Project Overview:

For my project I harvested text data from IMDb, however, my methods diverged slightly from those outlined in part one of the project's description. Although I did call the imdbpie API for a particular function, I also used the BeautifulSoup module to scrape IMDb for information pertaining to their top 250 ranked movies of all time. I was interested in learning more about web scraping and also wanted to determine the most successful actors and directors based on the frequency of appearances in the top 250 films. Thus, I built my own mini API and used several analysis techniques including word frequency, sentiment analysis, and visualization.

Implementation:

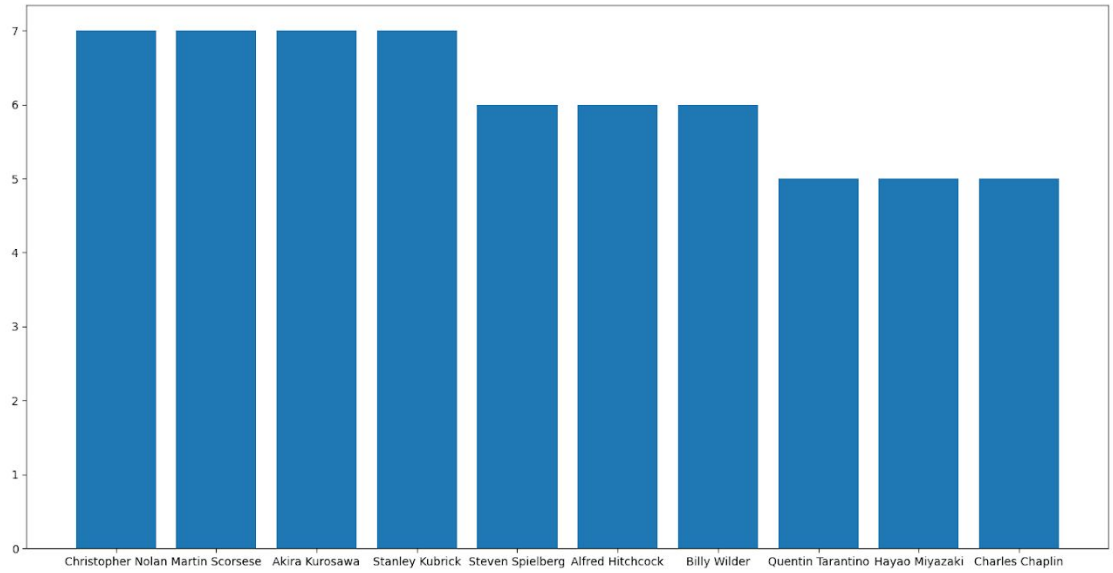
I began by building my own mini API (appropriately called imdbmini) to extract text from the web pages containing IMDb's top 250 movies of all time. I then planned to reference the imdbmini API in future functions that I would use for analysis. To build the initial API, I used the BeautifulSoup and requests libraries to scrape data from IMDb directly. I then used this data to create several ordered lists, those of which included the titles of these movies, their directors, their runtimes, and the actors starring in these films. Initially, I planned to store all of this information in a nested dictionary where the movie titles would be the keys and the remaining topics and their pairs would be the subsequent keys and values. However, most films had more than one star and I found it difficult to assign each star to his or her respective film without accidentally assigning certain films incomplete or incorrect values. Thus, I made the conscious choice to use ordered lists for each topic to ensure that the information I used for analysis would be complete.

After completing imdbmini, I began the analysis portion of the assignment and built several functions to analyze the text I had gathered. I created a bar chart function that transformed data from a list into a dictionary and plotted each key value in descending order of frequency. imdbmini calls are a valid parameter for this function, thus meaning one could analyze the top 10 directors by calling the previously constructed API. The same goes for the hist function, which was primarily built as a means of visualizing the runtimes of the most successful movies. For the average_reviewscore function, I called the imdbpie API and accepted a movie title from the user as a string. I then iterated through each user review left for the specified title and used sentiment analysis to print the average compound score of all reviews left for the particular title. The top_titles function also accepted calls to my API and printed the titles of the top n number of movies as per IMDb's website.

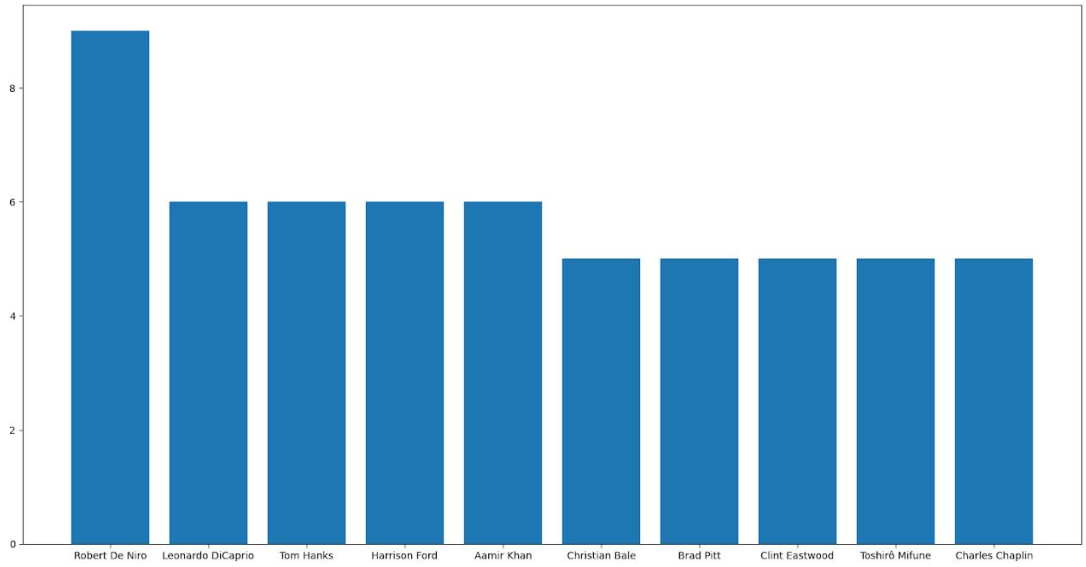
Results:

