Final Project Report: Sexism in Popular Movies

Deja VanOeveren-Goss

Maiara Lewis Cipriano

Margaret Baker

SI 206: Data-Oriented Programming

Prof. Barbara Ericson

December 12, 2023

Table of Contents

Project Goals

The goal of this project was to investigate sexism in popular movies. The initial plan was to gather information from the top 1000 movies of all time and the top billed actors in those movies using the following APIs and website:

1. Website: IMDb - Scrape the top 1000 movies of all time in order to gather the titles of all those movies.
2. API 1: OMDb API - Request data on all the 1000 movies, to collect the year the movie was released, the overall rating, and the main actors (top 3, being those the actors with main roles, and consequently higher pay).
3. API 2: Celebrity API (Ninjas API) - Request data on all the top 3 actors of every given movie, to collect their personal information, such as gender, age, birthday, net worth, and aliveness.

After starting the project, we had to pivot and use a different website than the one planned. The website used to generate the results obtained from this project was the following:

List Challenges Website: top 250 movies of all time, originated from an IMDb list.

After changing the website from where we would be gathering the movie titles, we were able to successfully gather information from both APIs, store it in a database, and make calculations such as the average net worth of male and female actors, and how different is the popularity of movies led by females as compared to males, to analyze whether sexism could be observed.

Documentation for Functions

The documentation below is sectioned by files. Here's the link to our Github repository.

1. **get_movies_list.py file:** this file contains one function only called `get_movies_list`. This function scrapes the website linked in the "Project Goals" section of this report using Beautiful Soup. The function takes no specific parameters, and returns a list of strings, where each string is a movie title.
2. **api_factory.py file:** this file contains two functions that gather data from both APIs used in the project (OMDb API and Celebrity API).
    a. `get_omdbapi_movie_data:` this function takes one parameter, a list of strings where each string is a movie title. Using each movie title in the list, the function requests information from the OMDb API and stores it in a tuple. Each tuple represents a movie, and it has the following information: title of the movie, the year the movie was released, rotten tomatoes rating, metascore rating, IMDb rating, and a list of the three main actors in the movie.

    This function returns a list of tuples, representing all the data that needs to be added to the database for each of the movies.

    b. `get_celebrityapi_actors_data:` this function takes a list of actors and retrieves information about each of the actors in the list from the Celebrity API. It returns two lists. This first list is a list of tuples with the following information: actor's name (string), age (integer), gender (string), birthday (string), net worth (integer), and aliveness (boolean). The second list is a list of strings, where each string is the name of an actor whose information was not found in the API.

3. **database_utilities.py file:** this file contains a total of six functions to create and fill in a database.
    a. `setup_database_connection:` this function creates a database cursor and connection. It takes a string as a parameter that represents the name of the database being created.

       The function returns the cursor and connection for a database.
    b. `setup_all_tables`: this function takes a database cursor and a connection as parameters, and creates three tables in the database: Movies, with the following columns: `movie_id`, `title`, `year`, `rotten_tomatoes`, `metascore`, and `imdb`; Actors, with seven columns: `actor_id`, `name`, `gender`, `age`, `birthday`, `net_worth`, and `is_alive`; and Movies_and_Actors, with the columns `movie_actor_combination_id`, `actor_id`, and `movie_id`.

       This function does not return anything.
    c. `add_omdbapi_data_to_database:` this function adds information collected from the OMDb API, returned from the `get_omdbapi_movie_data` function, to a database. It takes three parameters: a list, with all the data that will be added to the database, a database cursor, and a database connection. The function only adds three items from the passed list to the database per run. This function does not return anything.
    d. `get_last_actors_id:` this function takes a database cursor as a parameter, and collects the last `actor_id` available in the database in order to keep track of

which actor was last added. It returns an integer as the last actor_id of the database.

e. `add_celebrityapi_data_to_database:` this function takes three parameters, an integer that represents the last `actor_id` in the database, and a database cursor and connection. Using that information, the function creates a list of all actors' names that were last added to the database, retrieves data about those actors using the Celebrity API, and adds to the Actors table in the database. This function does not return anything.

f. `load_database:` this function takes a string as a parameter, that represents the name of the database, and calls all the functions described above, as well as `get_movies_list` from the `get_movies_list.py` and `get_omdbapi_movie_data` from the `api_factory.py` file in order to create and fill a database.

