# SI 206 Final Project Report

## CodeCrafters

By: Priya Shah, Dhara Patel, and Nethra Vijayakumar

## Table of Contents

Github Repository Link: https://github.com/shhapriya/SI-206-Final-Project-

## Introduction

For our project, we chose to analyze data for different songs, movies, and basketball players through our APIS. We aimed to use a variety of APIS to understand the differences between handling data among different websites. By doing this, we were able to understand correlations and relationships between data, and were able to demonstrate these through various visualizations and calculations.

## Initial Goals

Our Initial goals were to work with the Spotify, NASA and Tumblr API's.

1) Spotify
    a) Gather the following data:
        i) Song name and artist
        ii) Album they belong to
        iii) Release year
        iv) Song length
        v) Popularity score

    b) Calculate the following:
        i) Average popularity score
        ii) Most popular artists
        iii) Top listened to songs

2) NASA
    a) Gather the following data:
        i) Asteroid/comet size
        ii) Asteroid/comet speed
        iii) Closest approach to Earth

    b) Calculate the following:
        i) Average size of asteroids/comets
        ii) Average speed of asteroids/comets
        iii) The asteroid/comet that is closest to and farthest from Earth

3) Tumblr
    a) Gather the following data:
        i) Blog name
        ii) Blog description
        iii) Blog follower count and post count

    b) Calculate the following:
        i) Average follower count and average post count

ii) Average length of blog name
iii) Average growth of follower count over a specified period of time

## Actual Goals Achieved

We worked with the Last FM, OMDb, and NBA API's.

1) Last FM
    a) Gathered the following data:
        - Top Tracks (Names, Artists, Listeners, Playcounts)
        - Top Artists (Artist Names, Playcounts, Listeners)
    b) Calculated the following:
2) NBA
    a) Gathered the following data:
        - Player names
        - Player data
            - Depth chart order
            - Weight
            - Height
            - Salary
            - Years of experience
    b) Calculated the following:
        - Average weight
        - Average height
        - Average salary
        - Average years of experience
        - Average depth chart order
        - The correlation between salary and experience

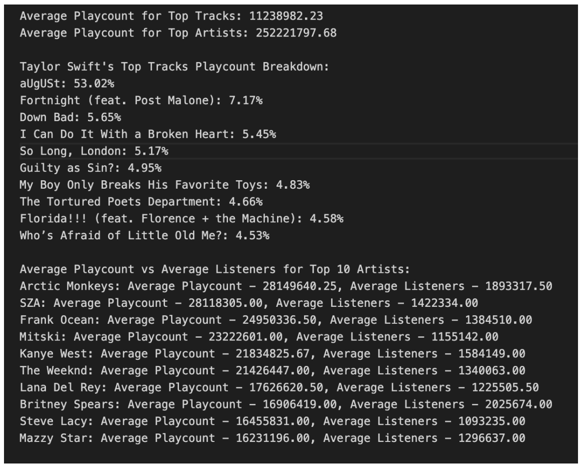
3) OMDb
    a) Gathered the following data:
        i) Movie title
        ii) Movie rating from IMDb
        iii) Movie rating from Rotten Tomatoes
        iv) Movie rating from metacritic
        v) Movie genre
    b) Calculated the following:
        i) Average movie rating from IMDb
        ii) Average Movie rating from Rotten Tomatoes
        iii) Average Movie rating from Metacritic
        iv) Percentage of movie ratings above two from IMDb
        v) Percentage of movie ratings above two from Metacritic

## Problems We Faced

- One of the biggest problems we faced was getting data from the APIS. We were having a lot of trouble with the Spotipy API in regards to the key and getting the data to load 25 or fewer rows at a time. When we tried using other APIS such as SoundCloud and Amazon Music, they were more focused on integration within other websites rather than allowing access to large amounts of data.
- Another problem we faced was how to deal with null values in the rating column for each rating site in the Movie_Ratings table when doing calculations. We ran into some problems when trying to calculate average ratings as you can't add a null value. To fix this issue, we used the Coalesce function.
- In the NBA data, almost all the values had data for each player. However, not every player had their years of experience listed. During calculations, this became an issue, because I was trying to extract null values. I fixed this issue differently than my peers - in my query, I added “IS NOT NULL.”