All of these functions returned fairly interesting results. First, I found that four individuals directed seven films each that were included in IMDb's top 250 films of all time. These individuals include Christopher Nolan, Martin Scorsese, Akira Kurosawa, and Stanley Kubrick. Based on these results, one may conclude that these four have garnered the most success or

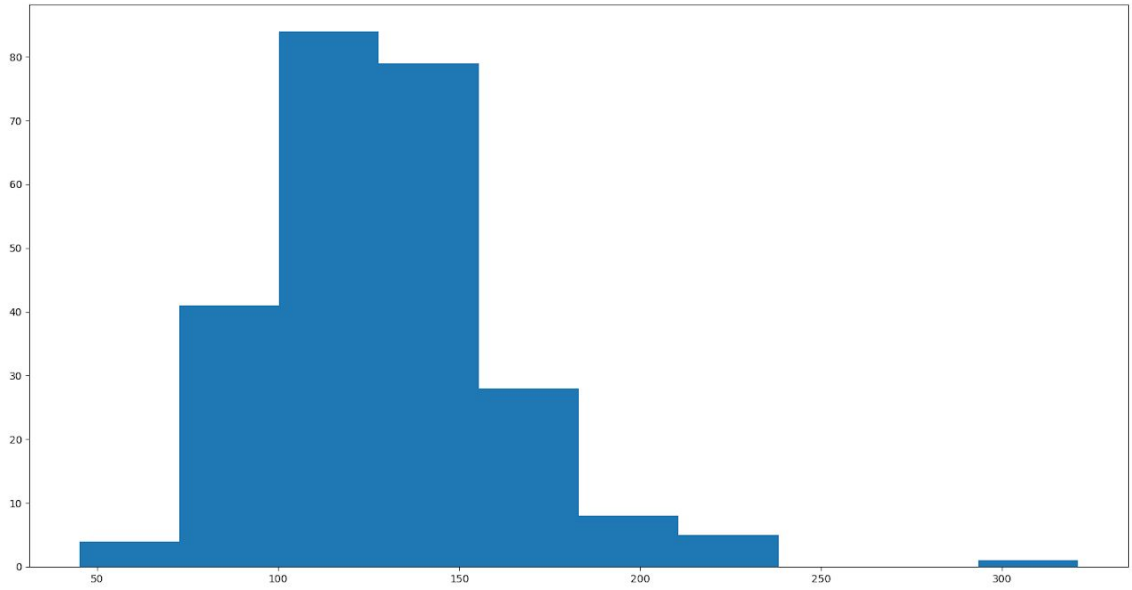
exhibited the most consistency throughout their careers. A bar chart depicting the top 10 directors by frequency of appearance is shown below:



Additionally, I found that Robert De Niro appeared in the most movies that fell within the top 250. He appeared in 9 of these films, which is 3 more than the next tier of performers (those of which include Leonardo DiCaprio and Tom Hanks). A bar chart showing the top 10 actors by frequency of appearance is shown below:



The histogram I created showed that the majority of top movies fell somewhere between the range of ~100 and ~120 minutes. Interestingly, there was also an outlier that lasted well over four hours in length!



As I mentioned previously, I created a function to display the top n movies as per IMDb's rankings. For simplicity purposes, I only decided to print the top 10 films. They are as follows:
1: The Shawshank Redemption
2: The Godfather
3: The Dark Knight
4: The Godfather: Part II
5: The Lord of the Rings: The Return of the King
6: Pulp Fiction
7: Schindler's List
8: 12 Angry Men
9: Inception
10: Fight Club

Finally, as a means of demonstrating the average_reviewscore function, I ran a sentiment analysis on the reviews of two films. I made sure to choose a film I knew was universally celebrated ('The Shawshank Redemption') and a film I knew was universally critiqued (Cats) as a means of proving the success of my function. I printed the average compound score of all reviews for the two movies. An output closer to -1 indicates extremely negative sentiments whereas an output close to 1 indicates extremely positive sentiments. Unsurprisingly, reviews

for the ‘The Shawshank Redemption’ had an average compound score of 0.906656 whereas reviews for Cats only had an average compound score of -0.134364.

Reflection:
Overall, I believe this project went pretty well as I was able to answer my two initial questions as well as provide further insight into certain topics I did not even consider when the project began. Although I was able to get everything to work, I think it would be beneficial to bolster my understanding of dictionaries. If the API I created were to be used in a real-world setting, I feel it would be more useful if all the information was stored in a nested dictionary rather than a collection of ordered lists. Given the circumstances, I feel the project was appropriately scoped and I was able to accomplish a lot in a relatively short period of time. I was able to learn a lot about scraping web pages and using CSS selectors to extract information I desire, which I feel may be useful for programs I design in the future. I definitely wish I knew more about regular expressions prior to tackling this project, however, as this simple function took me a while to understand and was imperative to the success of my project. All in all, though, I am fairly happy with the results.