4. **calculations.py file:** this file contains a total of six functions that help compute specific calculations using the data stored in the database.

   a. `access_database:` this function takes one parameter: string, that represents the name of the database and accesses the database with the given name. The parameter default value is set to be the name of the database created for this project - “Movies_And_Actors_Database.”

   The function returns the cursor and connection for the database.

   b. `write_txt_file:` this function takes three parameters, a string representing the file name, a second string representing the title of the file section about to be written, and a list that represents all the lines that need to be written into the file.

The function does not overwrite the file content if the file already exists, and instead, appends the new content to the end of the file.

In order to enable this function, access the `config.py` file and set the variable `WRITE_TO_CALCULATIONS_RESULTS` to **`True`**. This function does not return anything.

c. `Calculate_average_networth_based_on_gender:` this function takes a database cursor as a parameter. It calculates the average net worth of actors in the database based on their gender, females and males, respectively, and writes the results of those calculations into a text file. The function returns the average net worth of female actors, and the average net worth of male actors, respectively as integers.

d. `Calculate_average_imdb_rating_based_on_gender_year:` this function takes a database cursor, a string representing a gender, and an integer representing a year. It calculates the average imdb rating of the movies in the database with more actors of the given gender BEFORE and ON/AFTER the given year. This function returns the average IMDb rating before the given year, and the average rating on/after that given year as float numbers, respectively.

e. `calculate_slope_of_age_trend_over_years:` this function takes a database cursor as a parameter. It calculates the slope of the actors' age trend over the years. It returns the information necessary to build a scatterplot: a tuple with format (list A, list B) where list A = movie years, and list B = actors' ages; the slope of the best-fit line as a float number with 4 decimal places; and the y-intercept.

f. `calculate_slope_of_gender_age_trend_over_years:` this function takes a database cursor and a string representing a gender (female or male). It calculates the slope of the given gender actors' age trend over the years. It returns the information necessary to build a scatterplot: a tuple with format (list A, list B) where list A = movie years, and list B = actors' ages; the slope of the best-fit line as a float number with 4 decimal places; and the y-intercept.

g. `main:` this function does not take any parameters, and it calls all the functions described above in order to run all the calculations. The function does not return anything.

5. **visualizations.py file**: this file contains four functions that are used to create visualizations for the calculations in the `calculations.py` file.

   a. `plot_average_net_worth_based_on_gender:` this function takes a database cursor and creates a bar graph with the the average net worth of females and males showcased side by side. It does not return anything.

   b. `plot_average_imdb_rating_based_on_gender_year:` this function takes a database cursor, a string representing a gender, and an integer representing a year and plots the average IMDb rating of gender-led movies before and on/after the given year. The function does not return anything.

   c. `plot_scatterplot_of_age_trend:` this function takes a database cursor and creates a scatterplot with a best-fit line representing a potential actor's age trend over the years. It does not return anything.

   d. `plot_scatterplot_of_gender_age_trend:` this function takes a database cursor and a string representing a gender (female or male), and creates a

scatterplot with a best-fit line representing a potential actor's age trend over the years for actors of the given gender.

e. `main:` this function does not take any parameters, and it calls all the functions described above in order to create all the visualizations. The function does not return anything.

The Problems

During this project, we did not face too many problems. However, there were some complications along the way.

