SI 206
Final Project Report
30 April 2024

Jane Joyce, Bridget Bailey, Natasha Milosavljevic

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

Our original plan for the project was to use the Spotify, Billboard, and Bandsintown APIs to collect data for our project. From Billboard's API, we wanted to collect data on the artist names of the Top 10 Albums of the year from the past decade. From Spotify's API, we planned on collecting these artists' full names and top performing albums. Lastly, from Bandsintown, we planned on collecting the number of upcoming shows for these artists. Our original goal was to analyze the correlation between an artist's amount of upcoming concerts and their rank on Billboard Top 100 Artists. However, this plan did not end up being realistic as we were unable to get access to the Bandsintown API, so we had to pivot our project concept while staying true to our original focus area, which is the music industry.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

For our final project concept, we ended up using the Spotify/Spotipy API, the Billboard API, and used web scraping to collect data on the 2023 Grammy winners and nominees from Variety's website. We called the Billboard API to collect data on Billboard's Top 100 artists from the week of May 1st, 2023. Utilizing this data on the artist names from Billboard's Top 100 list, we used Spotify's API and the Spotipy package for python to collect data on each artist's follower count and popularity index from Spotify. Then we utilized web scraping and Beautiful Soup to scrape data on the 2023 Grammy nominees and winners from Variety's article on the event. We collected data on each nominee's number of nominations and wins from the 2023 Grammys. Using all of this collective data, our goal was to analyze whether or not there was a relationship between an artist's documented popularity on Spotify and their success at the 2023 Grammy Awards.

3. The problems that you faced (10 points)

One big challenge that we faced was coming up with a reliable way to execute the main() functions of our table files four times in order to ensure that only 25 items were entered into a table at one time while still accumulating around 100 items in the tables. We tried and tested a variety of different methods to try to piece together this practice. We found that the most reliable and consistent way to do so would be to use a “start id” text file, which essentially tracks the amount of items that are entered into a table after each file execution. We included get_start_id() and update_start_id(start_id) functions in each table file in order to have the given start_id begin at 0 and be updated by a maximum of 25 after each execution. Then, we created three execution files for each table that called the main() function again, allowing us to add up to 100 items of data into the table after executing main() four times.

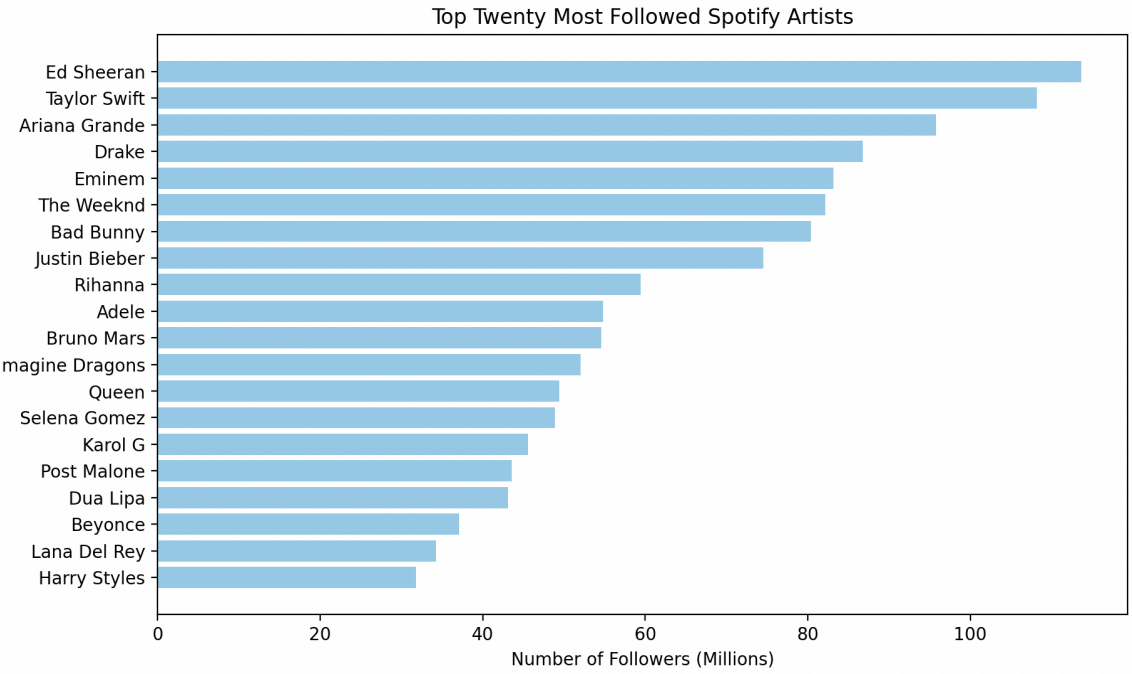
Another problem that we encountered was the fragile nature of working in a shared repository. It became clear to us from the very beginning that it would be essential for us to communicate clearly about who was working on which files in order to avoid merge issues. We would communicate frequently to let one another know when we were working on the project in VS code, and warned one another to avoid making changes to specific files when we were at work. We were diligent about executing git pulls before we started working, and consistently committed and pushed our changes at the end of a work session. We wanted to ensure that our work was not lost in the process of updating the shared repository.