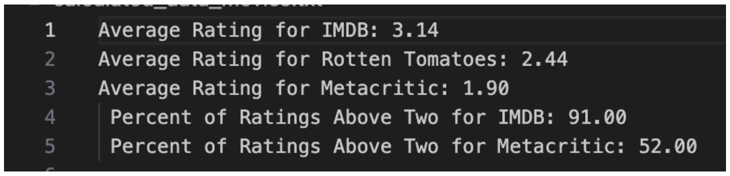
# Calculations

*Last FM:*



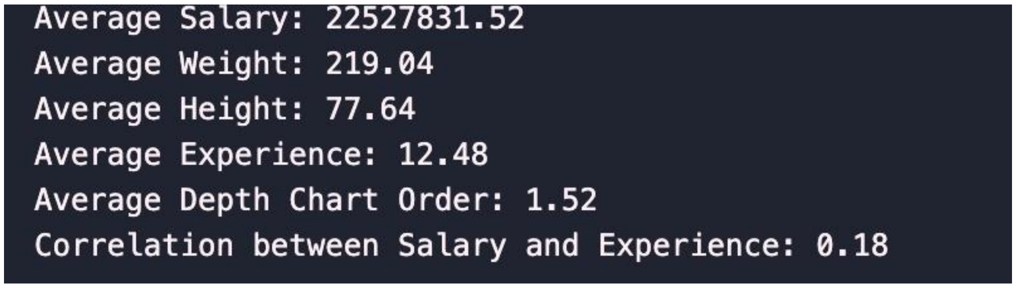
```
Average Playcount for Top Tracks: 11238982.23
Average Playcount for Top Artists: 252221797.68

Taylor Swift's Top Tracks Playcount Breakdown:
aUgUSt: 53.02%
Fortnight (feat. Post Malone): 7.17%
Down Bad: 5.65%
I Can Do It With a Broken Heart: 5.45%
So Long, London: 5.17%
Guilty as Sin?: 4.95%
My Boy Only Breaks His Favorite Toys: 4.83%
The Tortured Poets Department: 4.66%
Florida!!! (feat. Florence + the Machine): 4.58%
Who’s Afraid of Little Old Me?: 4.53%

Average Playcount vs Average Listeners for Top 10 Artists:
Arctic Monkeys: Average Playcount - 28149640.25, Average Listeners - 1893317.50
SZA: Average Playcount - 28118305.00, Average Listeners - 1422334.00
Frank Ocean: Average Playcount - 24950336.50, Average Listeners - 1384510.00
Mitski: Average Playcount - 23222601.00, Average Listeners - 1155142.00
Kanye West: Average Playcount - 21834825.67, Average Listeners - 1584149.00
The Weeknd: Average Playcount - 21426447.00, Average Listeners - 1340063.00
Lana Del Rey: Average Playcount - 17626620.50, Average Listeners - 1225505.50
Britney Spears: Average Playcount - 16906419.00, Average Listeners - 2025674.00
Steve Lacy: Average Playcount - 16455831.00, Average Listeners - 1093235.00
Mazzy Star: Average Playcount - 16231196.00, Average Listeners - 1296637.00
```

*OMDb:*



```
Average Rating for IMDB: 3.14
Average Rating for Rotten Tomatoes: 2.44
Average Rating for Metacritic: 1.90
 Percent of Ratings Above Two for IMDB: 91.00
 Percent of Ratings Above Two for Metacritic: 52.00
```