1. **IMDb 403 Forbidden Response:** Right at the beginning of this project, when we attempted to use the planned website for scraping, we were surprised with a 403 response. After reviewing the IMDb policies and conditions of use, we were able to confirm that they do not allow scraping data from their website. Because of that, we had to pivot and find another website that would present us with a valid list of movies.
2. **25 Items Per Run Restriction:** An indirect problem we had with the 25 limit restriction was that it made us change the number of movie titles we should collect. Initially, we wanted to have a list of 1000 movies to have more data to draw conclusions from. However, it was unreasonable to have 1000 movies because we could only add 3 movies to the database at a time. To add that amount of data to our database, the program would have needed to run an excessive amount of times. Because of that, we reduced the number of movies to 100, which led to a different problem.

   Since not all 100 movies would have information available on the OMDb API, we ended up violating the "100 items per API/website" requirement. Which made us finally select a list of 250 movies.
3. **Stopping Before Adding All Movies to the Database:** When creating the function that was gathering the movies' information from the OMDb API, we added a 'try' and 'except' logic that would prevent considering movies that didn't have the desired information available. This function would return a list of tuples, where each tuple had all the necessary information for a movie (title, release year, ratings, and actors). Because the

line of code that appended each tuple to the return list was written outside the ‘try’, that list of data ended up having repeated movies since, as soon as an exception occurred, the previous movie with valid information would be appended again to the list. This caused a problem when adding the OMDb information to the database because that movie was already in it, which prevented all movies after that from being added.

Calculations

Calculation Results:

calculations_results.txt

```
Average Net Worth of Actors Based on Gender

Average Net Worth of Female Actors Who are Still Alive: 38177215
Average Net Worth of Male Actors Who are Still Alive: 69688648

------------------------------------------------------------------------
Popularity of Female-Led Movies Before and After 2000

Average IMDb rating BEFORE 2000: 8.2
Average IMDb rating ON and AFTER 2000: 8.3

------------------------------------------------------------------------
Popularity of Male-Led Movies Before and After 2000

Average IMDb rating BEFORE 2000: 8.4
Average IMDb rating ON and AFTER 2000: 8.3

------------------------------------------------------------------------
Actors Age Trend Over Nearly 100 Years

Slope: 0.0733

------------------------------------------------------------------------
Female Actors Age Trend Over Nearly 100 Years

Slope: 0.1084

------------------------------------------------------------------------
Male Actors Age Trend Over Nearly 100 Years

Slope: 0.0544

------------------------------------------------------------------------
```

Calculation Functions:

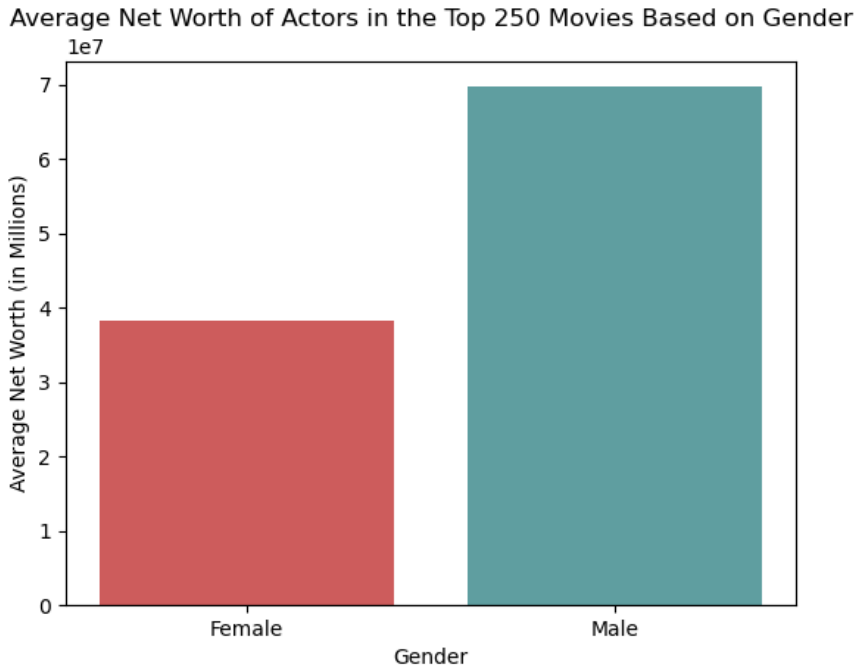
```python
def calculate_average_networth_based_on_gender(cur):
    """…

    # Gather the net worth data of all female actors that are still alive
    cur.execute("SELECT net_worth FROM Actors WHERE gender =(?) AND is_alive =(?)", ("female", 1))
    all_female_actors = cur.fetchall()

    # Combine the net worth of all female actors alive
    total_net_worth_females = 0
    for actress in all_female_actors:
        actress_net_worth = actress[0]

        total_net_worth_females += actress_net_worth

    # Calculate average net worth of female actors alive
    average_networth_females = int(total_net_worth_females / len(all_female_actors))


    # Gather the net worth data of all male actors that are still alive
    cur.execute("SELECT net_worth FROM Actors WHERE gender=(?) AND is_alive=(?)", ("male", 1))
    all_male_actors = cur.fetchall()

    # Combine the net worth of all male actors alive
    total_net_worth_males = 0
    for actor in all_male_actors:
        actor_net_worth = actor[0]

        total_net_worth_males += actor_net_worth

    # Calculate average net worth of female actors alive
    average_networth_males = int(total_net_worth_males / len(all_male_actors))


    # Write information to the calculation file
    females_networth_info = f"Average Net Worth of Female Actors Who are Still Alive: {average_networth_females}"
    males_networth_info = f"Average Net Worth of Male Actors Who are Still Alive: {average_networth_males}"
    information = [females_networth_info, males_networth_info]
    write_txt_file("calculations_results.txt", "Average Net Worth of Actors Based on Gender", information)

    return average_networth_females, average_networth_males
```