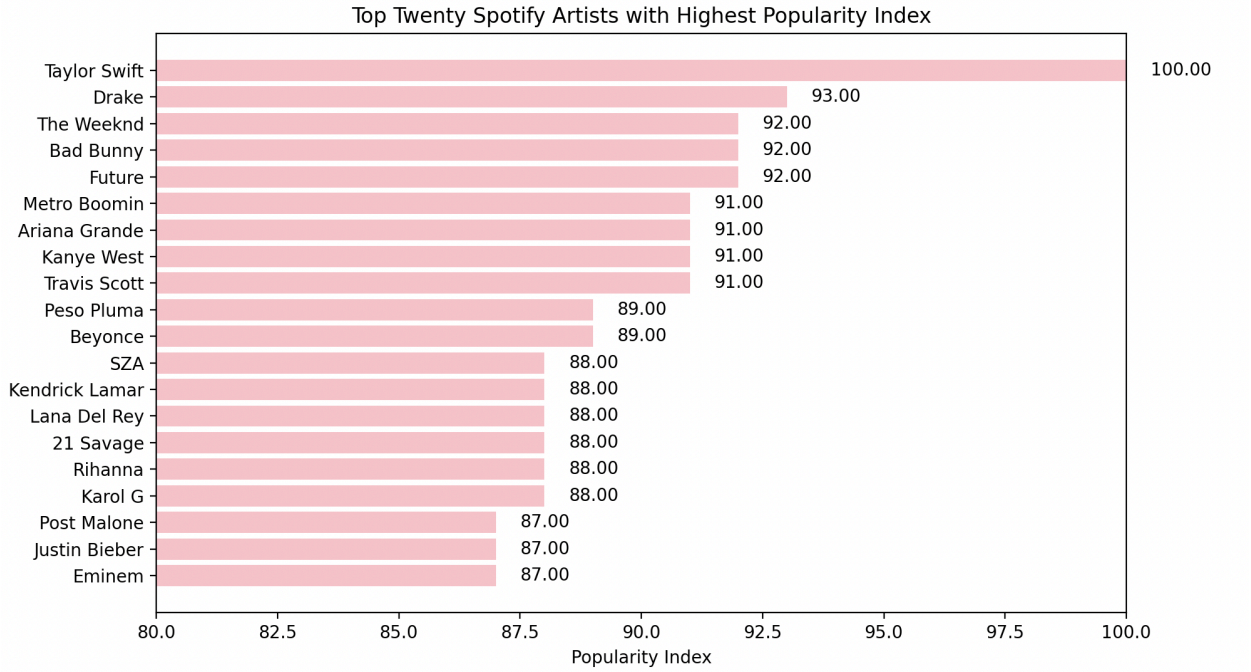
4. The calculations from the data in the database (i.e. a screen shot) (10 points)

