AJ deVaux & Siya Shah

Professor Ericson

SI 206

21 April 2023

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

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.

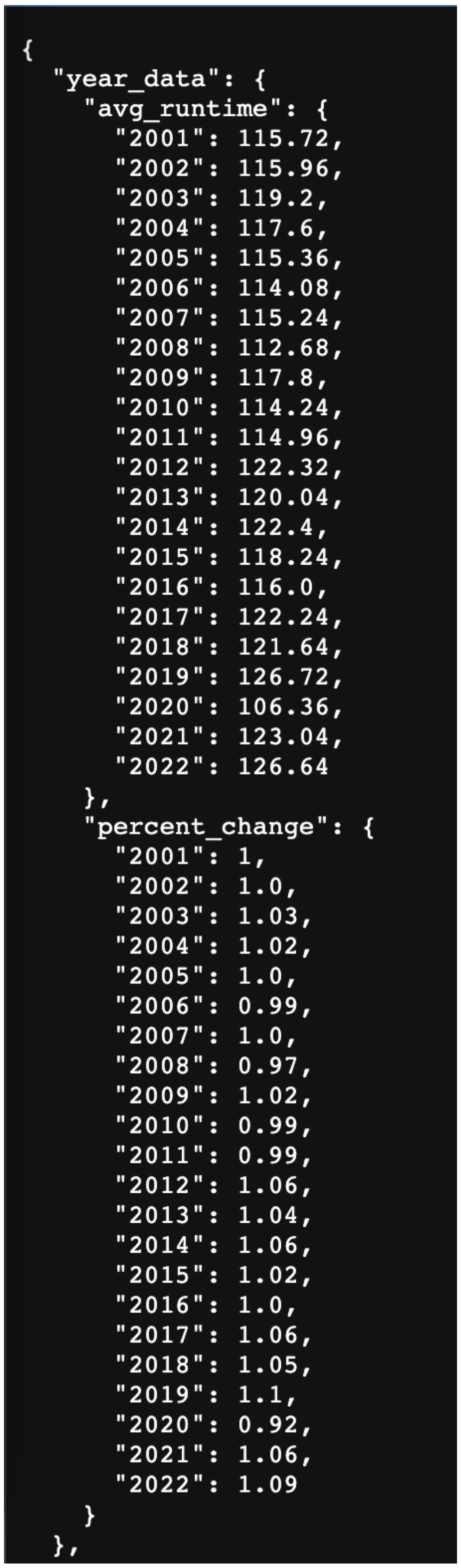
2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

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.

3. The problems that you faced (10 points)

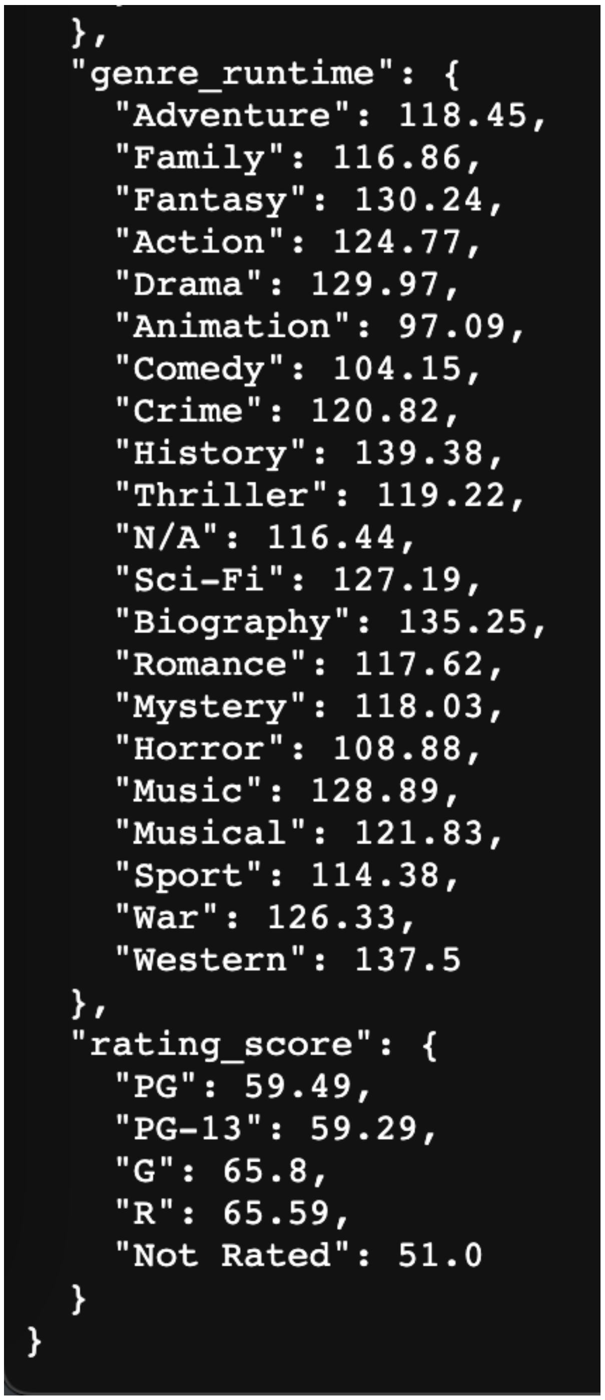
A major problem we faced with collecting data on movies was finding an API that let us search for movies by year. We tried many API's, including the OMDb API and track API, but none let me conduct this search. Thankfully, the IMDb-API offered this functionality through its advanced search, which also 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.