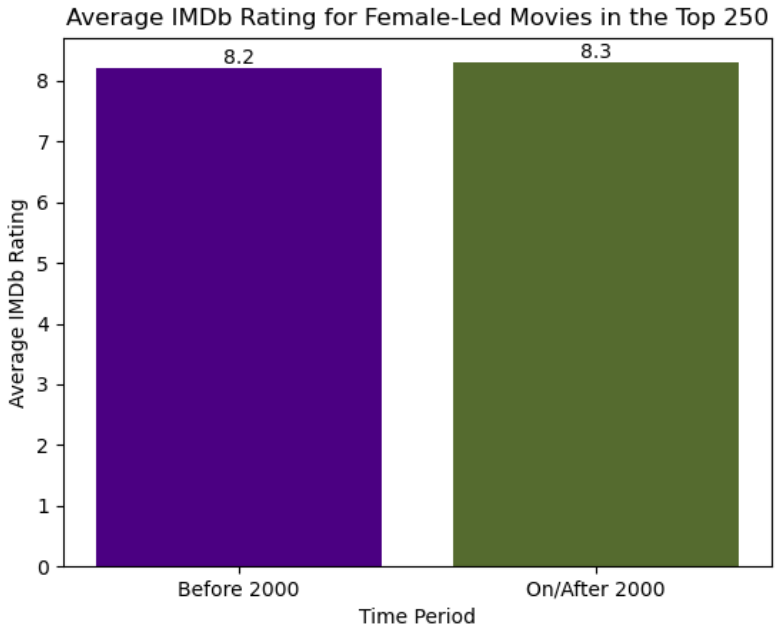
```python
def calculate_average_imdb_rating_based_on_gender_year(cur, gender: str, year: int):
    """…

    # Join all three tables to gather the IMDb ratings for the movies released ON or AFTER a given year
    # where there are more actors of a given gender
    cur.execute("""SELECT imdb FROM Movies
                JOIN Movies_and_Actors ON Movies.movie_id = Movies_and_Actors.movie_id
                JOIN Actors ON Movies_and_Actors.actor_id = Actors.actor_id
                WHERE Actors.gender =(?) AND Movies.year >= (?)
                GROUP BY Movies.movie_id HAVING COUNT(DISTINCT Actors.actor_id) >= (?)""", (gender, year, 2))

    all_ratings_on_and_after_year = cur.fetchall()

    # Calculate the average IMDb rating of the movies follow the given criteria
    all_imdb_ratings = 0

    for movie in all_ratings_on_and_after_year:
        movie_rating = movie[0]

        # Combine all the IMDb rating values
        all_imdb_ratings += movie_rating
    average_imdb_rating_on_and_after_year = all_imdb_ratings / len(all_ratings_on_and_after_year)
    # Round the average calculation result to have only 1 decimal
    average_imdb_rating_on_and_after_year = round(average_imdb_rating_on_and_after_year, 1)


    # Join all three tables to gather the IMDb ratings for the movies released BEFORE a given year
    # where there are more actors of a given gender
    cur.execute("""SELECT imdb FROM Movies
                JOIN Movies_and_Actors ON Movies.movie_id = Movies_and_Actors.movie_id
                JOIN Actors ON Movies_and_Actors.actor_id = Actors.actor_id
                WHERE Actors.gender =(?) AND Movies.year < (?)
                GROUP BY Movies.movie_id HAVING COUNT(DISTINCT Actors.actor_id) >= (?)""", (gender, year, 2))

    all_ratings_before_year = cur.fetchall()
    # Calculate the average IMDb rating of the movies follow the given criteria
    all_imdb_ratings = 0

    for movie in all_ratings_before_year:…
    average_imdb_rating_before_year = all_imdb_ratings / len(all_ratings_before_year)
    # Round the average calculation result to have only 1 decimal
    average_imdb_rating_before_year = round(average_imdb_rating_before_year, 1)


    # Write information to the calculation file
    before_year_info = f"Average IMDb rating BEFORE {year}: {average_imdb_rating_before_year}"
    on_after_year_info = f"Average IMDb rating ON and AFTER {year}: {average_imdb_rating_on_and_after_year}"

    information = [before_year_info, on_after_year_info]
    write_txt_file("calculations_results.txt", f"Popularity of {gender.capitalize()}-Led Movies Before and After {year}", information)

    return average_imdb_rating_before_year, average_imdb_rating_on_and_after_year
```

