SI 206 Final Report

Repository: https://github.com/wiggikay/Theater

--- Goals & Achieved Goals ---

Originally our project was going to be about calculating the country with the best cuisine for specific dietary needs. We were going to do this by scraping the TasteAtlas website (https://www.tasteatlas.com/most-popular-food-in-jamaica) for popular dishes and then use the spoonacular API to find the ingredients of the dish and then use the Edaman API to calculate the nutritional value of the dishes. However, we were unable to do this as the website that we were going to scrape data from used iframe in a way that kept us from scraping it.

Our new goal is to calculate the average ratings of the top 10 best films for each actor in IMDB's most popular actors and determine whether there was a correlation between the popularity rank and average film rating. We also wanted to see if the top best 50 best actors were really in better films than the bottom 50. Both these findings are done to see whether the popularity ranks are based on their films or if there are other factors like real-life personability or politics that contribute to popularity. We also wanted to find out the genre distribution of the best films of the top 100 Actors to determine what types of films most popular actors plan to do.

We planned on scraping IMDB for the list of the most popular actors in the world (https://www.imdb.com/list/ls022928819/) and then using The Movie Database (https://developers.themoviedb.org/3/getting-started/introduction) API to find their top 10 highest rated films. We wanted to make a histogram and a scatter plot to show the distribution of ratings and a pie chart to show the distribution of genres.

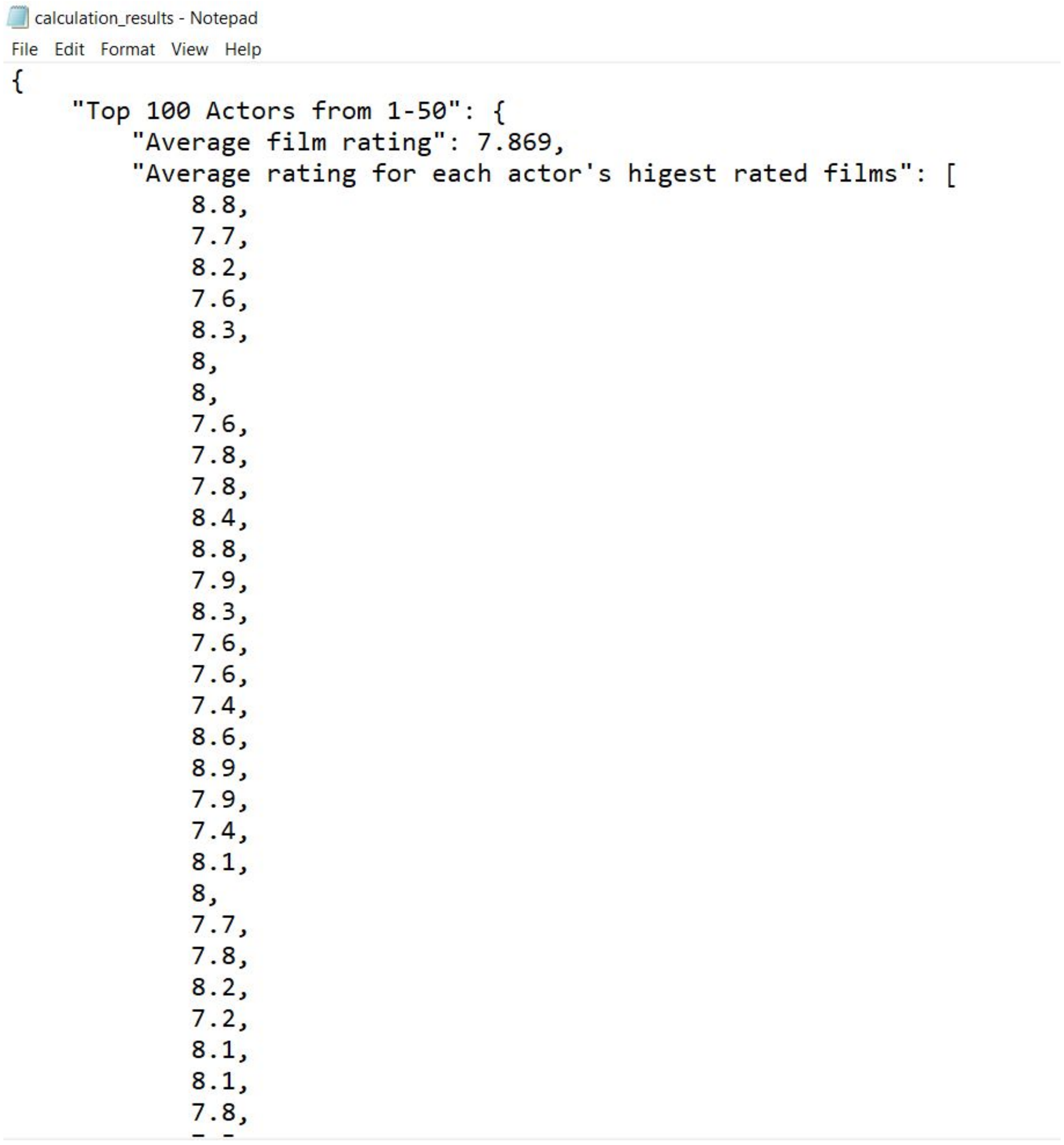
Despite the rough start with our original plan, we were able to meet all the goals of our current plan.

--- Problems we Faced ---

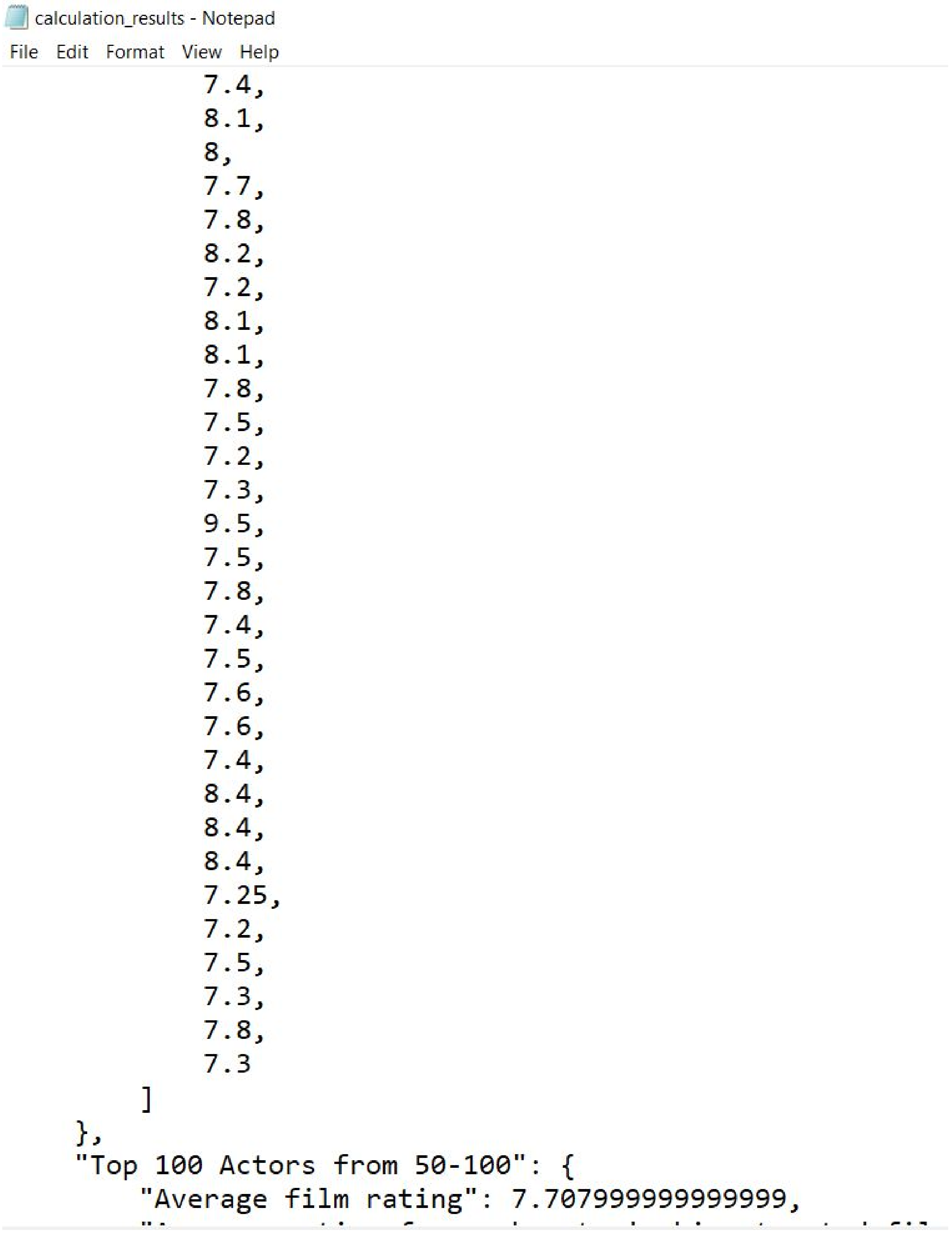
As I mentioned before we were unable to go through with our original plan because of issues with using Beautiful Soup on an iframe site. We also had a bit of trouble with modeling the database in the best way because we weren't always sure which attributes we would need to make the calculations efficiently and answer our questions. Sometimes we needed to go into the database and add attributes to a table or remake the table to remove certain attributes. Through trial and error, we figured out exactly which attributes we needed in which tables.