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



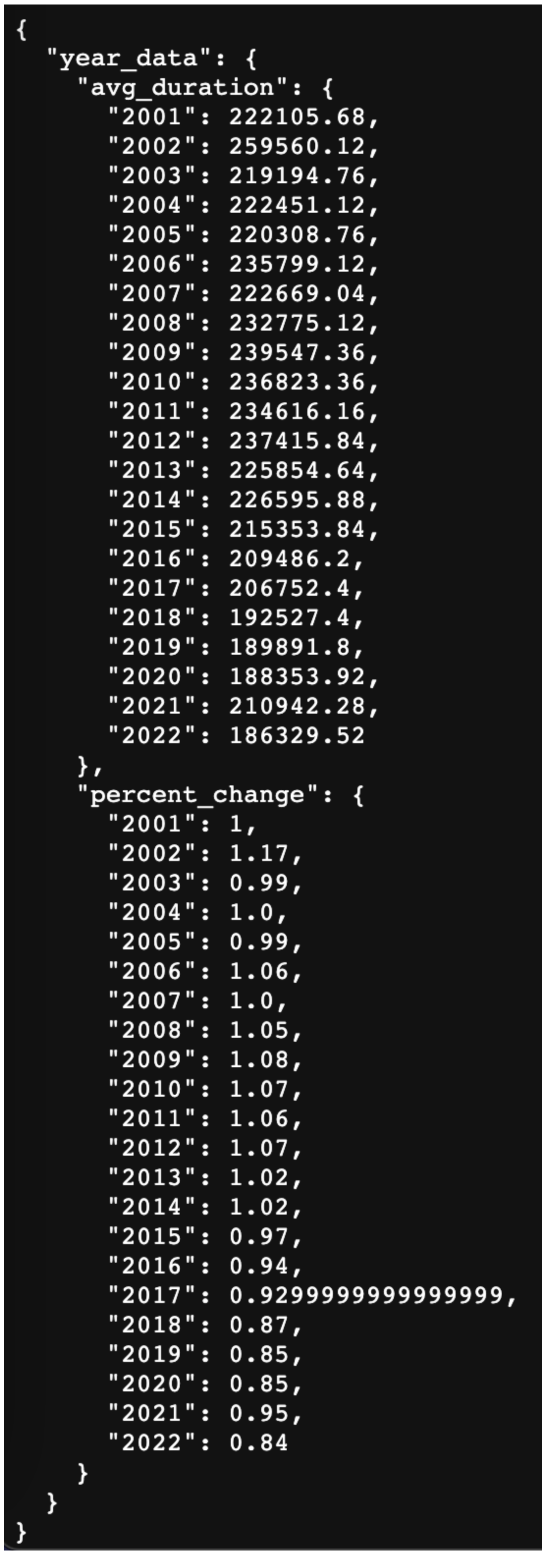
```
{
  "year_data": {
    "avg_runtime": {
      "2001": 115.72,
      "2002": 115.96,
      "2003": 119.2,
      "2004": 117.6,
      "2005": 115.36,
      "2006": 114.08,
      "2007": 115.24,
      "2008": 112.68,
      "2009": 117.8,
      "2010": 114.24,
      "2011": 114.96,
      "2012": 122.32,
      "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
  }
}
```