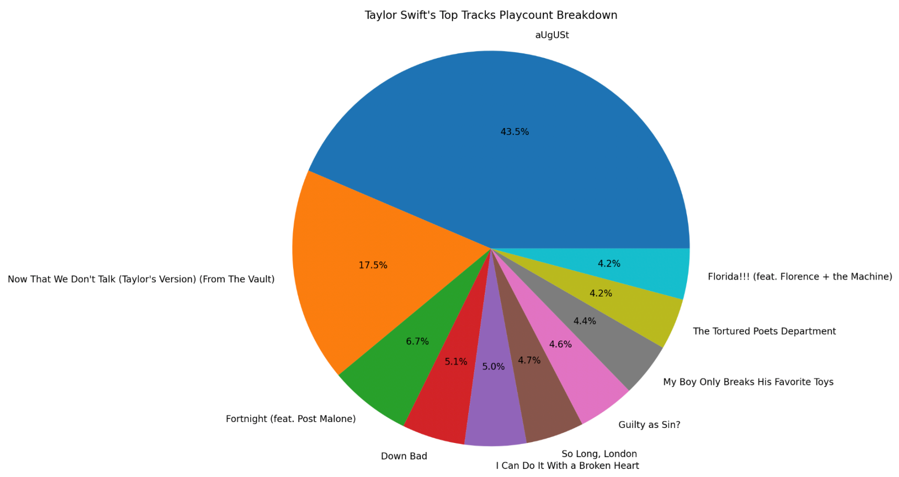
*NBA:*

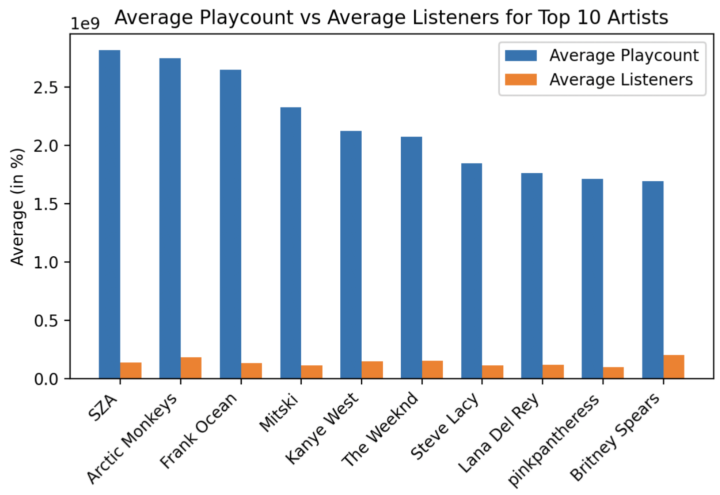


```
Average Salary: 22527831.52
Average Weight: 219.04
Average Height: 77.64
Average Experience: 12.48
Average Depth Chart Order: 1.52
Correlation between Salary and Experience: 0.18
```

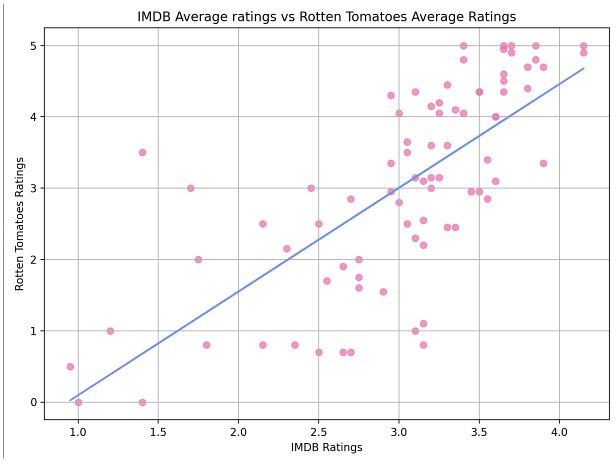
# Visualizations

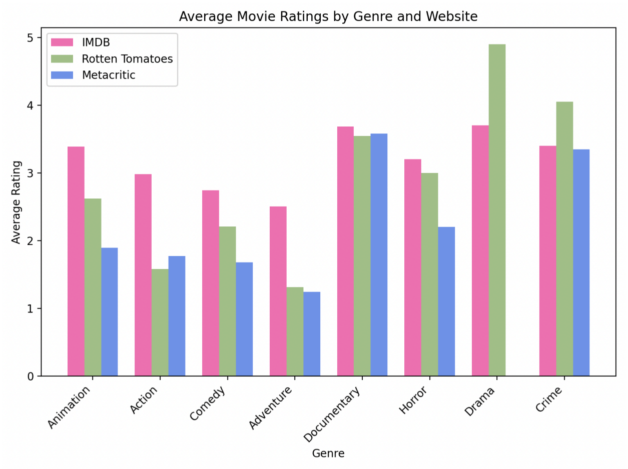
Last FM:



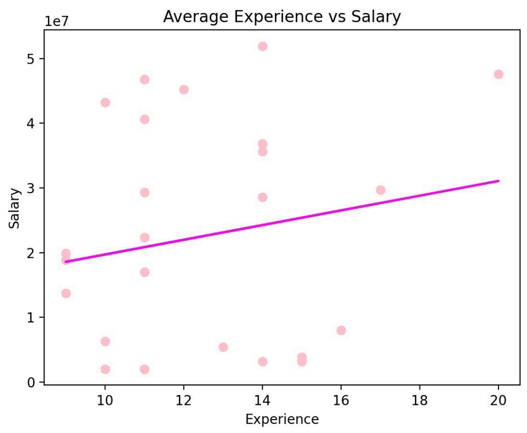


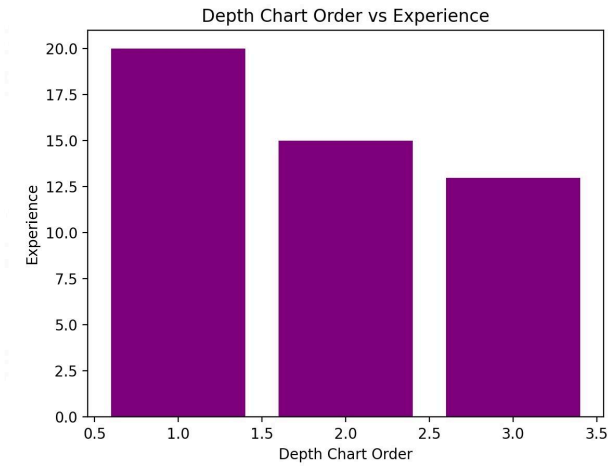
## OMDb:


IMDB Average ratings vs Rotten Tomatoes Average Ratings
Rotten Tomatoes Ratings
IMDB Ratings


Average Movie Ratings by Genre and Website
IMDB
Rotten Tomatoes
Metacritic
Average Rating
Animation
Action
Comedy
Adventure
Documentary
Horror
Drama
Crime
Genre

## NBA:


Average Experience vs Salary
1e7
Salary
Experience


Depth Chart Order vs Experience
Experience
Depth Chart Order

## Instructions For Running Code

1. **lastfmnew.py**
    a. Run this file 4 times, each time 25 rows will be added to the top_artists table.
2. **lastfm2.py**
    a. Run this file 4 times, each time 25 rows will be added to the top_tracks table.
3. **omdb.py**
    a. Run this file 4 times, each time 25 rows will be added to the Movie_Ratings table.
4. **nba.py**
    a. Run the file 4 times, each time, 25 rows will be added to the players table
5. **calculationslastfm.py**
    a. Run this file once, the calculations will be shown in a txt file, and the visualizations will appear. (Note: both lastfm files must be run 4 times before this file for the visualizations to accurately represent all the data).
6. **omdb_calculations.py**
    a. Run this file once, the calculations will be shown in a txt file, and the visualizations will appear. (Note: the omdb.py file must be run 4 times before this file for the visualizations to accurately represent all the data).
7. **nbacalculations.py**
    a. Run this file once, the calculations will be shown in a txt file, and the visualizations will appear. (Note: the nba.py file must be run 4 times before this file for the visualizations to accurately represent all the data).

## Function Documentation

*Last FM API*

- **lastfmnew.py - top artists**
    - 1. def read_api_key(file):
        - This function reads the api key from the api_key_lastfm.txt file. It takes in a string representing the file name from which the API key will be read and returns a string containing the API key read from the specified file, or None if the file is not found. It takes in a string representing the file name from which the API key will be read.
    - 2. def connect_to_database (database_name)
        - Input: database_name is a string that represents the name of the database we are connecting to
        - Output: It returns a tuple (conn, cur) where conn is a connection object to the database and cur is a cursor object
    - 3. def create_top_artists_table(cur)
        - Input: cur which is a cursor object connected to the SQLite database
        - Output: A table entitled "top_artists" in the database with the specified columns, if it doesn't already exist
    - 4. def fetch_top_artists(API_URL, params)
        - Input:
            - API_URL - a string that contains the URL for the Last FM API
            - Params - a dictionary with the specified parameters to make the API request
        - Output: A json object that holds the response data from the Last FM API
    - 5. def insert_top_artists (cur, artists_data)
        - Input:
            - Cur, which is a cursor object connected to the SQLite database
            - artist_data - a list of dictionaries that contains data about the top artists fetched from the API
        - Output: Inserts new artist data into the top_tracks table in the SQLite database, returns the total number of new artists inserted.
    - 6.  def main ()
        - Input: none
        - Calls all the functions, connects to the database, obtains data from API, prints necessary status messages, and inserts into database 25 at a time.