--- Calculations From The Data in the Database ---

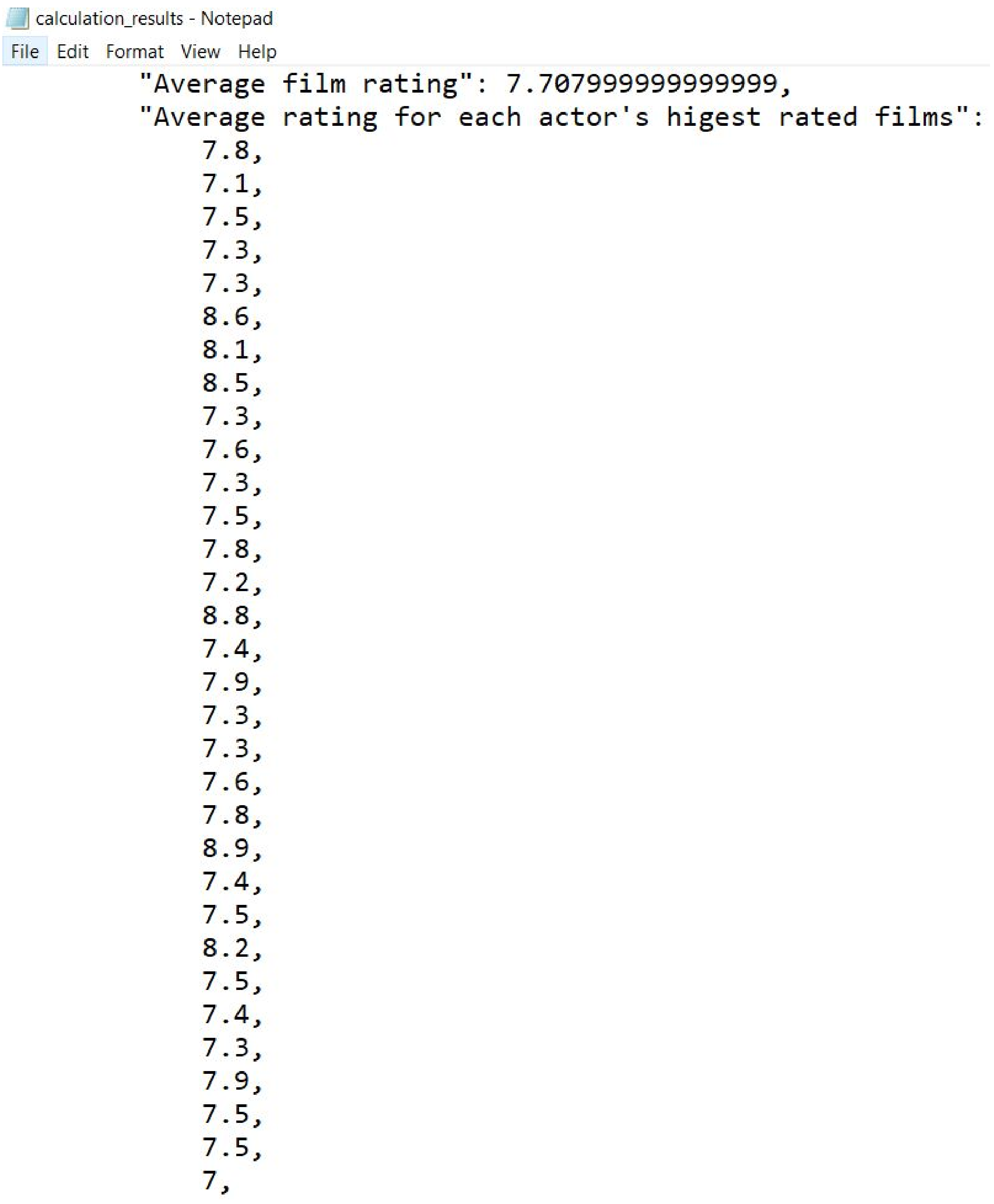
The name of the file containing the calculations from the data in the database for finding out the correlation between popularity rank and average film rating is: calculation_results.txt



