1. **Introduction :**

IMDb is the world’s most popular and authoritative source for movie, TV, and celebrity content. IMDb users often look at ratings to get an idea of how good movies are, so that they can decide what movies to watch or which ones to prioritize. However, movies that are not yet released don’t have ratings, and even the ones with few votes often change as more users vote. Therefore, I wrote code to predict IMDb ratings of new movies based on various features, such as budget, actors, directors, writers, release year, genres, and plot. While others have used linear regressions to predict ratings of movies in general, those predictions rely on features like movie earnings or number of votes, which would not be available for new movies. I instead combined cosine similarities and normalized Euclidean distances with a modified kNN algorithm, which still produced mostly very accurate predictions. This will provide a way to obtain an estimated rating that’s not yet provided by IMDB

1. **Data Collection :**

I coded all of my work, including the data collection, in Python in Jupyter Notebooks. It probably would’ve been simpler to scrape data from IMDb’s website, but because I wasn’t sure if that was allowed by IMDb, I collected the data from 4 sources:

1.IMDb’s datasets

2.RapidAPI’s Movie Database IMDb Alternative

3.TMDb’s (The Movie Database) API

4.Macrotrends’ data table.



**After exploring the data, I ultimately used the data from 4 of these TSV files:**

1.Regions shown in (called region) came from title.akas.tsv.

2.IMDb ratings (called averageRating) and IMDb # of votes (called numVotes) came from title.ratings.tsv.

3.Directors and writers came from title.crew.tsv.

4.Runtimes (called runtimeMinutes) and media types (called titleType) came from this by title.basics.tsv.

1. **Data Processing :**

Merging, Cleaning, & Filtering

After collecting the data, I first merged the data from IMDb’s datasets and RapidAPI. I renamed the column names to match, so that I could then merge most of the DataFrames based on title IDs.

After merging, the ‘akas’ and ‘principals’ DataFrames had multiple rows for each title ID, which resulted in an enormous CSV file of 12.81 GB. Therefore, I removed them, such that my merged DataFrame had a size of 237359 x 44, and a CSV file size 153.6 MB.

I continued processing the data in a DataFrame called df, which was the same as the merged data.