- **lastfm2.py - top tracks**
    - 1. def read_api_key(file):
        - This function reads the api key from the api_key_lastfm.txt file. It takes in a string representing the file name from which the API key will be read and returns a string containing the API key read from the specified file, or

None if the file is not found.  It takes in a string representing the file name from which the API key will be read.

- 2. def connect_to_database (database_name)
    - Input: database_name is a string that represents the name of the database we are connecting to
    - Output: It returns a tuple (conn, cur) where conn is a connection object to the database and cur is a cursor object
- 3. def create_top_tracks_table(cur)
    - Input: cur which is a cursor object connected to the SQLite database
    - Output: A table entitled “top_tracks” in the database with the specified columns, if it doesn’t already exist
- 4. def fetch_top_tracks (API_URL, params)
    - Input:
        - API_URL - a string that contains the URL for the Last FM API
        - Params - a dictionary with the specified parameters to make the API request
    - Output: A json object that holds the response data from the Last FM API
- 5. def insert_top_tracks (cur, tracks_data)
    - Input:
        - Cur, which is a cursor object connected to the SQLite database
        - tracks_data - a list of dictionaries that contains data about the top tracks fetched from the API
    - Output: Inserts new track data into the top_tracks table in the SQLite database, returns the total number of new tracks inserted.
- 6.  def main ()
    - Input: none
    - Calls all the functions, connects to the database, obtains data from API, prints necessary status messages, and inserts into database 25 at a time.

- **calculationslastfm.py**
    - 1. def calculate_average_playcount_top_tracks(cur)
        - Input: cur - cursor object connected to the SQLite database
        - Output: Returns the average play count calculated from the SQL query for the top tracks in the database as a floating point number
    - 2. def calculate_average_playcount_top_artist(cur)
        - Input:  cur - cursor object connected to the SQLite database
        - Output: Returns the average play count calculated from the SQL query for the top artists in the database as a floating point number
    - 3. def calculate_average_playcount_and_listeners_top_artists(cur)
        - Input: cur - cursor object connected to the SQLite database
        - Output: Returns top_artists_data, which is a list of tuples containing information about the top 10 artists (each tuple has the name of the artist,

        average playcount of the artist’s top tracks, average number of listeners of their top tracks)
    - 4. def calculate_percentage_playcount_taylor(cur)
        - Input: cur - cursor object connected to the SQLite database
        - Output: Returns percentage playcount, which is a list of tuples containing the name of each top track by Taylor Swift and its corresponding percentage of playcount relative to the total play count of all Taylor Swift's top tracks
    - 5. def write_calculated_data(average_playcount_top_tracks, average_playcount_top_artist, taylor_breakdown, top_artists_data)
        - Input: This function takes in the following input:
            - average_playcount_top_tracks: Average play count for top tracks,
            - average_playcount_top_artist: Average play count for top artists.
            - taylor_breakdown: Breakdown of play counts for Taylor Swift's top tracks.
            - top_artists_data: Data containing average play count and average listeners for the top 10 artists.
        - Output: This function does not return anything. Instead, it writes all of the calculations to a txt file
    - 6. def plot_taylor_top_tracks_playcounts(percentage_playcount)
        - Input: This function takes in percentage_playcount, a list of tuples containing the name of each top track by Taylor Swift and its corresponding percentage of play count relative to the total play count of all Taylor Swift's top tracks.
        - Output: This function creates a pie chart visualization showing the breakdown of play counts for Taylor Swift's top tracks. If there is no data available, it prints a message indicating that there is no data.
    - 7. def plot_average_playcount_vs_listeners(top_artists_data)
        - Input: Takes in top_artists_data, which is a list of tuples containing information about the top 10 artists (each tuple has the name of the artist, average playcount of the artist’s top tracks, average number of listeners of their top tracks).
        - Output: This function outputs a bar chart visualization comparing the average play counts and average listeners for the top 10 artists.
    - 8. def main():
        - Input: This function doesn’t take in any input
        - Output: It calls all of the other functions, establishes a connection to the SQLite database, and then executes analysis and visualization tasks based on the data retrieved from the database.