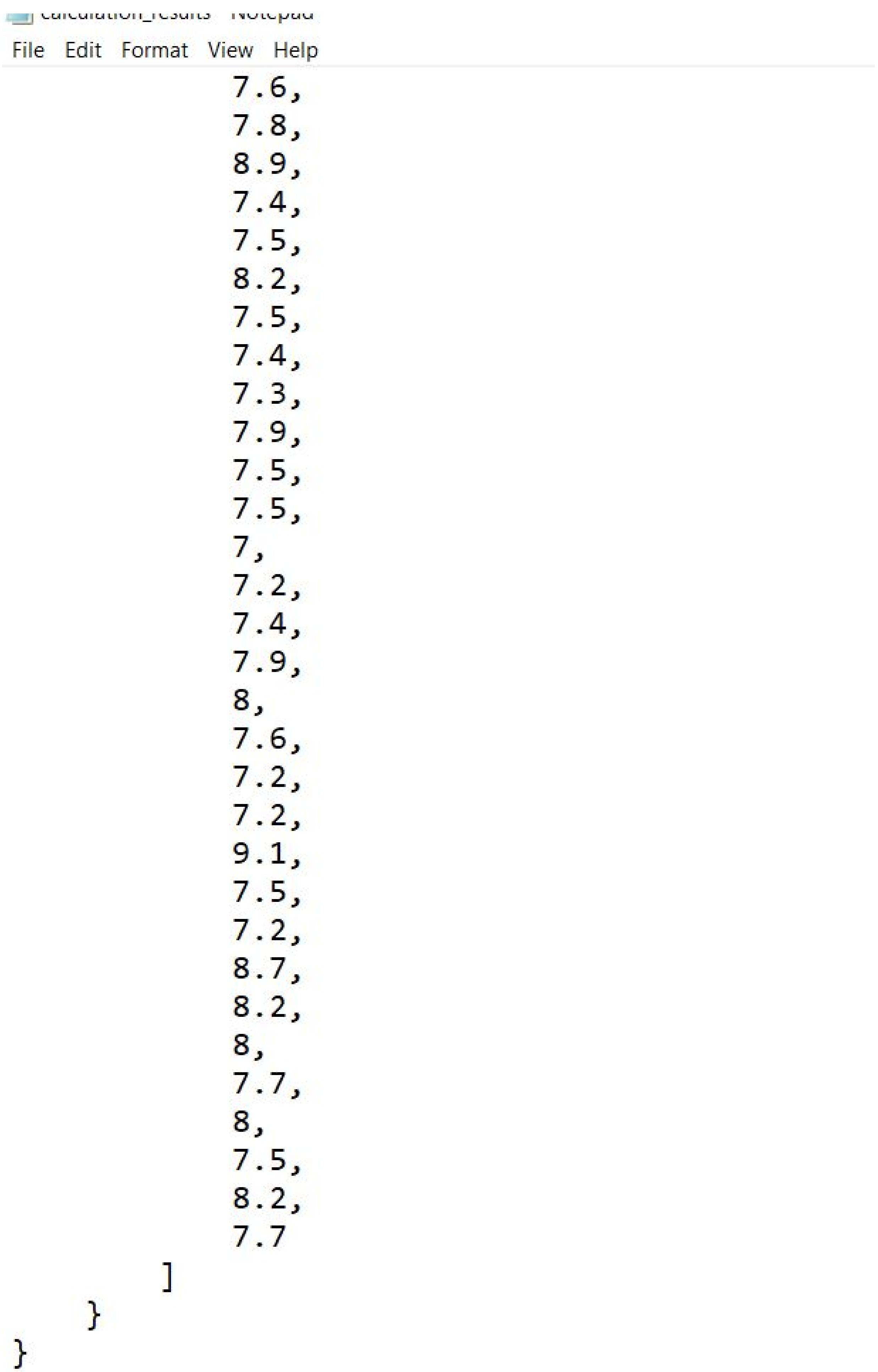
```
{
    "Top 100 Actors from 1-50": {
        "Average film rating": 7.869,
        "Average rating for each actor's higest rated films": [
            8.8,
            7.7,
            8.2,
            7.6,
            8.3,
            8,
            8,
            7.6,
            7.8,
            7.8,
            8.4,
            8.8,
            7.9,
            8.3,
            7.6,
            7.6,
            7.4,
            8.6,
            8.9,
            7.9,
            7.4,
            8.1,
            8,
            7.7,
            7.8,
            8.2,
            7.2,
            8.1,
            8.1,
            7.8,
```



```
            7.4,
            8.1,
            8,
            7.7,
            7.8,
            8.2,
            7.2,
            8.1,
            8.1,
            7.8,
            7.5,
            7.2,
            7.3,
            9.5,
            7.5,
            7.8,
            7.4,
            7.5,
            7.6,
            7.6,
            7.4,
            8.4,
            8.4,
            8.4,
            7.25,
            7.2,
            7.5,
            7.3,
            7.8,
            7.3
        ]
    },
    "Top 100 Actors from 50-100": {
        "Average film rating": 7.707999999999999,
```

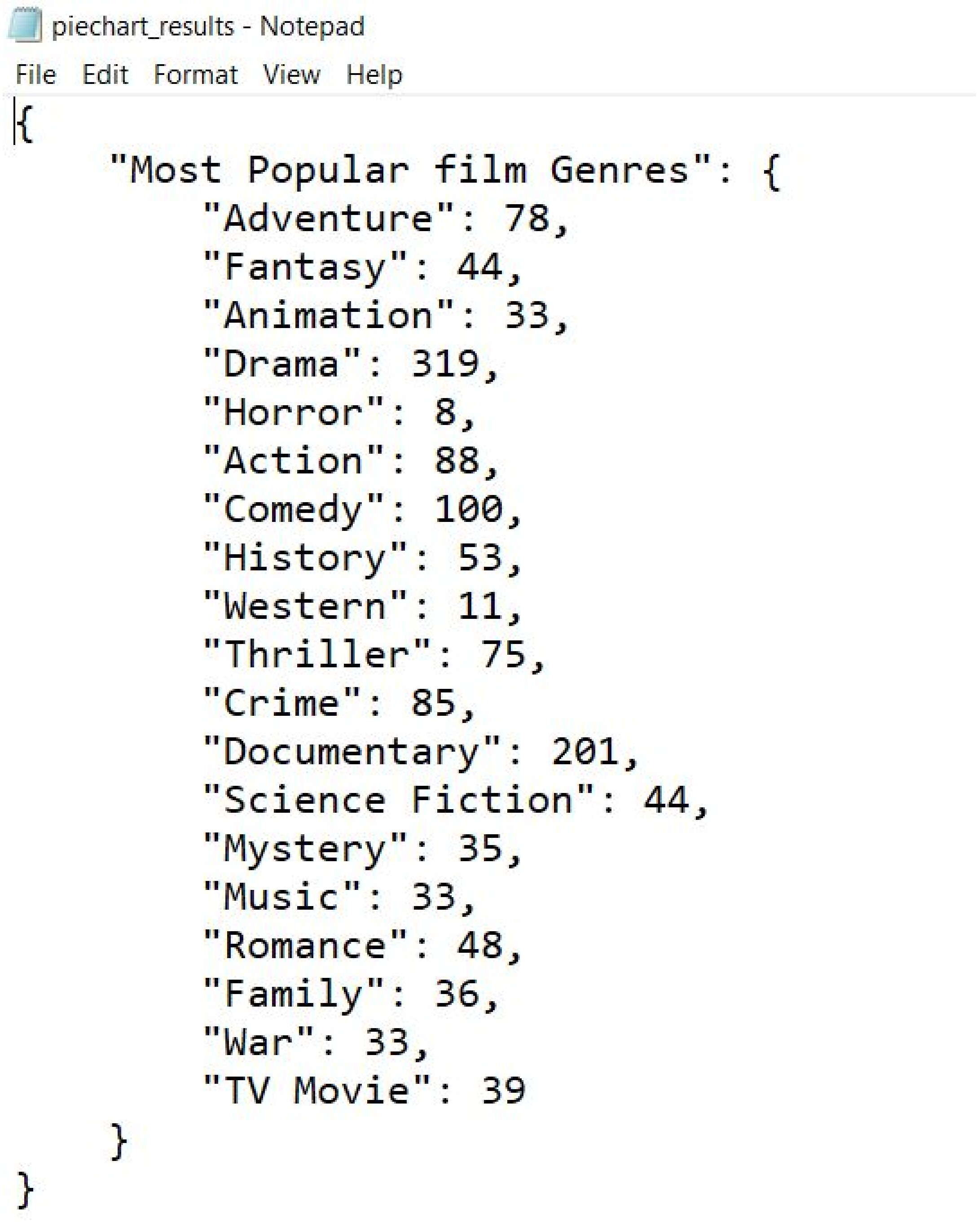


```
        "Average film rating": 7.707999999999999,
        "Average rating for each actor's higest rated films":
            7.8,
            7.1,
            7.5,
            7.3,
            7.3,
            8.6,
            8.1,
            8.5,
            7.3,
            7.6,
            7.3,
            7.5,
            7.8,
            7.2,
            8.8,
            7.4,
            7.9,
            7.3,
            7.3,
            7.6,
            7.8,
            8.9,
            7.4,
            7.5,
            8.2,
            7.5,
            7.4,
            7.3,
            7.9,
            7.5,
            7.5,
            7,
```



```
            7.6,
            7.8,
            8.9,
            7.4,
            7.5,
            8.2,
            7.5,
            7.4,
            7.3,
            7.9,
            7.5,
            7.5,
            7,
            7.2,
            7.4,
            7.9,
            8,
            7.6,
            7.2,
            7.2,
            9.1,
            7.5,
            7.2,
            8.7,
            8.2,
            8,
            7.7,
            8,
            7.5,
            8.2,
            7.7
        ]
    }
}
```