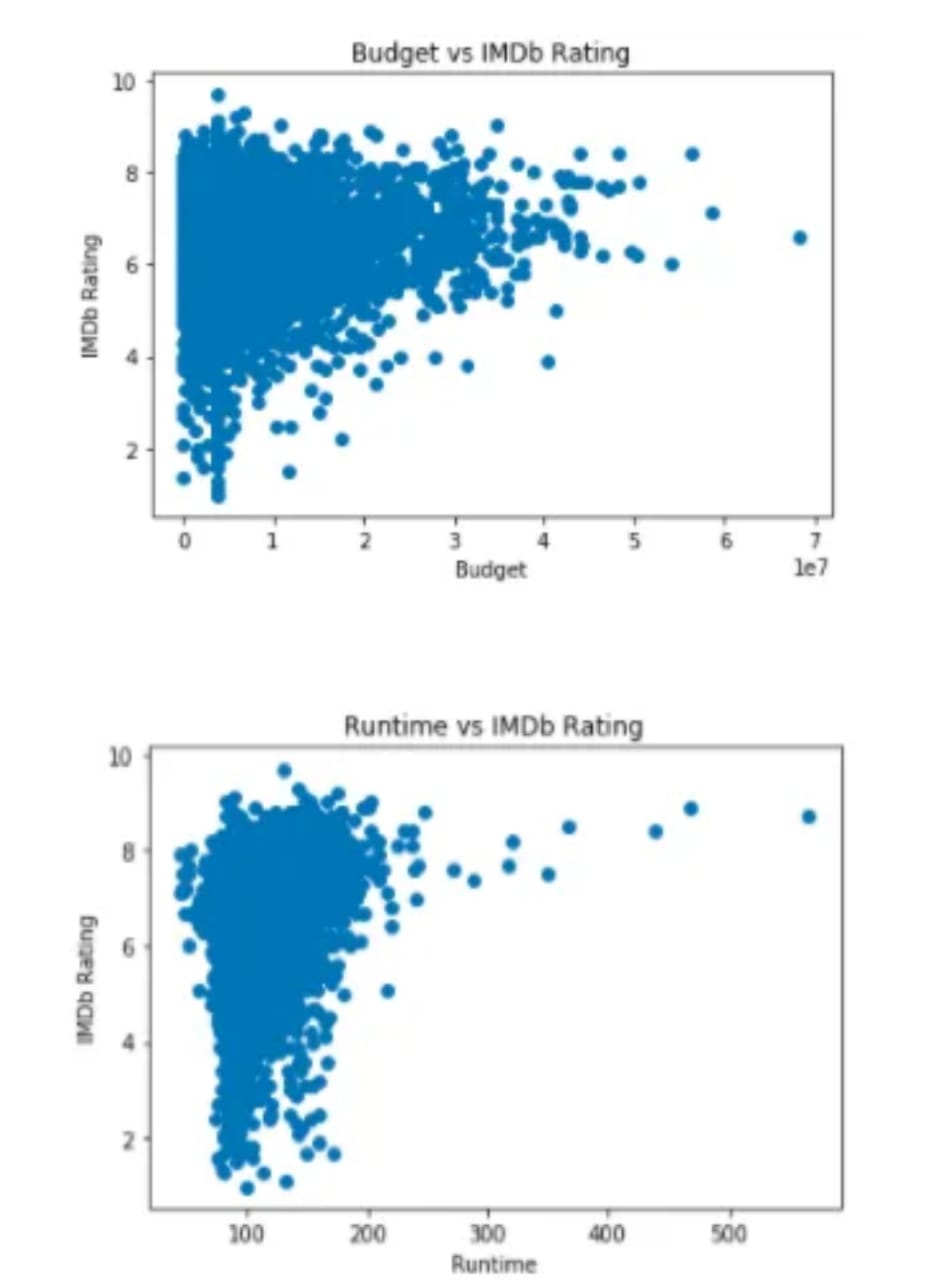
The missing data was provided as newline characters, so I converted those into null values, and the numeric IMDb data was provided as strings, so I converted them to integers.