*OMDB API*

- **omdb.py - main py file**
    - 1. def read_api_key(file)
        - This function reads the api key from the api_key_lastfm.txt file. It takes in a string representing the file name from which the API key will be read

and returns a string containing the API key read from the specified file, or None if the file is not found. It takes in a string representing the file name from which the API key will be read.

- 2. def get_movie_titles(num_pages, year=None)
    - Input: Number of pages to fetch movie titles from as an int.
    - Output: It obtains the movies based on num_pages and year (optional). If a year is provided, it filters the movies by the specified year. It returns a list of movie titles or an error message if it is unable to get the data from the API.
- 3. def clean_movie_name(title)
    - Input: A title as a string.
    - Output: Strips the title and returns the cleaned movie name.
- 4. def get_all_ratings(title)
    - Input: The title of a movie as a string
    - Output: Uses the OMDb API to fetch the ratings of that movie from 3 movie rating sites (IMDb, Rotten Tomatoes, and Metacritic). This function uses the convert_to_decimal function to convert each rating to a float value. It returns the movie title, a dictionary where the keys are the movie rating websites (OMDb, Rotten Tomatoes, Metacritic) and the values are the ratings associated with each websitem and the movie genre. Or, it will return an error message if the movie is not found and a different error message if it is unable to access the OMDb API.
- 5. def create_database()
    - Input: This function does not take in anything.
    - Output: It creates a SQLite database and creates 2 tables: one for the Genres (to take care of duplicate string data) and one for the movie ratings.
- 6. def add_movie_ratings(title, ratings, genres)
    - Input: This function takes in a title, a dictionary of ratings, a string of genres.
    - Output: This function inserts the genres into the Genres table in the database and it inserts the move data in the Movie_Ratings function. It returns nothing.
- 7. def get_genres()
    - Input: This function does not take in anything.
    - Output: This function uses a select statement to join the Movie_Ratings table to the Genres table on the genre_id. It returns the Genres.
- 8. def main()
    - Input: This function does not take in anything.
    - Output: Calls all the functions, connects to the database, obtains data from API,  prints necessary status messages, and inserts into database 25 at a time. It does not return anything.

- **omdb_calculations.py - calculations py file**
  - 1. def calculate_average_rating_imdb(cur):
    - Input: Cur, which is a cursor object connected to the SQlite database
    - Output: It uses an execute statement to get the average IMDb ratings from the Movie_Ratings table and returns the average rating value as a float.
  - 2. def calculate_average_rating_rotten_tomatoes(cur)
    - Input: Cur, which is a cursor object connected to the SQlite database
    - Output: It uses an execute statement to get the average Rotten Tomatoes ratings from the Movie_Ratings table and returns the average rating value as a float.
  - 3. def calculate_average_rating_metacritic(cur)
    - Input: Cur, which is a cursor object connected to the SQlite database
    - Output: It uses an execute statement to get the average Metacritic ratings from the Movie_Ratings table and returns the average rating value as a float.
  - 4. def calculate_percentage_rating_above_two_imdb(cur)
    - Input: Cur, which is a cursor object connected to the SQlite database
    - Output: It uses an execute statement to get the amount of IMDb ratings that are above 2 and returns the percentage of IMDb ratings that are above 2.
  - 5. def calculate_percentage_rating_above_two_metacritic(cur)
    - Input: Cur, which is a cursor object connected to the SQlite database
    - Output: It uses an execute statement to get the amount of Metacritic ratings that are above 2 and returns the percentage of Metacritic ratings that are above 2.
  - 6. def write_calculated_data(average_imdb_ratings, average_rotten_tom_ratings, average_metacritic_ratings, count_ratings_above_two_imdb, count_ratings_above_two_metacritic)
    - Input: The average IMDb rating, the average Rotten Tomatoes rating, the average Metacritic rating, the percentage of IMDb rating above 2, the percentage of Metacritic ratings above 2
    - Output: Writes each calculation to a txt file, doesn't return anything.
  - 7. def fetch_movie_ratings(cur)
    - Input: Cur, which is a cursor object connected to the SQlite database.
    - Output: Returns all the movie ratings from all 3 websites (OMDb, Rotten Tomatoes, Metacritic).
  - 8. def calculate_regression_line(x, y)
    - Input: List of rating values from the chosen rating websites (x is one website and y is the other).
    - Output: Returns the coefficients of the linear regression line as a NumPy array.
  - 9. def scatter_plot(imdb_ratings, rotten_tomatoes_ratings)
    - Input: A list of the IMdb ratings and a list of the Rotten Tomatoes ratings.