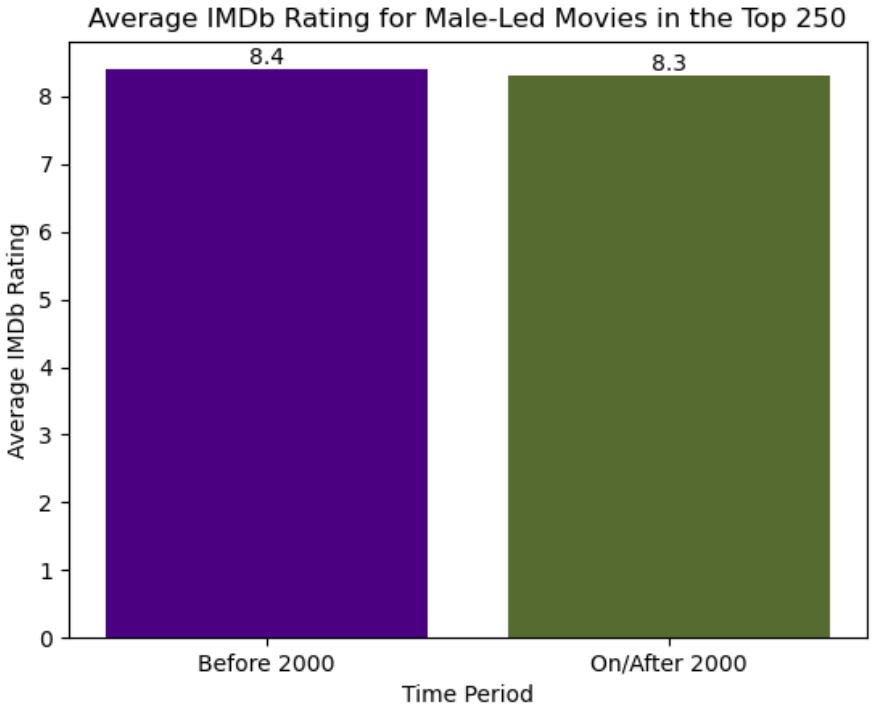
```python
def calculate_slope_of_age_trend_over_years(cur):
    """…

    # Gather all movies years and the valid actors birthdays
    cur.execute("""SELECT year, birthday FROM Movies
                JOIN Movies_and_Actors ON Movies.movie_id = Movies_And_Actors.movie_id
                JOIN Actors ON Movies_And_Actors.actor_id = Actors.actor_id
                WHERE birthday IS NOT NULL
                """)
    list_of_data_points = cur.fetchall()

    x_values = []
    y_values = []


    for point in list_of_data_points:
        # Extract data from the tuple
        movie_year, birthday = point

        # Select actor's birth year and convert to an integer
        birth_year = int(birthday.split("-")[0])

        # Calculate the age of the actor when the movie was released
        actor_age = movie_year - birth_year

        # If the actor is not a child actor, consider that data point
        if actor_age > 17:
            y_values.append(actor_age)
            x_values.append(movie_year)

    # Check if there is a trend over the years by calculting the slope
    slope, y_intercept = numpy.polyfit(x_values, y_values, 1)
    slope = round(slope, 4)

    # Write information to the calculation file
    information = [f"Slope: {slope}"]
    write_txt_file("calculations_results.txt", "Actors Age Trend Over Nearly 100 Years", information)

    # Return data to build scatterplot and best-fit line
    return (x_values, y_values), slope, y_intercept
```

