn. Prompts the user for the year to collect data from and finds the current number of data already collected from that year to set an index of. Will only continue if a valid year is input or -1 is input to end the program. Then, calls all functions to collect data and update the database.
- ➔ runtimeByYear(cur)/durationByYear(cur) -> dict
    - ◆ Takes a cursor of the database as the only parameter. Uses it to search through all years present in the database for either movies (runtime) or songs (duration) and calculate the average runtime/duration respectively and add it to a dictionary with the years as keys and averages as values. Return that dictionary.
- ➔ percentDif(data) -> dict
    - ◆ Takes data (dictionary) of the average runtime/duration for each year. Uses this data to calculate the percentage change for each year based on the first given year.

Returns these new values as a dictionary with the years as keys and percent change as values

- runtimeByGenre(cur) -> dict
    - Takes a cursor of the database as the only parameter. Uses it to search through all genres present in the database for movies and calculate the average runtime for each. It then adds it to a dictionary with the genres as keys and averages as values. Return that dictionary.
- scoreByRating(cur) -> dict
    - Takes a cursor of the database as the only parameter. Uses it to search through all ratings present in the database for movies and calculate the average score for each. It then adds it to a dictionary with the ratings as keys and scores as values. Return that dictionary.
- plotYear(data) -> None [For both movies and songs]
    - Takes the dictionary of average runtime/duration for each year and plots it on a scatter plot, including a line of best fit. Save this figure and return None.
- plotPercent(data) -> None [For both movies and songs]
    - Takes the dictionary of average percent change in runtime/duration for each year and plots it on a scatter plot, including a line of best fit. Save this figure and return None.
- plotGenre(data) -> None
    - Takes the dictionary of average runtime for each genre and plots it on a horizontal bar graph. Save this figure and return None.
- plotRating(data) -> None

    - ◆ Takes the dictionary of average score for each rating and plots it on a bar graph. Save this figure and return None.
- ➔ writeData(data_l) -> None [For both movies and songs]
    - ◆ Takes in a list of dictionaries representing all calculated data. Writes this data as a json file with keys representing what each dictionary represents. Return None
- ➔ main() [For movie_calc.py and music_calc.py] -> returns None
    - ◆ Runs all above calculations, plotting, and write functions to calculate all the required data, plot it, and then write it to a file. Returns None.
- ➔ plotPercent(movies,songs) -> None
    - ◆ Takes in two dictionaries. One of the average percent changes in runtime for each year (movies) and one for the average percent changes in duration for each year (songs). Plots both dictionaries on a single scatterplot, including a line of best fit. Save this figure and return None.
- ➔ main() [For percent_graph.py] -> returns None
    - ◆ Opens both data files, reads them into dictionaries with json, and calls plotPercent to graph them. Then closes the files and returns None.

Documentation for the file 'books_visualizations.py':

requests.get(url): This function makes a GET request to the specified URL and returns a Response object. The url parameter is a string representing the URL to be requested. The output is a Response object containing the server's response to the request.

BeautifulSoup(response.content, "html.parser"): This function creates a BeautifulSoup object from the HTML content of the Response object. The response.content parameter is the HTML content of the response, and the second parameter specifies the parser to be used, which in this case is "html.parser". The output is a BeautifulSoup object that can be used to extract data from the HTML content.

find(): This method is used to find the first occurrence of a specified element in the BeautifulSoup object. The parameters passed to this method are the element tag name and any attributes to search for. The output is a BeautifulSoup object representing the found element.

find_all(): This method is used to find all occurrences of a specified element in the BeautifulSoup object. The parameters passed to this method are the element tag name and any attributes to search for. The output is a list of BeautifulSoup objects representing the found elements.

text: This property of a BeautifulSoup object is used to extract the text content of the element.

get(): This method is used to retrieve the value of a specified attribute of an element. The parameter passed to this method is the name of the attribute. The output is a string representing the value of the attribute.

strip(): This method is used to remove any leading or trailing whitespaces from a string.

try-except: This is a Python exception handling construct used to catch and handle exceptions. In this code, it is used to handle cases where the rating of a book cannot be converted to a float, by skipping that book using the continue statement.

pd.DataFrame(data={...}): This function is used to create a Pandas DataFrame object from a dictionary. The keys of the dictionary represent the column names, and the values represent the column data. The output is a Pandas DataFrame object.

groupby(): This method is used to group rows of a DataFrame by the values in one or more columns. The parameter passed to this method is the column(s) to group by. The output is a GroupBy object that can be used to perform aggregate functions on the grouped data.