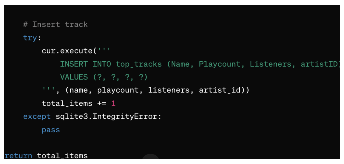
        - Output: It creates a scatterplot of the average IMDb ratings vs the average Rotten Tomatoes Ratings where each dot on the plot represents a different movie. It also displays a regression line. This function does not return anything.
    - 10. def fetch_ratings_genres_and_names(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: Fetches and returns movie ratings by genre and by movie rating website.
    - 11. def calculate_average_ratings_by_genre(ratings_genres_and_names)
        - Input: Movie ratings by genre and by movie rating website.
        - Output: Finds the average movie rating by genre in the form of tuples and returns it.
    - 12. def grouped_bar_chart(avg_ratings_by_genre):
        - Input: Average movie rating by genre in the form of tuples.
        - Output: It creates a grouped bar chart where genre is on the x axis and there are 3 bars for each genre displaying the average movie rating of that genre for that particular movie rating website (OMDb, Rotten Tomatoes, Metacritic).
    - 13. def main()
        - Input: This function does not take in anything.
        - Output: Sets the average movie ratings to variables and sets the percentage of ratings above 2 to variables. Calls the write function so the data is written to the txt file, and calls the scatterplot function and the grouped bar chart function so they are displayed. It does not return anything.

*NBA API*

- **nba.py - main py file**
    - 1. def gather_player_data(cur, conn)
        - Input: takes in the connection to the database
        - Output: reads the API key, creates a table, inserts a unique set of players and their data by 25, doesn't return anything
    - 2. def main()
        - Input: None
        - Output: Connects to the database, calls gather_player_data and allows the data to be inserted into the database, doesn't return anything, then closes the connection
- **nbacalculations.py - calculation py file**
    - 1. def get_salary_avg(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: avg, average salary
    - 2. def get_weight_avg(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: avg_weight, average weight
    - 3. def get_height_avg(cur)

        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: avg_height, average height
    - 4. def get_avg_experience(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: avg_experience, average experience
    - 5. def_avg_dco(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: avg_dco, average depth chart order
    - 6. def get_sal_exp_corr(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: corr_coef, the correlation coefficient between salary and experience
    - 7. def exp_vs_sal(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: doesn't return anything, but codes for a scatter plot with a regression line through it, demonstrating the correlation between salary and experience.
    - 8. def dco_vs_exp(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: doesn't return anything, but codes for a bar graph that shows relationships between experience and depth chart order
    - 9. def writecalcs(cur)
        - Input: Cur, which is a cursor object connected to the SQlite database.
        - Output: doesn't return anything, but writes the calculations from previous functions into another txt file
    - 10. def main()
        - No input
        - Output: doesn't return anything, but establishes connection to the database, calls the functions that show visualizations, calls the function that writes the calculations into a txt file

## Resources

| Date | Issue Description | Location of Resource | Result |
|---|---|---|---|
| 04/14/2024 | Having trouble understanding how api keys work and how to obtain data from APIs | HW 6 Files | Was able to use the format from this homework in the files since they all utilized API keys |
| 04/18/2024 | I was not sure of how SQL join statements worked and how to use them in my calculations - was thinking it was joining two tables fully as opposed to rows based on columns | W3Schools SQL Join | I used this to understand the different types of join statements and what kinds of JOINS could be performed within my calculations |
| 04/20/2024 | Was wondering why the database was not returning the same artists rather than different ones each time |  `# Insert track`<br>`try:`<br>`    cur.execute('''`<br>`        INSERT INTO top_tracks (Name, Playcount, Listeners, artistID`<br>`        VALUES (?, ?, ?, ?)`<br>`    ''', (name, playcount, listeners, artist_id))`<br>`    total_items += 1`<br>`except sqlite3.IntegrityError:`<br>`    pass`<br>`return total_items`<br>ChatGPT | Added this try statement in order to see what was already in the table and what should be ignored to ensure that different artists were added each time |
| 04/30/2024 | I was having trouble getting my graphs to show up, and was having trouble making them turn out correctly when they did. | Lecture 21 | I used similar formats to examples used in lecture to assist me in making my visualizations, especially for specific commands. |