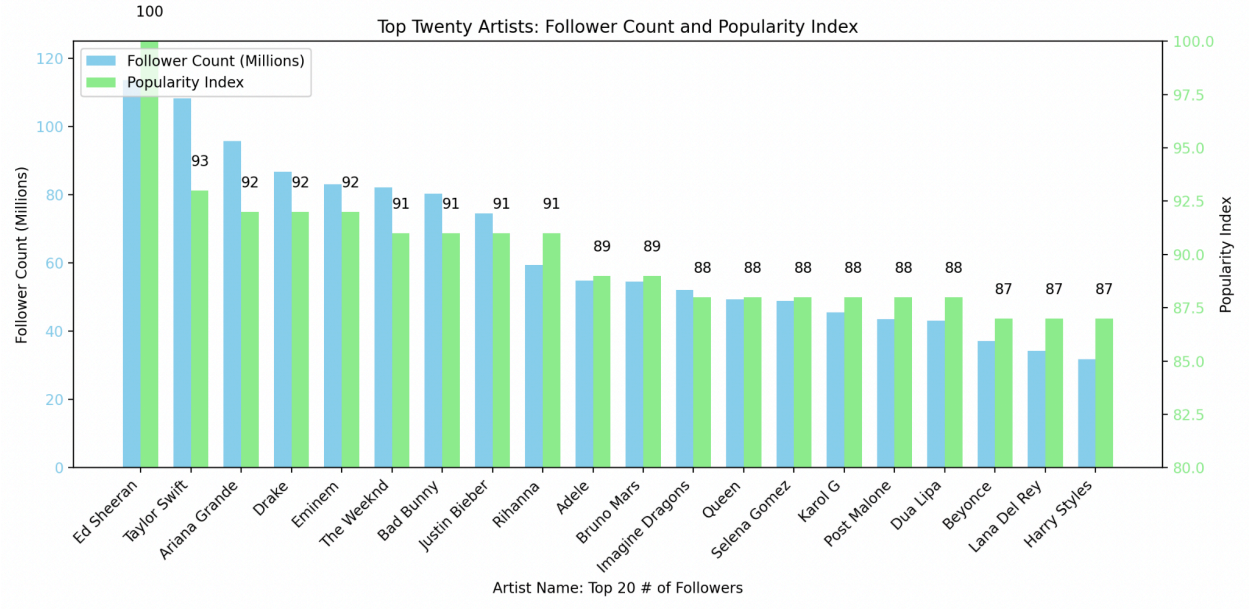
final_calculations.txt

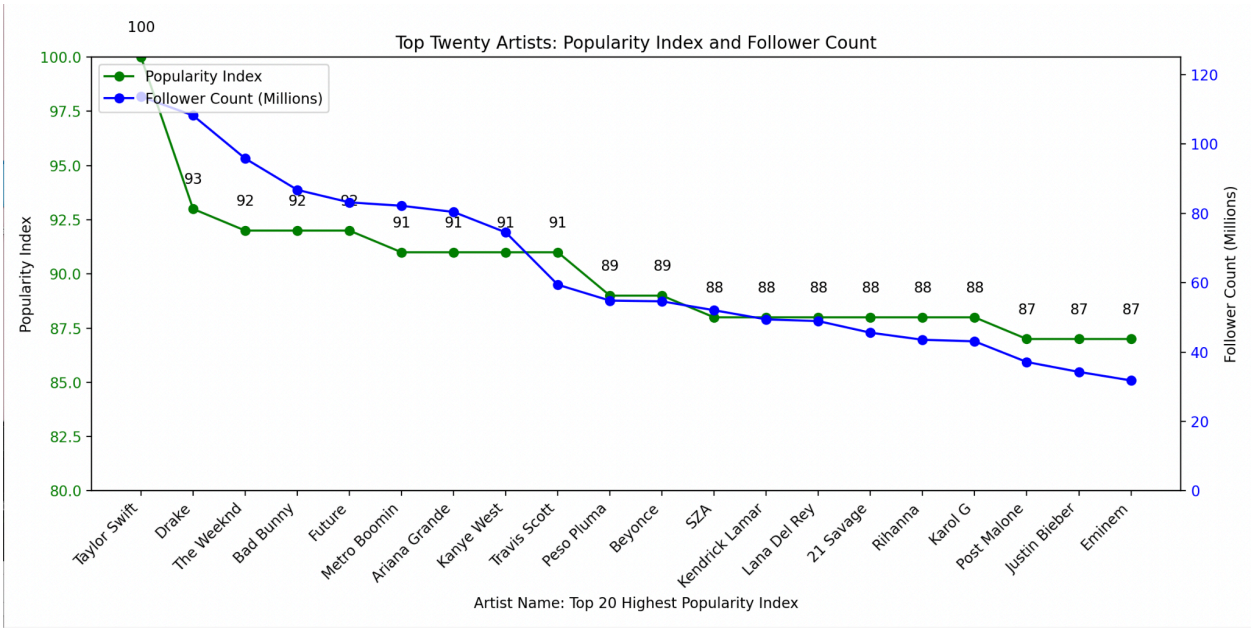
```
Most Popular Artists on Spotify:

AVG NUM OF FOLLOWERS = 21,182,451.90
AVG POPULARITY INDEX = 78.08
AVG GRAMMY NOMINATIONS = 0.37

Top Ten Nominations:
1. Artist ID: 38, Nominations: 6
2. Artist ID: 14, Nominations: 5
3. Artist ID: 15, Nominations: 4
4. Artist ID: 77, Nominations: 4
5. Artist ID: 82, Nominations: 4
6. Artist ID: 13, Nominations: 3
7. Artist ID: 65, Nominations: 3
8. Artist ID: 91, Nominations: 2
9. Artist ID: 0, Nominations: 1
10. Artist ID: 3, Nominations: 1

AVG GRAMMY NOMINATION FOR TOP TEN ARTISTS: 3.3