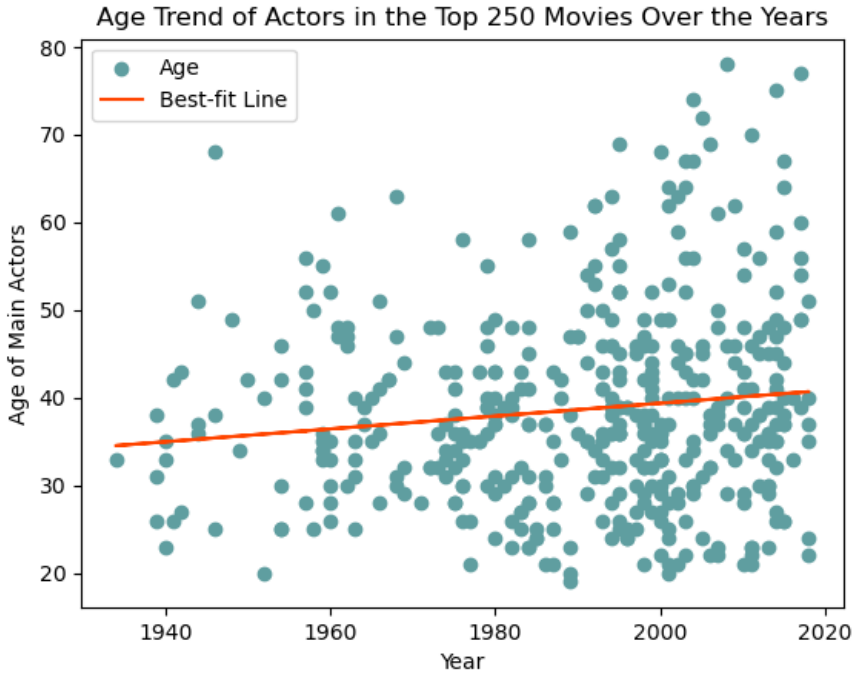
```python
def calculate_slope_of_gender_age_trend_over_years(cur, gender: str):
    """…

    # Gather all movies years and the valid actors birthdays
    cur.execute("""SELECT year, birthday FROM Movies …
    list_of_data_points = cur.fetchall()

    x_values = []
    y_values = []


    for point in list_of_data_points:
        # Extract data from the tuple
        movie_year, birthday = point

        # Select actor's birth year and convert to an integer
        birth_year = int(birthday.split("-")[0])

        # Calculate the age of the actor when the movie was released
        actor_age = movie_year - birth_year

        # If the actor is not a child actor, consider that data point
        if actor_age > 17:
            y_values.append(actor_age)
            x_values.append(movie_year)

    # Check if there is a trend over the years by calculting the slope
    slope, y_intercept = numpy.polyfit(x_values, y_values, 1)
    slope = round(slope, 4)

    # Write information to the calculation file
    information = [f"Slope: {slope}"]
    write_txt_file("calculations_results.txt", f"{gender.capitalize()} Actors Age Trend Over Nearly 100 Years", information)

    # Return data to build scatterplot and best-fit line
    return (x_values, y_values), slope, y_intercept
```