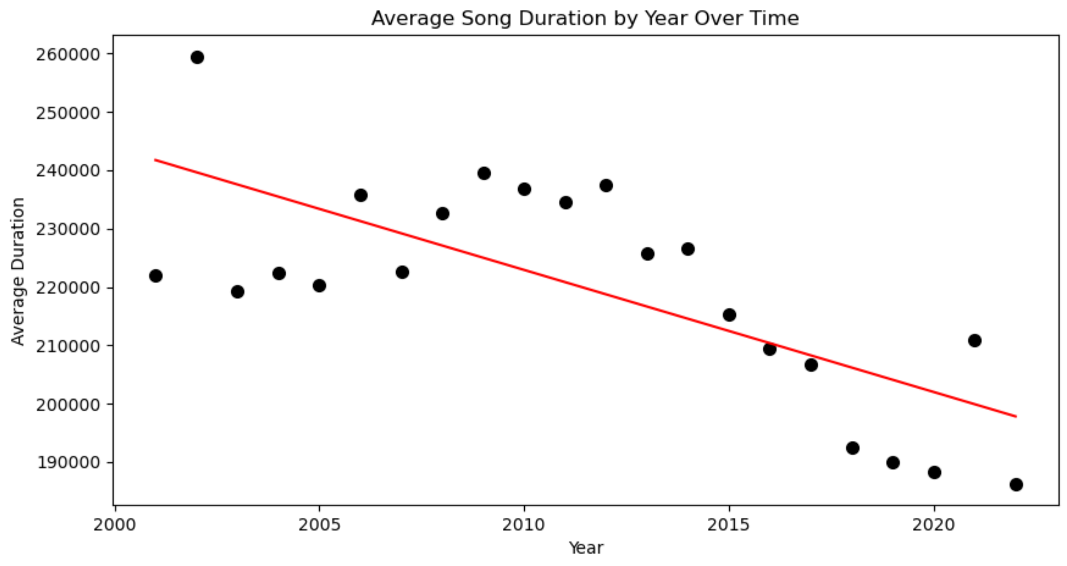
Movies.json

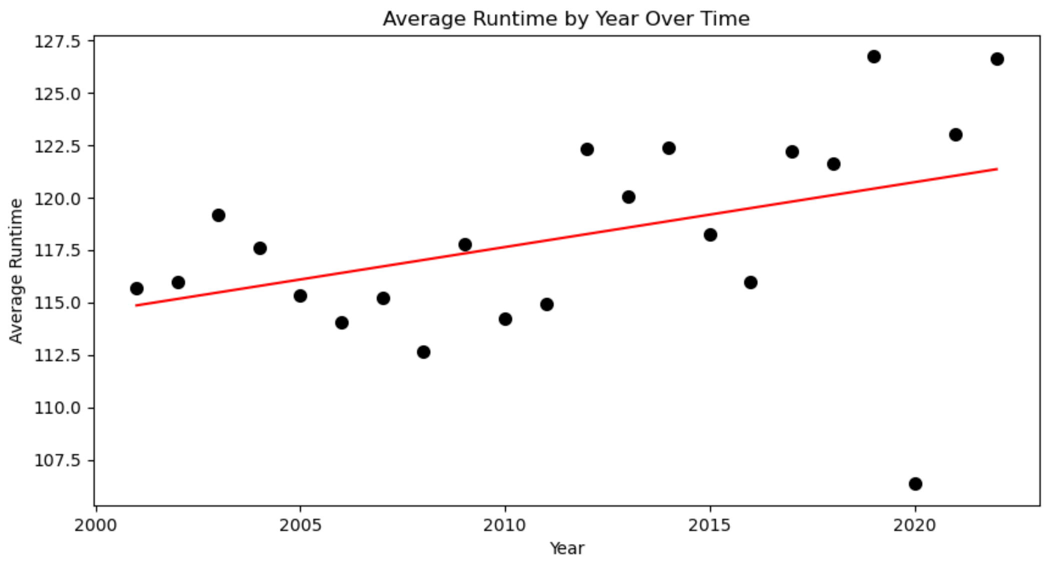


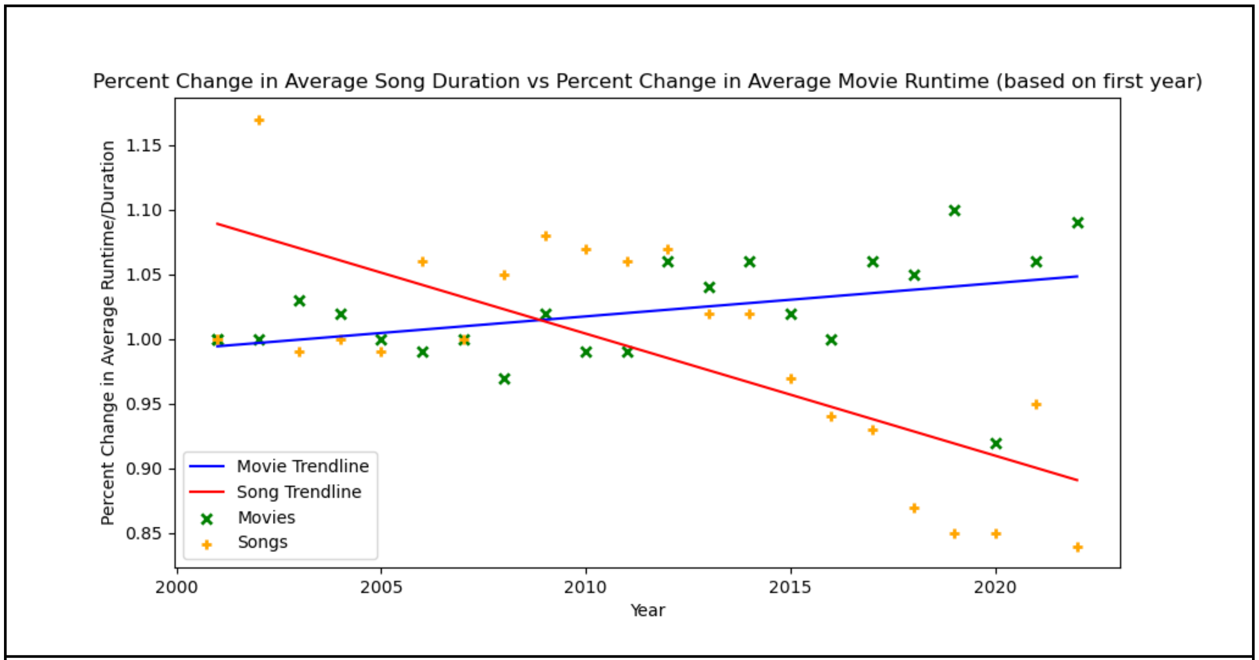
```json
{
  "year_data": {
    "avg_duration": {
      "2001": 222105.68,
      "2002": 259560.12,
      "2003": 219194.76,
      "2004": 222451.12,
      "2005": 220308.76,
      "2006": 235799.12,
      "2007": 222669.04,
      "2008": 232775.12,
      "2009": 239547.36,
      "2010": 236823.36,
      "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
    }
  }
}
```