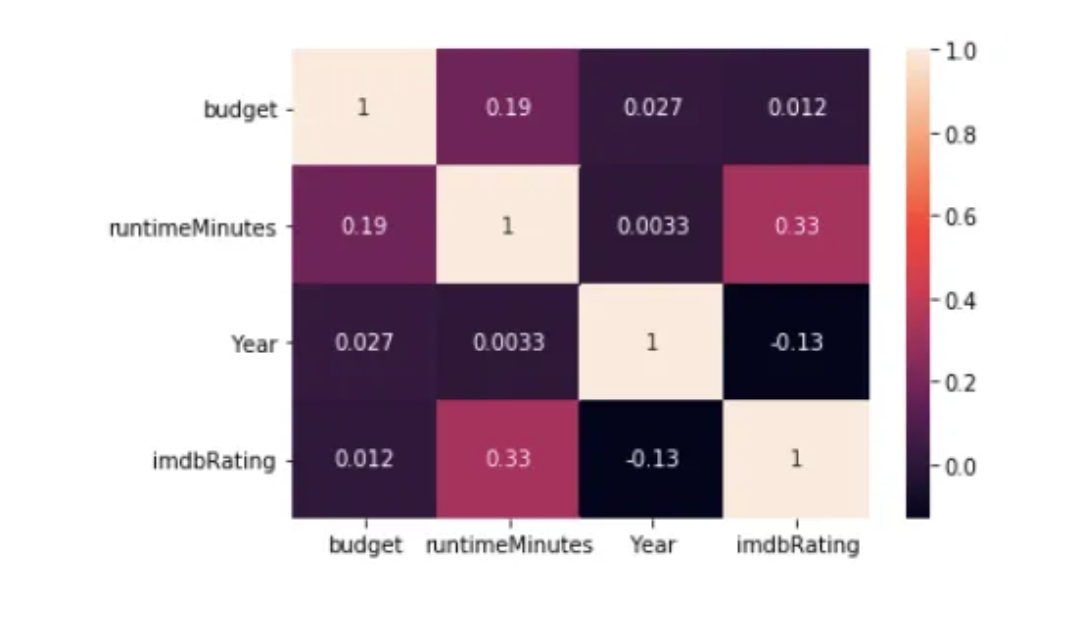
1. **Initial Data Visualizations :**

To help understand the data, in addition to looking through the data and using a lot of .groupby() functions that has too much data to show, I created scatter plots and a heatmap to compare each of my 3 numeric features to IMDb ratings, and I created data tables to explore the categorical features. They ultimately showed that I needed to analyze combinations of features because no single feature can accurately predict ratings of new movies, and they also showed that there were numerous combinations of features.

**Scatter Plots & Heatmap :**

In the below scatter plots, higher budgets and runtimes tend to have higher IMDb ratings, but for the vast majority of budgets and runtimes, the IMDb ratings are spread from about 1 to 9, so they can’t be predicted using either of these features by themselves.





1. **Additional Data Processing :**

I performed additional data processing in order to generate my predicted ratings. I used TF-ID and one-hot encoding on the categorical features to calculate cosine similarities. Then, I calculated normalized Euclidean distances for the numeric features.

TF-IDF

I used TF-IDF (term frequency-inverse document frequency) to reflect how important each word of each movie plot is. TF is the number of times the word appears in the plot. IDF is the logarithm of the total number of plots divided by the number of plots containing the word. TF-IDF is TF multiplied by IDF. I used sklearn’s TfidfVectorizer() function to create the TF-IDF vectors for the movie plots.

1. **Predicting IMDb Ratings :**

Using the cosine similarities and Euclidean distances from the previous (Additional Data Processing) section, I calculated the total similarity score as the (cosine similarities + (1 — Euclidean distances)) with some weights applied. Then, I predicted ratings using the mean IMDb ratings for the k Nearest Neighbors (kNN), which are the movies with the highest similarity scores. Calculating similarities between movies incorporates what I learned in the IS688 course, and because of the sufficiently diverse and high number of movies to compare in the data, it successfully circumnavigated needing to estimate the effects that numerous combinations of features have on IMDb ratings.