Visualizations

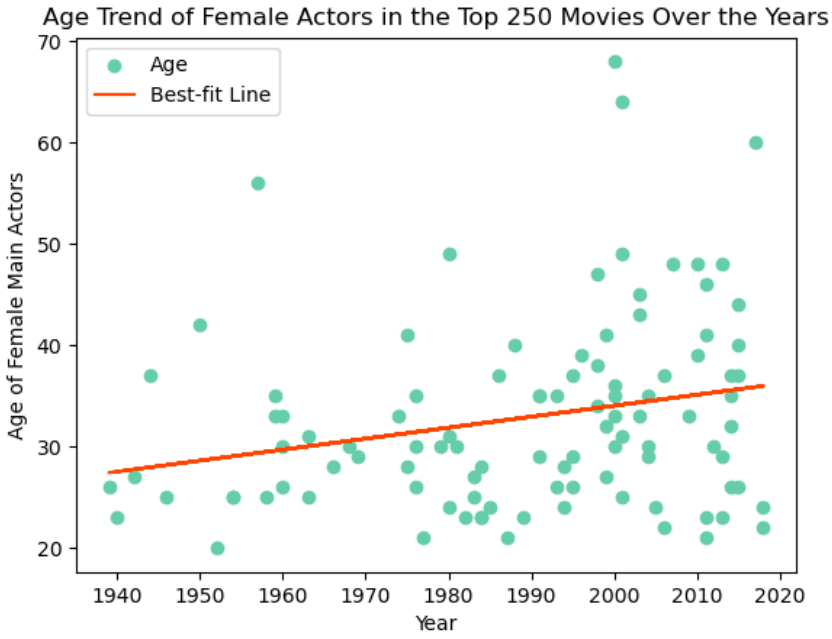


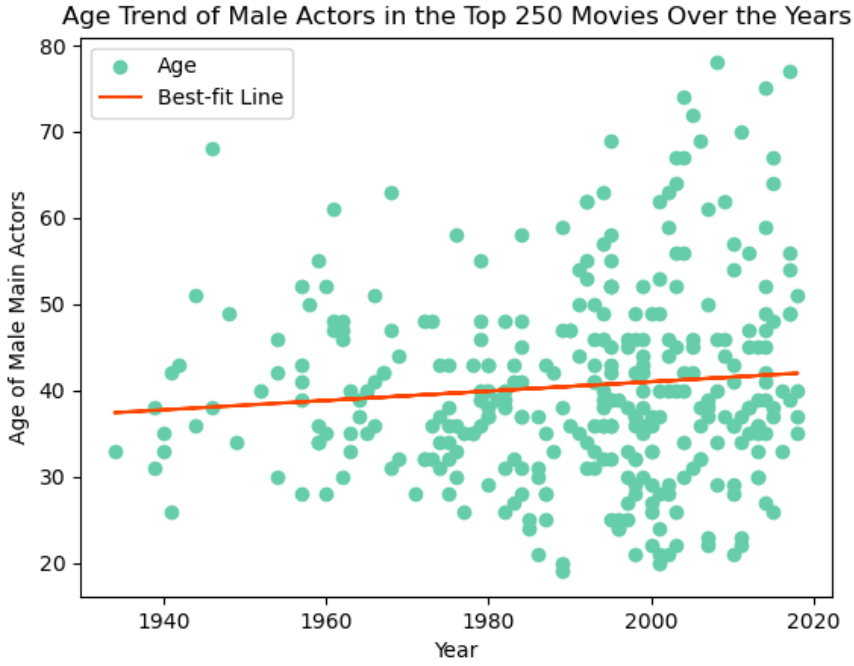



Average IMDb Rating for Male-Led Movies in the Top 250
8.4
8.3
Average IMDb Rating
0
1
2
3
4
5
6
7
8
Before 2000
On/After 2000
Time Period


Age Trend of Actors in the Top 250 Movies Over the Years
Age
Best-fit Line
Age of Main Actors
20
30
40
50
60
70
80
1940
1960
1980
2000
2020
Year

The two visualizations below were created AFTER the presentation/grading session!





Instructions for Running the Program

Creating and Filling the Database:

1. If accessing the program from the zip file submitted in Canvas, please go to the file named `apicredentials.py` and insert your API keys for both the OMDb and Celebrity APIs.

   If accessing the program by cloning the repository in Github, please CREATE a file named `apicredentials.py` and fill it with the information given in the `README.md` file.

   The steps above must be followed if you are trying to recreate the database with a different name; otherwise, since the filled database is available, the `apicredentials.py` file just needs to exist, and you don't have to insert your API keys in it.
2. If recreating the database, please go to the `database_utilitites.py` file and assign a different name to the `database_name` variable on line 140.

   If you're not recreating the database, then run the file to see the message *"We collected all movies already!"*
3. If recreating the database, to fill it with all the data available, please run the file called `run_database.sh`, this will allow the `database_utilities.py` file to run automatically enough times to insert all the data in the new database you are creating.

Running the Calculations:

1. Delete or change the name of the existing text file called `calculations_results.txt` so you can observe the calculations file being created.
2. Go to the file named `config.py` and change the statement to True.
3. Go back to the `calculations.py` file and:

   If you created a completely new database and inserted all the available data using `run_database.sh`, then go to line 266 and pass your database name to the function `access_database()`. If you're using the Movie_And_Actors_Database already available, you do not need to follow this step.