The name of the file containing the calculations from the data in the database for finding out the distribution of genres for best films is: piechart_results.txt

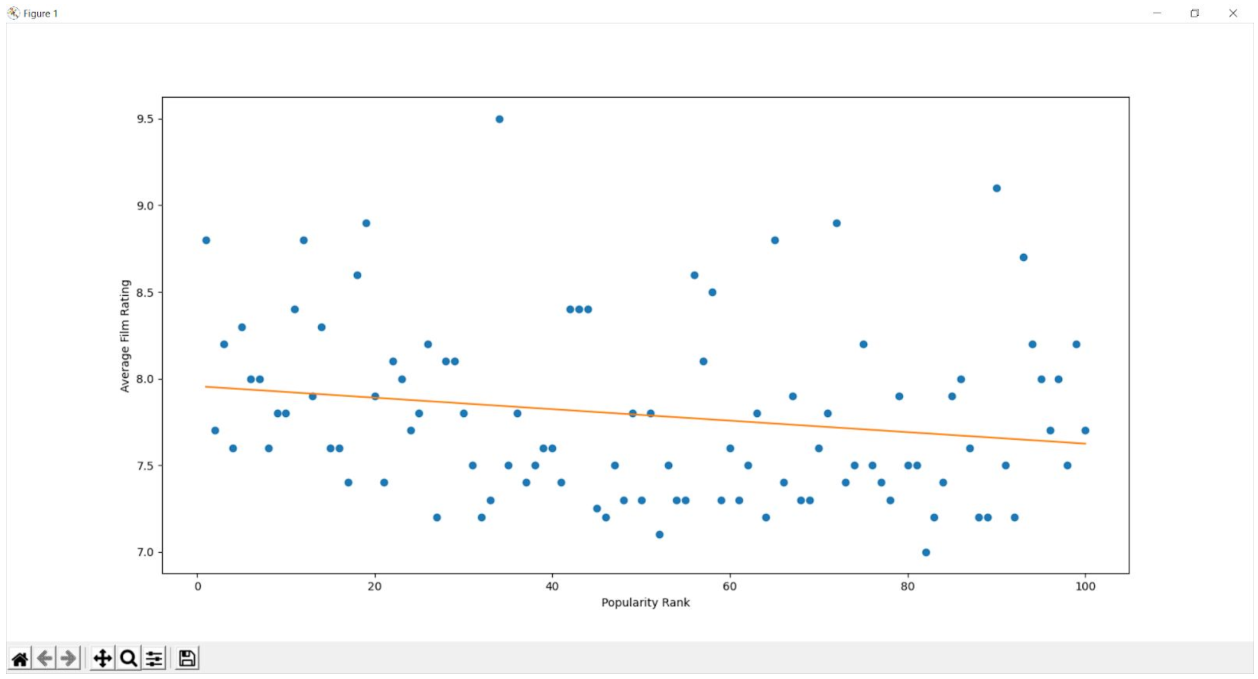


```
{
    "Most Popular film Genres": {
        "Adventure": 78,
        "Fantasy": 44,
        "Animation": 33,
        "Drama": 319,
        "Horror": 8,
        "Action": 88,
        "Comedy": 100,
        "History": 53,
        "Western": 11,
        "Thriller": 75,
        "Crime": 85,
        "Documentary": 201,
        "Science Fiction": 44,
        "Mystery": 35,
        "Music": 33,
        "Romance": 48,
        "Family": 36,
        "War": 33,
        "TV Movie": 39
    }
}
```

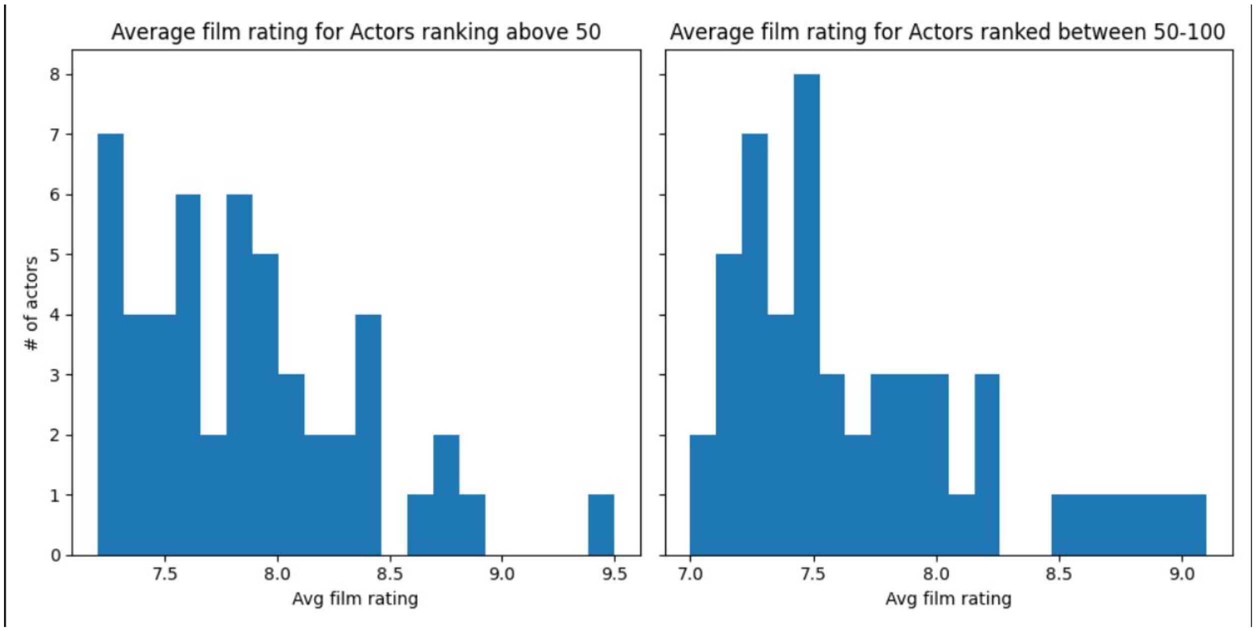
--- The Visualizations ---

We have three visualizations.