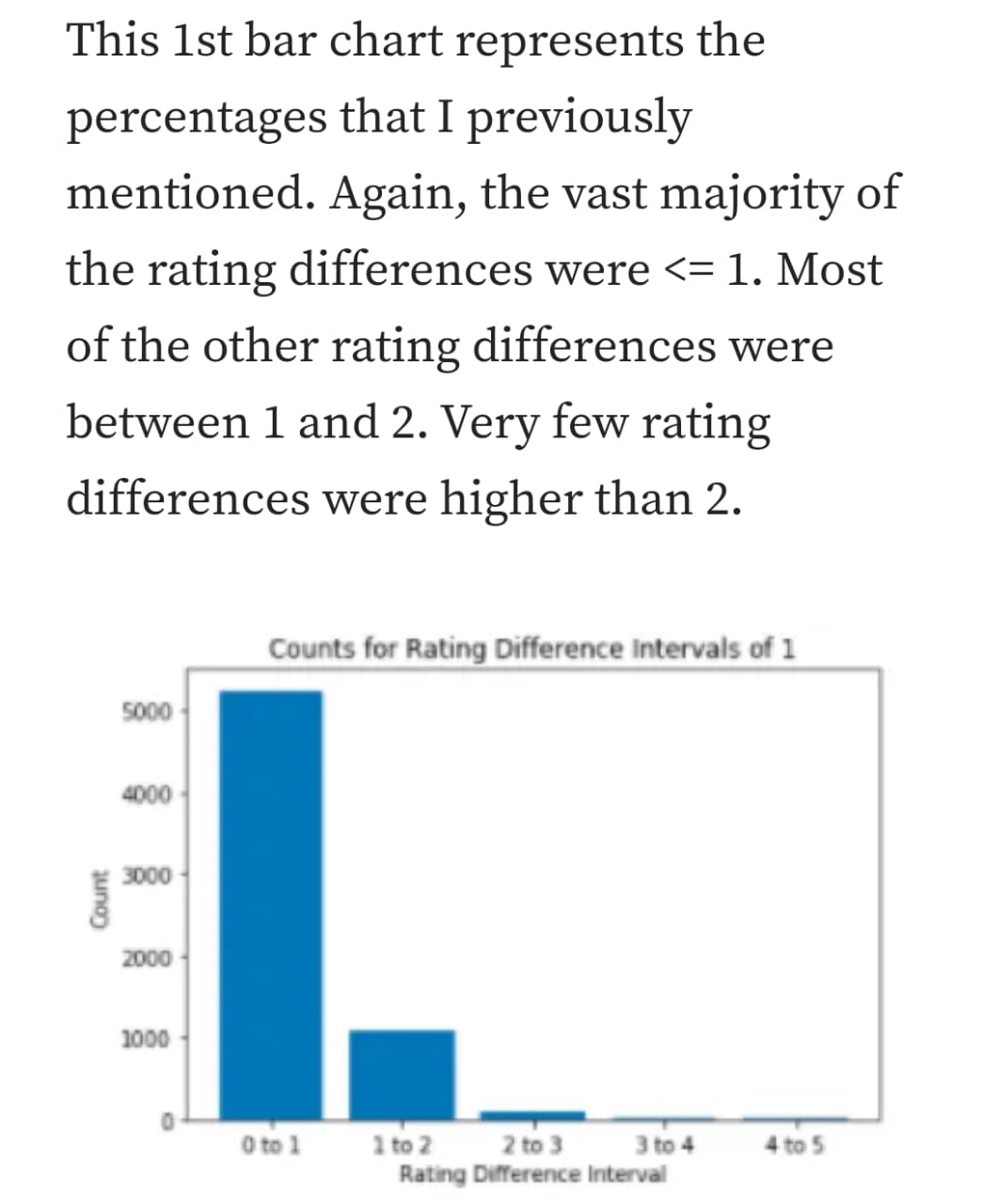
1. **Accuracy :**

I used my predict\_rating() function to create predicted ratings (with the column name “Predicted Rating”) for every movie in df. I calculated “rating difference” (with the column name “Difference”) as the absolute value of (the IMDb rating minus the predicted rating).