```

5. The visualization that you created (i.e. screen shot or image file) (10 points)




Top Twenty Spotify Artists with Highest Popularity Index
Taylor Swift
Drake
The Weeknd
Bad Bunny
Future
Metro Boomin
Ariana Grande
Kanye West
Travis Scott
Peso Pluma
Beyonce
SZA
Kendrick Lamar
Lana Del Rey
21 Savage
Rihanna
Karol G
Post Malone
Justin Bieber
Eminem
100.00
93.00
92.00
92.00
92.00
91.00
91.00
91.00
91.00
89.00
89.00
88.00
88.00
88.00
88.00
88.00
88.00
87.00
87.00
87.00
80.0
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
Popularity Index


Top Twenty Artists: Follower Count and Popularity Index
Follower Count (Millions)
Popularity Index
Follower Count (Millions)
Popularity Index
Artist Name: Top 20 # of Followers
Ed Sheeran
Taylor Swift
Ariana Grande
Drake
Eminem
The Weeknd
Bad Bunny
Justin Bieber
Rihanna
Adele
Bruno Mars
Imagine Dragons
Queen
Selena Gomez
Karol G
Post Malone
Dua Lipa
Beyonce
Lana Del Rey
Harry Styles
100
93
92
92
92
91
91
91
91
89
89
88
88
88
88
88
88
87
87
87



6. Instructions for running your code (10 points)

1. Delete the file 'artist.db' in order to run the code again
2. Run the file "billboard_read_api.py"
3. Run the file "spotify_read_api.py"
4. Run the file "grammy.py"
5. Run the file "create_database.py"
6. Ensure that the text file "start_id.txt" contains "0" as a string
7. Run the file "create_artist_table.py"
8. Run the files in the following order:
    a. "artist_table_execution_2.py"
    b. "artist_table_execution_3.py"
    c. "artist_table_execution_4.py"
9. Ensure that the text file "grammy_start_id.txt" contains "0" as a string
10. Run the file "create_grammy_table.py"
11. Run the files in the following order:
    a. "Grammy_table_execution.py"
    b. "Grammy_table_execution_2.py"
    c. "grammy_table_execution_3.py"
12. Ensure that the text file "spotify_start_id.txt" contains "0" as a string
13. Run the file "create_spotify_table.py"
14. Run the files in the following order:
    a. "spotify_table_execution.py"

    b. “spotify_table_execution_2.py”
    c. “spotify_table_execution_3.py”
15. Ensure that the text file “billboard_grammy_start_id.txt” contains “0” as a string
16. Run the file “create_grammy_billboard_table.py”
17. Run the files in the following order:
    a. “NEW_grammy_table.py”
    b. “NEW_grammy_table_2.py”
    c. “NEW_grammy_table_3.py”
18. Run “calculations.py”
19. View artist.db that was created
20. View “final_calculations.txt”

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

Function Documentation

Billboard_read_api.py

- **Get_rank_artist_data(date)-** takes in a date, in form “YYYY-MM-DD”, as input and calls the Billboard API and returns a nested dictionary where the keys are the artist names from the Billboard Top 100 Artists of that given week and the values are dictionaries of information about the artist. If the function does not run properly, the function eventually returns “Billboard Top Artist Chart Not Found” from the get_rank_artist_data(date) function from billboard_read_api.py
- **Get_artist_list(date)-** takes in a date, in form “YYYY-MM-DD”, as input and uses the output of get_artist_rank_data to iterate through dictionary keys in order to return a list of the names of the artists on Billboard’s Top 100 list for that given date.