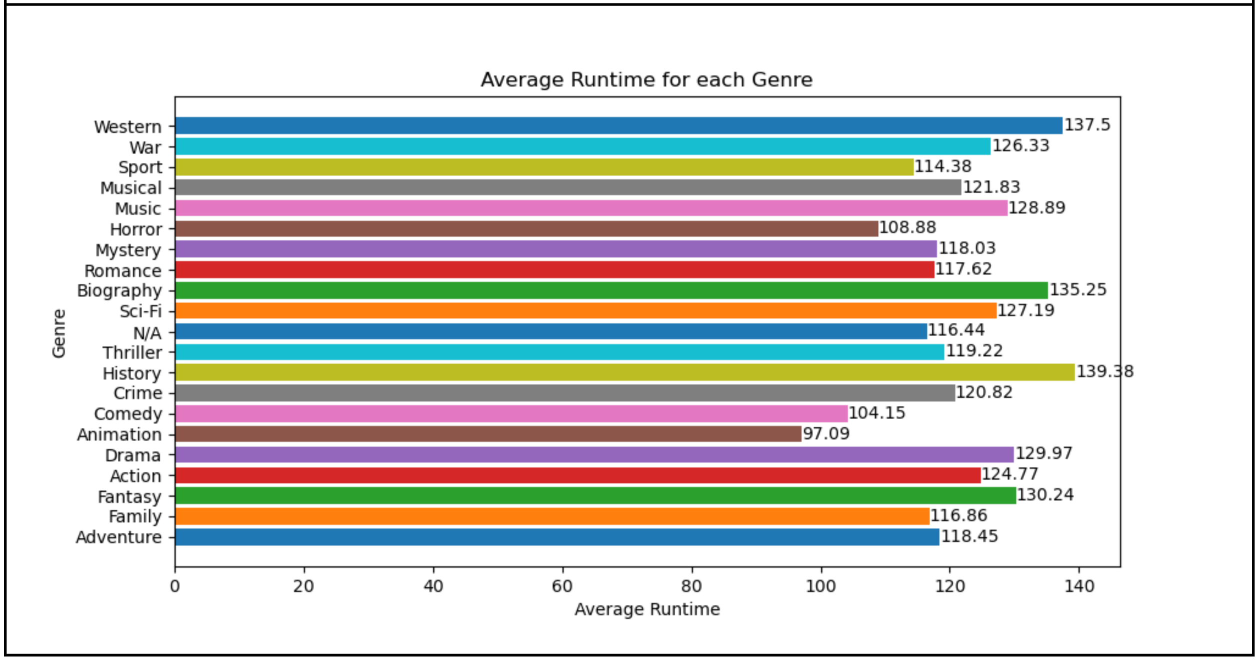
Songs.json

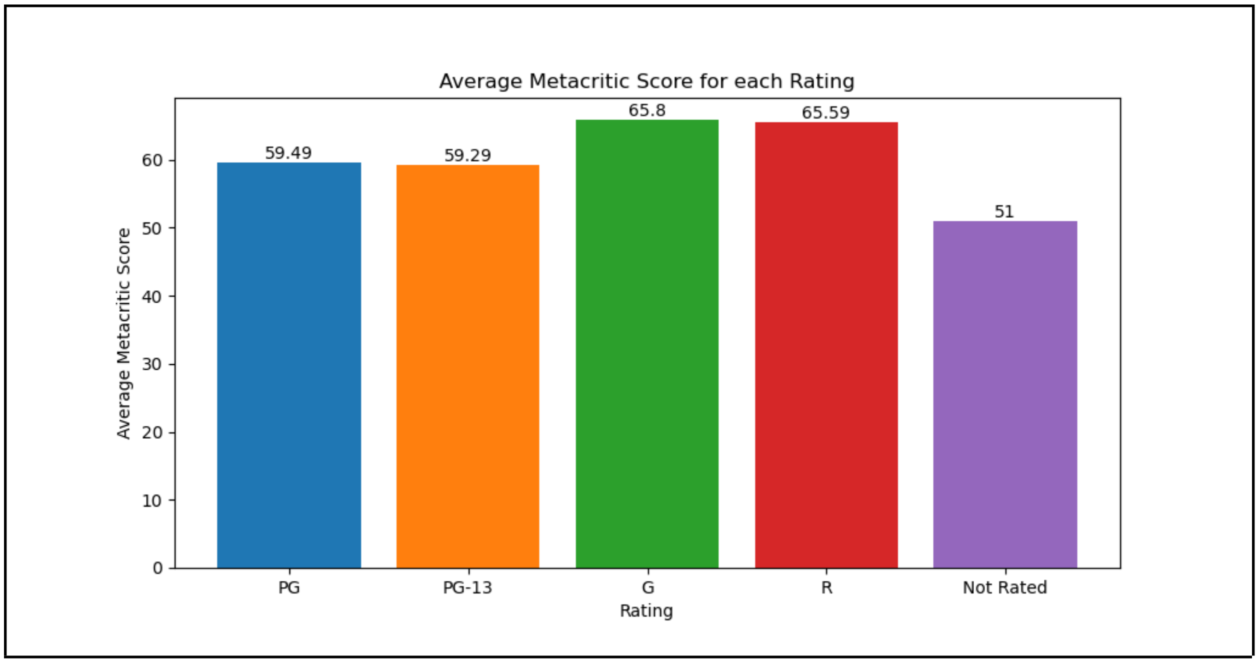
5. The visualization that you created (i.e. screen shot or image file) (10 points)






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


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



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.

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.

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