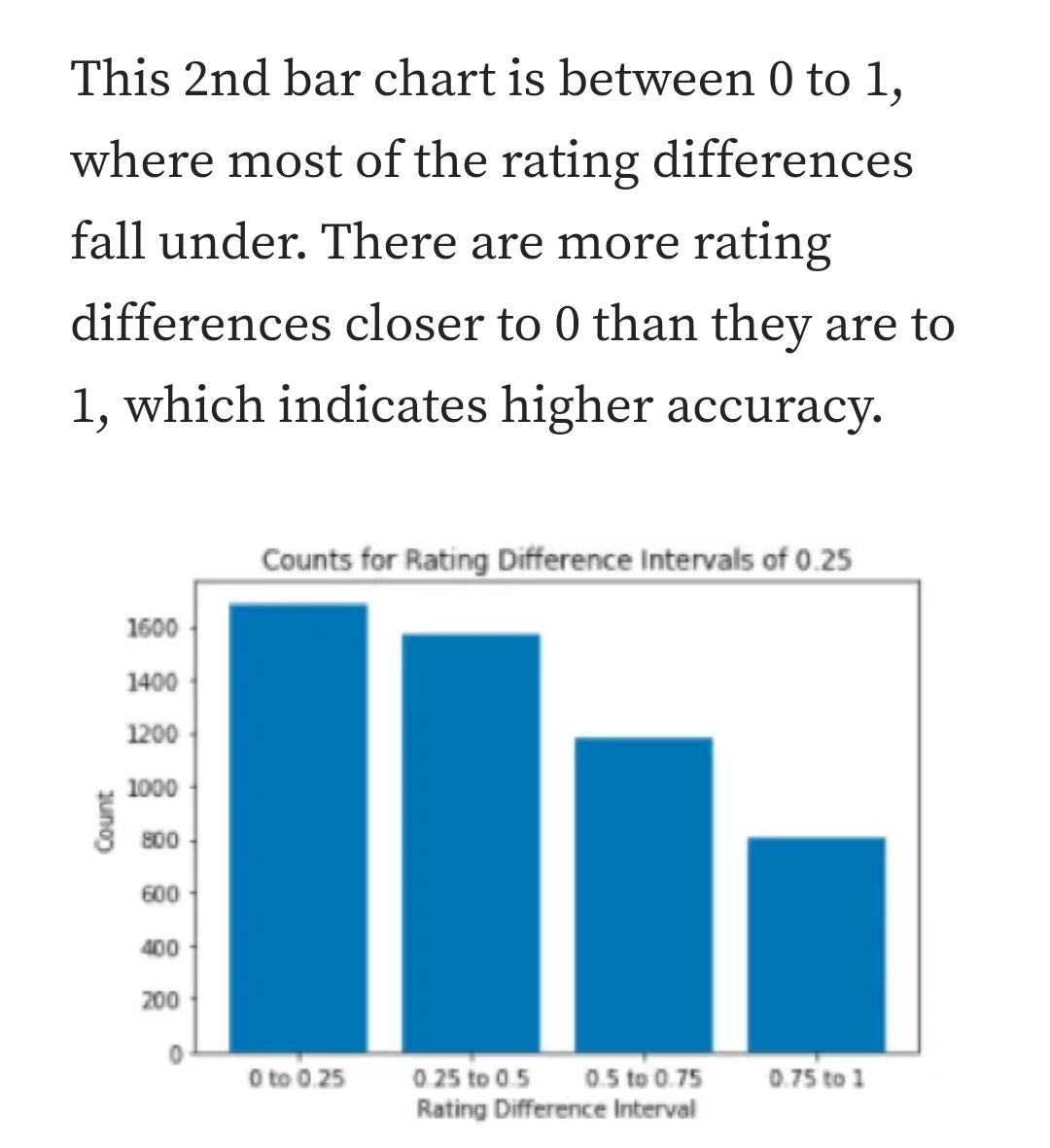
The rating differences were overall very low, which indicates good accuracy. The mean rating difference was 0.60, and the standard deviation of rating differences was 0.55.

The vast majority of the rating differences were <= 1. Most of the other rating differences were between 1 and 2. Very few rating differences were higher than 2.

**This 1st bar chart represents the percentages that I previously mentioned. Again, the vast majority of the rating differences were <= 1. Most of the other rating differences were between 1 and 2. Very few rating differences were higher than 2.**

****

**This 2nd bar chart is between 0 to 1, where most of the rating differences fall under. There are more rating differences closer to 0 than they are to 1, which indicates higher accuracy.**



1. **Conclusion :**

Accurately predicting IMDb ratings of new movies is challenging. I needed to analyze combinations of features because no single feature can accurately predict ratings of new movies. Yet, there were numerous combinations of features that can affect IMDb ratings differently than the individual values would indicate. I performed a lot of data processing and ultimately combined weighted cosine similarities and normalized Euclidean distances with a modified kNN algorithm. With this technique, my predicted ratings of most movies are within a difference of 1 from their actual IMDb rating, with very few exceptions.

The main limitation is that the predicted ratings are dependent on having data for movies that have similar combinations of features. This is usually but not always the case, as evidenced by the fact that because there are much fewer low rated movies in my filtered data, the predicted ratings are more inaccurate for IMDb ratings below 4. In the future, I potentially could include similarities based on movie production companies (e.g. Marvel Studios, Paramount Pictures, Walt Disney Pictures), which may improve the accuracy of the predicted ratings. This would likely require scraping from IMDb’s website because only the 1st (not necessarily most important) production company was provided by the IMDb datasets, and many weren’t correct from TMDb’s API.