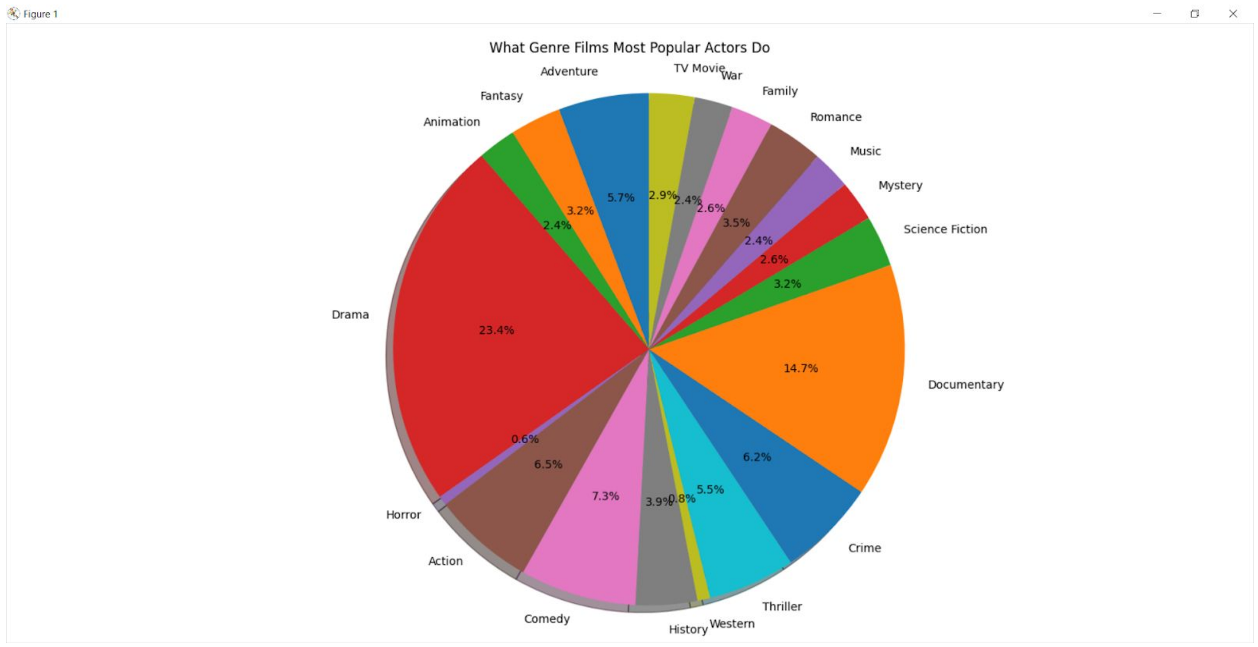
The scatterplot that shows the correlation between popularity rank and average film rating. File name: scatterplot.PNG



The histogram that shows the distribution of the top 50 and bottom 50 of the top 100 actors with relation to their average film rating. File name: histogram.png



The pie chart that shows the distribution of genres for the top 10 films of the most popular actors. File name: piechart.PNG



--- Instructions for running the code ---

Step 1:

Step 1 is to run the part1.py python file to set up a database and create two tables Actors and Actor_Popularity. Data about most popular actors are collected from the 'themoviedb' API and the IMDb website. The two tables are filled with these two different data and completed. One point to note, the Actors_Popularity table has an empty string in the films column. This column will be updated later.

Also to note, after every run, only 25 new rows will be added to the database. Part1. Py needs to be run four times to complete the database and move to the next step.

Step 2:

Step two is to use the Popular_Actors.db Actors table to create a new table called Films that list the top 10 highest rated films from all of the actors and put the ids of those films in the actor_name attribute of the actor. Run populate_films.py to create and populate the Films table with the top 10 films for 25 of the actors listed in the Actors table whose films haven't already been recorded.

Step 3:

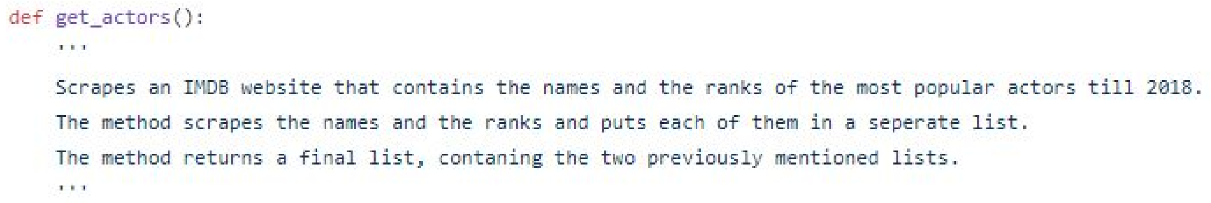
--Visualization 1 (Scatterplot) Run visualization1.py to create a scatterplot showing the relation of popularity rank and their average film rating. The method also outputs the calculated correlation coefficient of the scatterplot. The results show that there is a negative weak correlation. This shows that as popularity rank increases, the average film rating decreases. The correlation value is -0.19.

-- Visualization 2 (Histogram) Run calculate.py to populate the film_avg attribute of the Actor's table, create a histogram of the average film ratings for the top and bottom half of the top 100 list, and output a JSON formatted text file called calculation_results.txt with the calculation results. The histogram shows that both sides of the rank have similar distribution but the top 10 are more distributed in the < 8.5 range. Overall, the top 50 actors do have, on average, films that are ranked higher, so there is a strong correlation between the two.

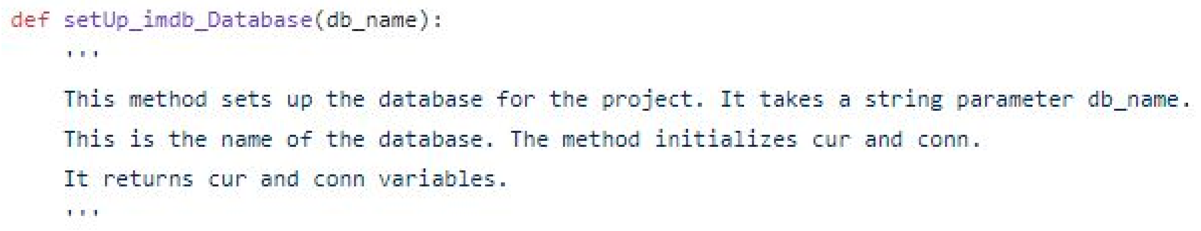
-- Visualization 3 (Pie chart) Run visualization2.py to create a pie chart depicting the different genres of the 731 films present in the Films table. This shows what kinds of films, popular actors mostly do. The top five results are Drama (23.4%), Documentary (14.7%), comedy (7.3%), Action (6.5%), Crime (6.2%)