Spotify_read_api.py

- **Spotify_artist_information(date)-** takes in a date, in form “YYYY-MM-DD”, as input and calls **get_artist_list(date)** from billboard_read_api.py to get a list of the artist names from Billboard’s Top 100 for that given week. It then calls Spotify’s API and uses Spotipy to iterate through the list of artist names to return a nested dictionary. The keys are the names of the artists and the values are dictionaries that contain the artist’s follower count and popularity index: artist_popularity_dict[artist] = {'followers': followers, 'popularity': popularity_index}

Create_database.py

- **create_database(db_name)-** takes in the name of a database in string form (ie. “artist.db”) and creates an SQL database with that specified name. It returns cur and conn, which are the cursor object and the connection object for the database, respectively.

Create_artist_table.py

- **get_start_id()-** creates a text file called 'start_id.txt". This file will contain a number that keeps track of the number of times the contents of the create_artist_table.py file is executed. Returns an integer of first line of "start_id.txt"
- **create_artist_table(cur, conn, artist_list, start_id)-** takes in the SQL cursor and the Sqlite3 connection with a database file as input. Also takes in artist_list, a list of artist names from billboard_read_api.py's **get_artist_list(date)** function as an input. The last argument, start_id, refers to the output of **get_start_id()**, so start_id = get_start_id(). Returns none, but creates a table "Artist" in "artist.db". Because this function uses a range loop to limit the amount of data stored into the Artist table, it will return "Billboard Top Artist Chart Not Found" since it calls on
- **update_start_id(start_id)-** The argument, start_id, refers to the output of **get_start_id(),** so start_id = get_start_id(). This function rewrites "start_id.txt" to contain the newly updated start_id (which is updated in create_artist_table function). Returns none.

Create_spotify_table.py

- add_spotify_info(cur, conn, start_id)- takes in the SQL cursor and the Sqlite3 connection with a database file as input. The argument, start_id, refers to the output of **get_start_id()**, so start_id = get_start_id(). This function creates a table called Spotify_followers and uses a range loop to call on spotify_artist_information(date) from Spotify_read_api.py, entering a unique spotify artist ID and the following count of that respective artist into the table. Returns None.
- create_spotify_popularity_table(cur, conn, start_id)- takes in the SQL cursor and the Sqlite3 connection with a database file as input. The argument, start_id, refers to the output of **get_start_id()**, so start_id = get_start_id(). This function creates a table called Spotify_popularity and uses a range loop to call on spotify_artist_information(date) from Spotify_read_api.py, entering a unique spotify artist ID, each artist's name, and that artist's popularity index into the table. Returns None.
- **update_start_id(start_id)-** The argument, start_id, refers to the output of get_start_id(), so start_id = get_start_id(). This function rewrites "spotify_start_id.txt" to contain the newly updated start_id (which is updated in create_artist_table function). Returns None.
- **get_start_id()-** creates a text file called 'spotify_start_id.txt". This file will contain a number that keeps track of the number of times the contents of the create_artist_table.py file is executed. Returns an integer of first line of "start_id.txt"

Grammy.py

- **retrieve_listings()** - This sends a request to https://www.vulture.com/2023/02/grammys-2023-full-list-of-winners.html, and parses through the HTML content using beautiful soup. This function returns award_dic, where

the keys are the 2023 Grammy awards, and the corresponding value is a list of all that awards nominations.