reset_index(): This method is used to reset the index of a DataFrame and convert it back to a regular DataFrame. The output is a DataFrame with a new index.

plt.subplots(): This function is used to create a figure and one or more subplots in Matplotlib. The outputs are a figure object and one or more subplot objects.

ax.bar(): This method is used to create a bar chart in Matplotlib. The parameters passed to this method are the x-axis values (in this case, the authors), the y-axis values (in this case, the ratings), and optional parameters for customizing the chart's appearance.

ax.set_title(), ax.set_xlabel(), ax.set_ylabel(): These methods are used to set the title, x-axis label, and y-axis label of a plot, respectively.

plt.subplots_adjust(): This function is used to adjust the spacing between subplots in Matplotlib. The parameter passed to this function is the spacing value.

plt.xticks(rotation=90): This function is used to rotate the x-axis tick labels in Matplotlib for better visibility.

Documentation for the file ‘book_data.py’:

sqlite3.connect(database):

Input: database (string) - The name of the SQLite database to connect to.

Output: Returns a connection object representing the database connection.

c.execute(sql):

Input: sql (string) - The SQL query to execute.

Output: None. Executes the given SQL query on the connected database.

c.commit():

Input: None.

Output: None. Commits the current transaction and saves the changes made to the database.

BeautifulSoup(response.content, "html.parser"):

Input: response.content (bytes) - The content of the HTTP response.

Output: Returns a BeautifulSoup object, which represents the parsed HTML content of the response.

find(name, attrs):

Input: name (string) - The name of the HTML tag to find.

attrs (dictionary) - Optional. A dictionary of attribute-value pairs to filter the search.

Output: Returns the first occurrence of the specified HTML tag that matches the given attributes, as a BeautifulSoup object.

find_all(name, attrs):

Input: name (string) - The name of the HTML tag to find.

attrs (dictionary) - Optional. A dictionary of attribute-value pairs to filter the search.

Output: Returns a list of all occurrences of the specified HTML tag that match the given attributes, as a list of BeautifulSoup objects.

get(attr):

Input: attr (string) - The name of the attribute to retrieve.

Output: Returns the value of the specified attribute of the HTML tag, as a string.

text:

Input: None.

Output: Returns the text content of the HTML tag, as a string.

get(href):

Input: href (string) - The name of the attribute to retrieve.

Output: Returns the value of the specified attribute of the HTML tag, as a string.

strip():

Input: None.

Output: Returns the text content of the HTML tag with leading and trailing whitespace removed, as a string.

split(sep):

Input: sep (string) - The separator used to split the string.

Output: Returns a list of substrings obtained by splitting the original string at occurrences of the specified separator.

float(x):

Input: x (string) - The string to convert to a float.

Output: Returns a floating-point number representation of the input string.

list(zip(*iterables)):

Input: *iterables (multiple iterables) - Multiple iterables to be combined into a list of tuples.

Output: Returns a list of tuples, where each tuple contains elements from the input iterables combined element-wise.

c.executemany(sql, seq_of_parameters):

Input: sql (string) - The SQL query to execute.

seq_of_parameters (list of tuples) - A list of tuples containing the parameters to be inserted into the query.

Output: None. Executes the given SQL query with the provided parameters multiple times.

c.executemany(sql, seq_of_parameters):

Input: sql (string) - The SQL query to execute.

seq_of_parameters (list of tuples) - A list

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

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4/8/2023	Getting API with runtime by year	https://imdb-api.com/api/#Search-header	Had an option to search by year unlike other movie API's
4/13/2023	Trying to make each bar in a bar graph a different color	-https://matplotlib.org/stable/gallery/color/named_colors.html -https://stackoverflow.com/questions/42086276/get-default-line-c	I realized by not specifying the color when I plotted each value, it would cycle automatically

		olour-cycle	
4/15/2023	Figuring out similarities between 2 APIs and conceptually connecting them into a visualization		Using percentage differences instead of raw values of movie and song durations
4/18/2023	Putting values of each bar on the graph	https://stackoverflow.com/questions/30228069/how-to-display-the-value-on-horizontal-bars	Found the bar_label method that does this automatically and added it in
4/20/2023	Goodreads web scraping	https://www.goodreads.com/list/show/1.Best_Books_Ever	Section of the Goodreads website used to web scrape data from
4/20/2023	Visualization ideas for web scraped data	https://www.projectpro.io/article/data-visualization-projects-ideas/471	Heatmap, bar graph and scatter plot created