4. Run the file to see that there are no trackbacks and a new `calculations_results.txt` file was created.

Creating the Visualizations:

1. Go to the file called `visualizations.py`.
2. If you created a new database and are using that to create visualizations, then go to line 98 inside the `main()` function, and make sure to pass the new database name as a string to the `access_database()` function.

   If you are creating the visualizations from the Movies_And_Actors_Database already available, then you do not need to follow this step.
1. Run the file to see 6 graphs appear as visualizations.

Congratulations, you have successfully run our program!

Resources Used

Date	Issue Description	Resource	Result (Did it solve it?)
12/01	Needed to make sure apicredentials.py file wouldn't be accidentally pushed to the Github repository by any members of the team	Medium Resource Link	Yes, using gitignore allowed us to work with API keys without the risk of exposing them.
12/02	I needed to run the program enough times to fill the database with all the data but didn't want to do it manually	Stackoverlow Resource Link	Yes, I was able to successfully make the database-utilities.py file run automatically 85 times.
12/07	Needed to use different colors for the visualizations but didn't know the options.	Matplotlib Resource Link	Yes, I was able to select different, interesting colors for the two extra visualizations created after the presentation.
12/09	I needed to add value labels to our IMDb rating bar graphs to showcase the very slight difference in ratings.	Geeks for Geeks Resource Link	Yes, I was able to successfully add value labels to the bars, as seen above.
12/09	I was having trouble using Beautiful Soup to input the Top Movie Titles into a list and wanted to make sure that my code was written correctly.	RealPython Beautiful Soup Resource Link	Yes, I realized that my code was written correctly but that IMDb did not want me to scrape their data. We pivoted to another website to scrape