--- Code Documentation ---

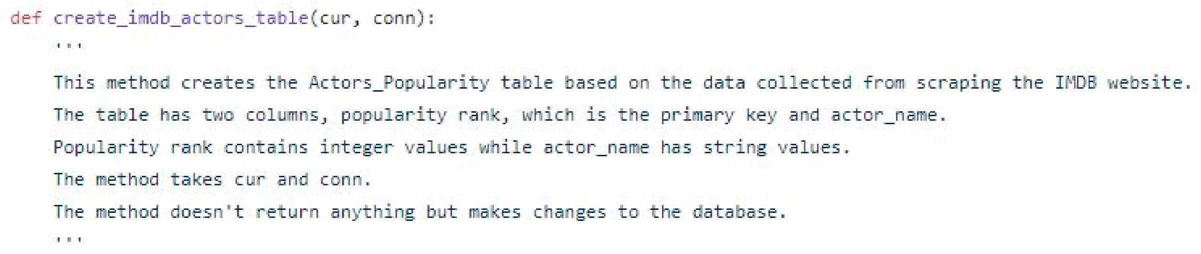
Methods in Part1.py



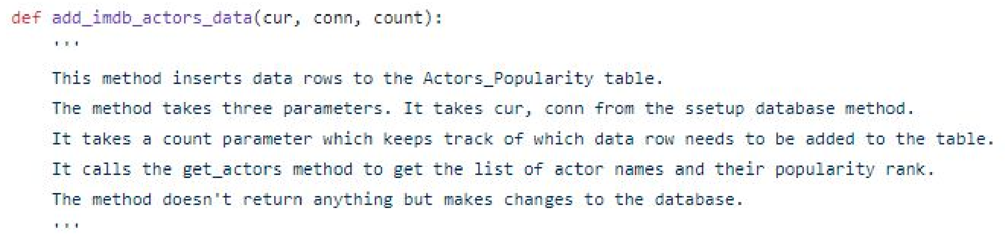
```python
def get_actors():
    '''
    Scrapes an IMDB website that contains the names and the ranks of the most popular actors till 2018.
    The method scrapes the names and the ranks and puts each of them in a seperate list.
    The method returns a final list, contaning the two previously mentioned lists.
    '''
```



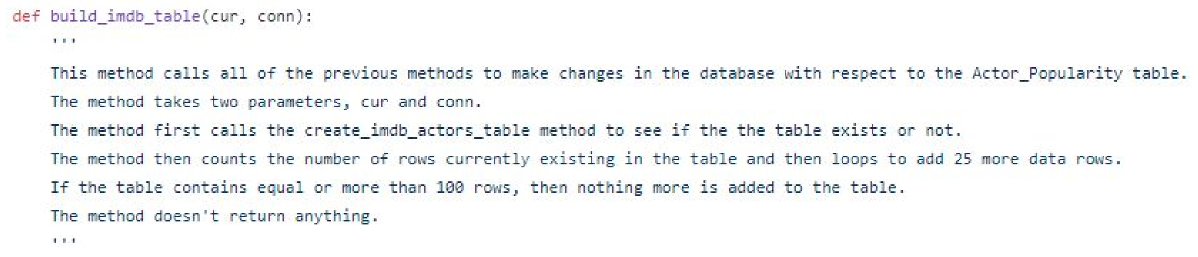
```python
def setUp_imdb_Database(db_name):
    '''
    This method sets up the database for the project. It takes a string parameter db_name.
    This is the name of the database. The method initializes cur and conn.
    It returns cur and conn variables.
    '''
```



```python
def create_imdb_actors_table(cur, conn):
    '''
    This method creates the Actors_Popularity table based on the data collected from scraping the IMDB website.
    The table has two columns, popularity rank, which is the primary key and actor_name.
    Popularity rank contains integer values while actor_name has string values.
    The method takes cur and conn.
    The method doesn't return anything but makes changes to the database.
    '''
```



```python
def add_imdb_actors_data(cur, conn, count):
    '''
    This method inserts data rows to the Actors_Popularity table.
    The method takes three parameters. It takes cur, conn from the ssetup database method.
    It takes a count parameter which keeps track of which data row needs to be added to the table.
    It calls the get_actors method to get the list of actor names and their popularity rank.
    The method doesn't return anything but makes changes to the database.
    '''
```



```python
def build_imdb_table(cur, conn):
    '''
    This method calls all of the previous methods to make changes in the database with respect to the Actor_Popularity table.
    The method takes two parameters, cur and conn.
    The method first calls the create_imdb_actors_table method to see if the the table exists or not.
    The method then counts the number of rows currently existing in the table and then loops to add 25 more data rows.
    If the table contains equal or more than 100 rows, then nothing more is added to the table.
    The method doesn't return anything.
    '''
```

```python
def get_total_actors_list():
    '''
    This method calls the get_actors method to get the list containing the names of the most popular actors.
    The method then creates a new list that contains the names of the actors.
    The names are slightly changed, where any space between the words is replaced by '%20'.
    This is done so that the new list can be run through an API query.
    The method return thee new list with the changed actor's names.
    '''
```

```python
def get_actors_data(actor_name):
    '''
    This method takes a string name of an actor.
    The method requests the 'themoviedb' api to get data about the actor.
    The requested data is recieved in the form of a json file and is then stored in the form of a dictionary object.
    The method returns a list containing the actor's id, his name and an empty string for films.
    The empty string is filled in a later method.
    '''
```

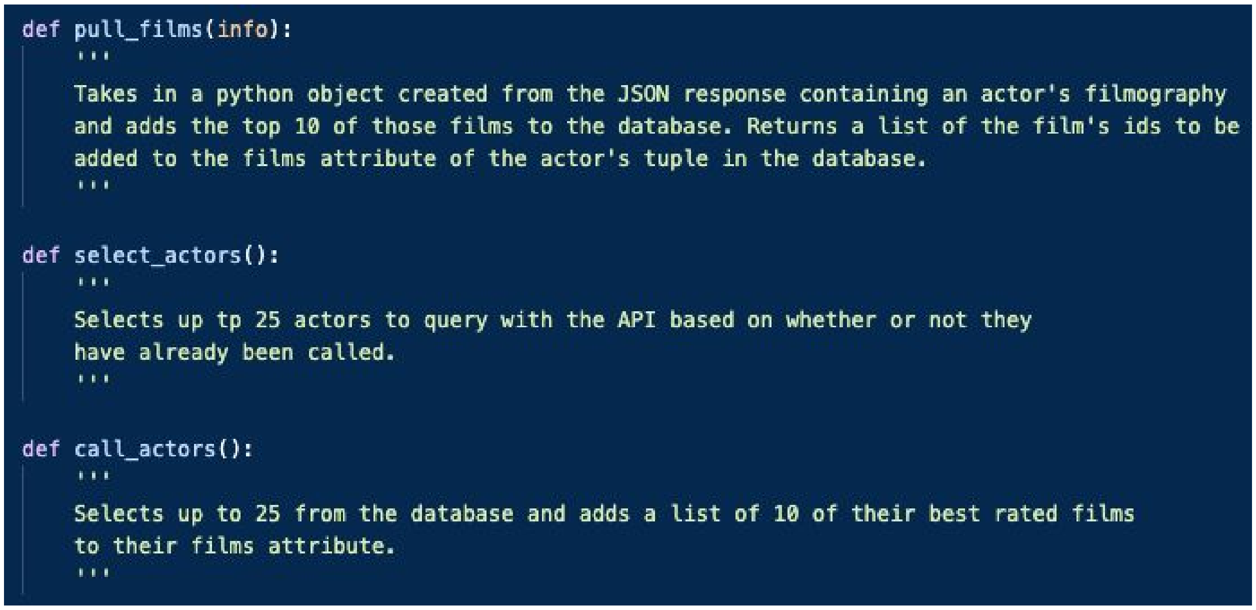
```python
def create_actors_table(cur, conn):
    '''
    This method creates the Actors table based on the data collected from requesting to the 'themoviedb' API.
    The table has three columns, actor_id which is the primary key, actor_name and actor_films.
    Actor_id contains integer values while actor_name and actor_films have string values.
    The method takes cur and conn.
    The method doesn't return anything but makes changes to the database.
    '''
```

```python
def add_actor_data(cur, conn, count):
    '''
    This method inserts data rows to the Actors table.
    The method takes three parameters. It takes cur, conn from the setup database method.
    It takes a count parameter which keeps track of which data row needs to be added to the table.
    It calls the get_total_actors_list method to get the list of actor names.
    Using this list the method calls the get_actors_data method to get the actor's id, name and films.
    The method doesn't return anything but makes changes to the database.
    '''
```

```python
def build_actors_table(cur, conn):
    '''
    This method calls all of the previous methods to make changes in the database with respect to the Actors table.
    The method takes two parameters, cur and conn.
    The method first calls the create_actors_table method to see if the the table exists or not.
    The method then counts the number of rows currently existing in the table and then loops to add 25 more data rows.
    If the table contains equal or more than 100 rows, then nothing more is added to the table.
    The method doesn't return anything.
    '''
```

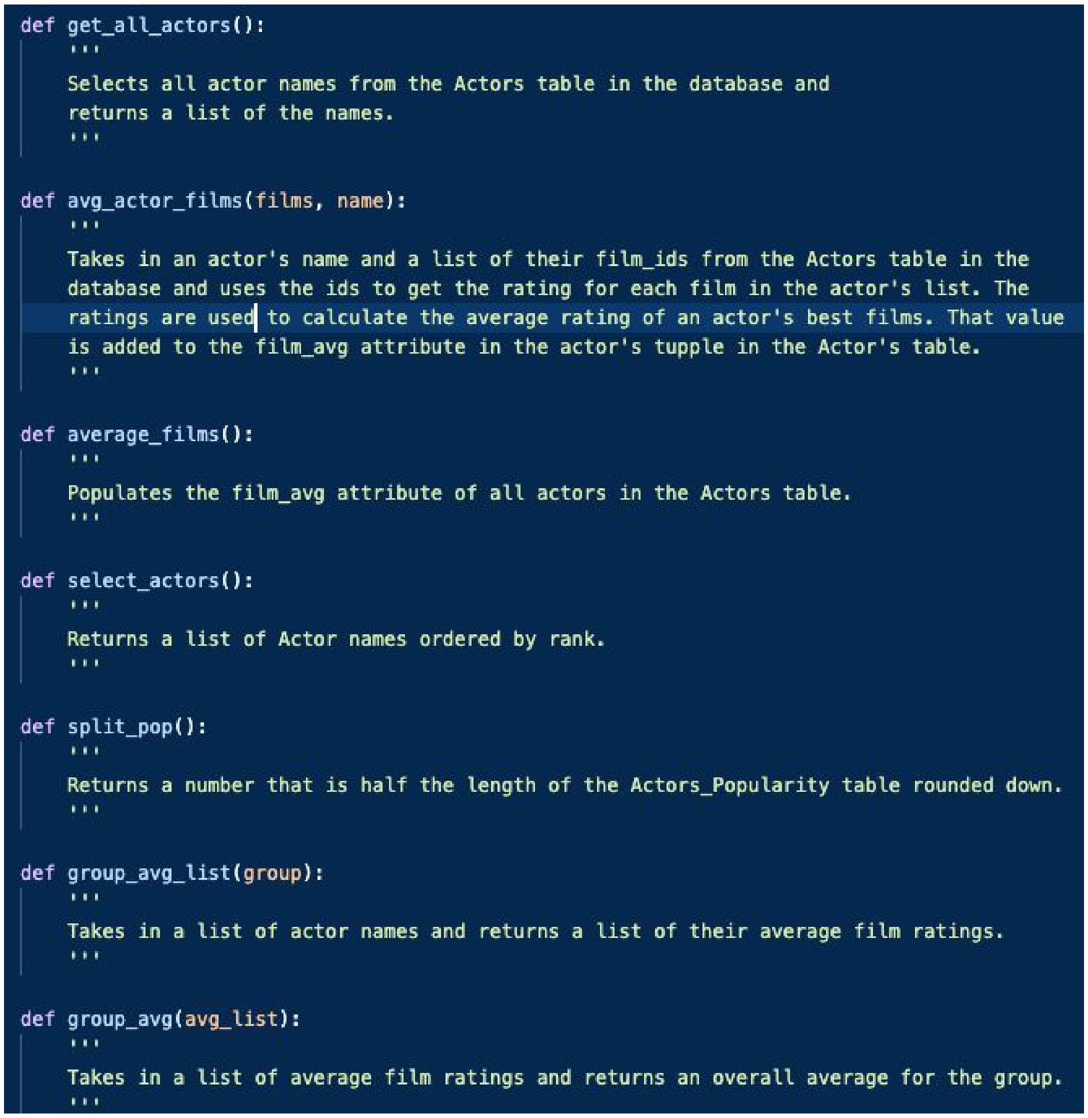
```python
def main():
    '''
    This is the main method. It calls three functions.
    It first calls the setUp_imdb_Database method to create a database file.
    It then calls the build_imdb_table to complete the Actors_Popularity Table.
    It then calls the build_actors_table to complete the Actors Table.
    '''
```

Methods in populate_films.py

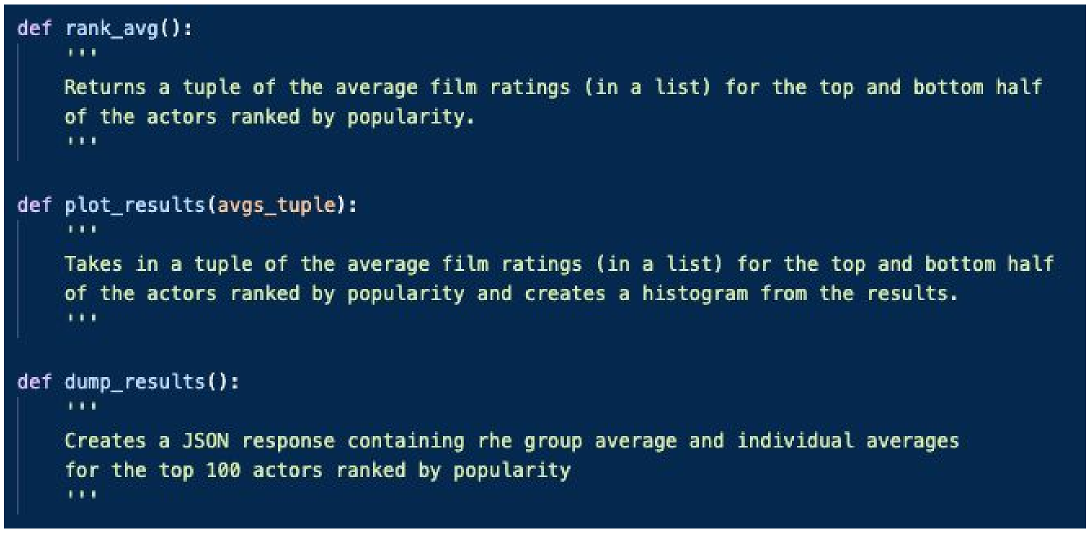


```
def pull_films(info):
    '''
    Takes in a python object created from the JSON response containing an actor's filmography
    and adds the top 10 of those films to the database. Returns a list of the film's ids to be
    added to the films attribute of the actor's tuple in the database.
    '''

def select_actors():
    '''
    Selects up tp 25 actors to query with the API based on whether or not they
    have already been called.
    '''

def call_actors():
    '''
    Selects up to 25 from the database and adds a list of 10 of their best rated films
    to their films attribute.
    '''
```

Methods in calculate.py

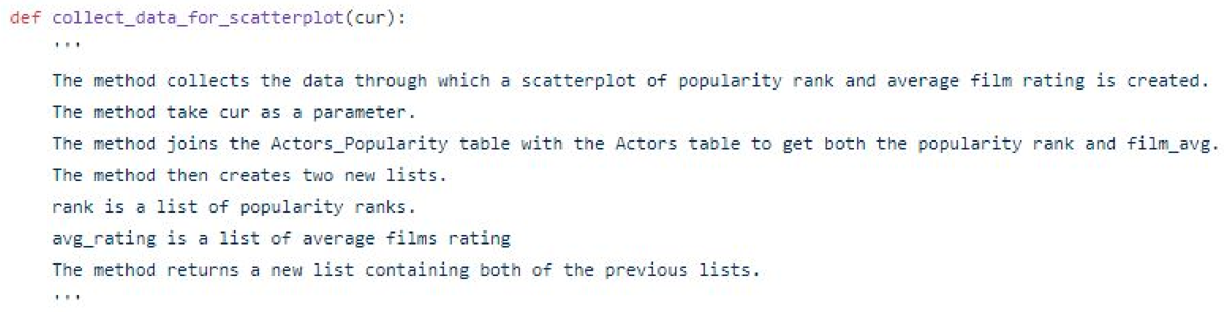


```
def get_all_actors():
    '''
    Selects all actor names from the Actors table in the database and
    returns a list of the names.
    '''

def avg_actor_films(films, name):
    '''
    Takes in an actor's name and a list of their film_ids from the Actors table in the
    database and uses the ids to get the rating for each film in the actor's list. The
    ratings are used to calculate the average rating of an actor's best films. That value
    is added to the film_avg attribute in the actor's tupple in the Actor's table.
    '''

def average_films():
    '''
    Populates the film_avg attribute of all actors in the Actors table.
    '''

def select_actors():
    '''
    Returns a list of Actor names ordered by rank.
    '''

def split_pop():
    '''
    Returns a number that is half the length of the Actors_Popularity table rounded down.
    '''

def group_avg_list(group):
    '''
    Takes in a list of actor names and returns a list of their average film ratings.
    '''

def group_avg(avg_list):
    '''
    Takes in a list of average film ratings and returns an overall average for the group.
    '''
```

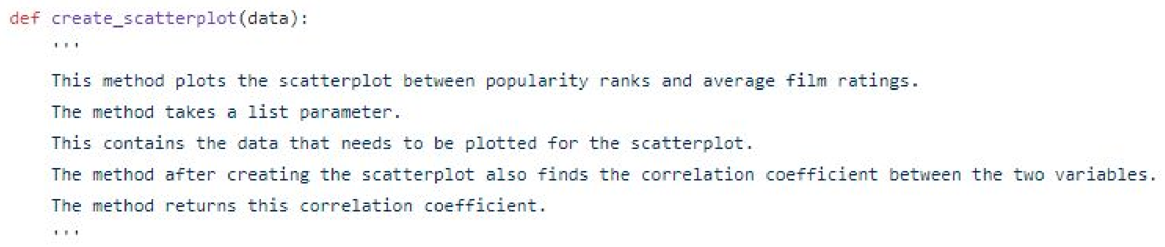


```python
def rank_avg():
    '''
    Returns a tuple of the average film ratings (in a list) for the top and bottom half
    of the actors ranked by popularity.
    '''

def plot_results(avgs_tuple):
    '''
    Takes in a tuple of the average film ratings (in a list) for the top and bottom half
    of the actors ranked by popularity and creates a histogram from the results.
    '''

def dump_results():
    '''
    Creates a JSON response containing rhe group average and individual averages
    for the top 100 actors ranked by popularity
    '''
```

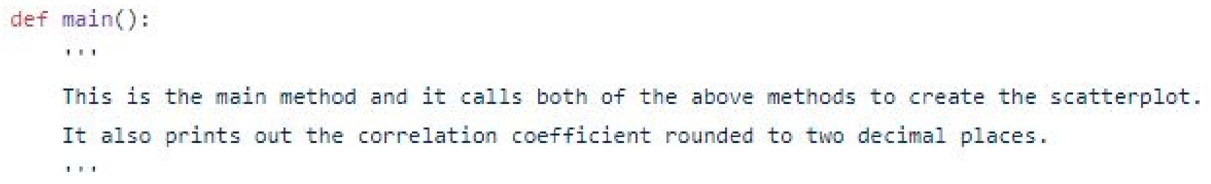
Methods in visualization1.py



```
def collect_data_for_scatterplot(cur):
    '''
    The method collects the data through which a scatterplot of popularity rank and average film rating is created.
    The method take cur as a parameter.
    The method joins the Actors_Popularity table with the Actors table to get both the popularity rank and film_avg.
    The method then creates two new lists.
    rank is a list of popularity ranks.
    avg_rating is a list of average films rating
    The method returns a new list containing both of the previous lists.
    '''
```



```
def create_scatterplot(data):
    '''
    This method plots the scatterplot between popularity ranks and average film ratings.
    The method takes a list parameter.
    This contains the data that needs to be plotted for the scatterplot.
    The method after creating the scatterplot also finds the correlation coefficient between the two variables.
    The method returns this correlation coefficient.
    '''
```



```
def main():
    '''
    This is the main method and it calls both of the above methods to create the scatterplot.
    It also prints out the correlation coefficient rounded to two decimal places.
    '''
```

Methods in visualization2.py

```python
def create_data_dict(cur):
    '''
    This method creates a dictionary object that keeps genre id's as keys and their count as values.
    The method takes the cur parameter to access the database.
    The method returns the dictionary object.
    '''

def clean_data_dict(dict):
    '''
    This method cleans the dictionary object to make it easier for creating a pe chart.
    The method takes the dictionary object as a parameter.
    The method returns a new list which contains genre_id's in int and then corresponding occurrences.
    '''

def add_genre_names(data, cur):
    '''
    This method updates the list and changes the genre id's to the names of genres.
    The method takes two parameters.
    It takes cur to access the database.
    It takes a list that contains information received from the previous method.
    It returns the updated list.
    '''

def dump_results(data):
    '''
    Creates a JSON response containing the different genre id's and the number of occurrences
    for the films of the top 100 actors ranked by popularity.
    The method takes the data as a a parameter.
    '''

def create_pie_chart(data):
    '''
    This method plots the pie chart.
    The method takes the final data that is needed to create the pie chart.
    The method doesn't return anything.
    '''

def main():
    '''
    The main method where code execution takes place.
    It calls the above mentioned four methods in order.
    '''
```

---Documentation of Resources Used ---

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
12-7-20 to 12-10-20 periodically	Forgetting the syntax for certain methods.	https://docs.python.org/3/tutorial/index.html	Yes.
12-04-20	Problem with scraping using selenium	StackOverflow	No
12-05-20	Problem understanding 25 limit requirment	Piazza	Yes
12-08-20	Understanding Matplotlib documentation	https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.scatter.html	Yes
12-06-20	Learning about Github Branches	https://thenewstack.io/dont-mess-with-the-master-working-with-branches-in-git-and-github/	Yes

List of packages used:

1) Sqlite3
2) Requests
3) beautifulSoup
4) Json
5) Os
6) Matplotlib
7) Matplotlib.pyplot
8) Numpy

List of Resources used

1) Piazza
2) StackOverflow
3) SI 206 Resources
4) MatplotLib documentation
5) Internet sources