- **get_winners()** - This function parses through the same HTML content using beautiful soup. This function returns winners_dic, where the keys are the 2023 Grammy awards, and the value is a string containing the winner of that corresponding award.
- **get_info_about_artist (artist)** - This function takes in an artist as the argument, which is defined in main() and can be altered in main(), and returns None. Instead, it prints out a series of statements that provide information about what awards that artist was nominated for, and what awards they won (if any). This was used for clarification purposes.

create_grammy_table.py

- **create_grammy_table(cur, conn, start_id, listing_data, winners_data, artist_list)** - This function takes in the start_id, which is currently set at 0, listing_data, which is the dictionary returned by **retrieve_listings()** in **Grammy.py,** winners_data which is the dictionary returned by **get_winners()** in **Grammy.py,** and artist_list, which is the list returned by **get_artist_list().** It also takes in the SQL cursor and the Sqlite3 connection with a database file as input. This function creates two tables, and iterates through listing_data and winners_data to create a dictionary where the key is the artist, and the value is a list with 2 dictionaries that contain the musical piece they were nominated ('noms') for and the awards they won, if they won any ('winner'). It then inserts this data into the two tables, Grammy_artists, Grammy_awards, . Grammy_artists has an id key with every artist nominated for a Grammy. Grammy_awards has a key id with every award possible at the 2023 Grammys (91 total awards). Returns None.
- **update_start_id(start_id)**- The argument, start_id, refers to the output of get_start_id(), so start_id = get_start_id(). This function rewrites "grammy_start_id.txt" to contain the newly updated start_id (which is updated in create_artist_table function). Returns None.
- **get_start_id()**- creates a text file called 'grammy_start_id.txt". This file will contain a number that keeps track of the number of times the contents of the create_artist_table.py file is executed. Returns an integer of first line of "start_id.txt"

create_billboard_grammy_table.py:

- **create_grammy_billboard_table(cur, conn, start_id)** - takes in the SQL cursor and the Sqlite3 connection with a database file as input. The argument, start_id, refers to the output of **get_start_id()**, so start_id = get_start_id(). This function creates a table called Grammy_billboard_artists in the artist database. Then, it iterates through the dictionary returned by **retrieve_listings()** in the grammy.py file, creating a new dictionary called artist_nominations where each individual artist name is a key and the value is the number of nominations earned by the artist at the 2023 Grammys. Then, the function iterates through the list of artist names returned by **get_artist_list(date)**, using .get() on the artist_nominations dictionary to identify the amount of nominations earned by Billboard's

Top 100 artists. The function inserts each of these artists and their amount of nominations into the table. This function returns None.

- **update_start_id(start_id)-** The argument, start_id, refers to the output of get_start_id(), so start_id = get_start_id(). This function rewrites "billboard_grammy_start_id.txt" to contain the newly updated start_id (which is updated in create_artist_table function). Returns None.
- **get_start_id()-** creates a text file called 'billboard_grammy_start_id.txt". This file will contain a number that keeps track of the number of times the contents of the create_artist_table.py file is executed. Returns an integer of first line of "start_id.txt"

Calculations.py:

- **calculate_average_follower_count(conn, cursor):** Calculates the average follower count of artists from the Spotify_followers table. Parameters: conn, the SQLite database connection object and cursor, the SQLite cursor object. Returns average_follower_count: Average follower count of artists.
- **calculate_average_popularity_index(conn, cursor):** Calculates the average popularity index of artists from the Spotify_popularity table. Parameters: conn, cursor. Returns: average_pop_index: Average popularity index of artists.
- **calculate_average_grammy_nomination(conn, cursor):** Calculate the average number of Grammy nominations from the Grammy_billboard_artists table. Parameters: conn and cursor. The function returns the average_grammy_nom, which is the average number of Grammy nominations.
- **find_top_ten(conn, cursor):** This function retrieves the top ten artist IDs with the highest number of Grammy nominations from the Grammy_billboard_artists table. Parameters: conn and cursor. The return value is top_twenty_nominations, which is a list of top ten artist IDs with their corresponding nominations.
- **join_tables(conn, cursor):** This function joins the Spotify_followers, Spotify_popularity, and artist tables to retrieve combined data. Parameters: conn and cursor. The return value is joined_data, which contains combined data from all joined tables.
- **top_twenty_followers(conn, cursor):** This function retrieves data for the top twenty artists with the highest number of followers. Parameters: conn and cursor .The return value is top_twenty_data, which contains data for the top twenty artists with follower counts.
- **top_twenty_popularity(conn, cursor):** This function retrieves data for the top twenty artists with the highest popularity index. Parameters: conn and cursor. The return value is top_twenty_data, which contains data for the top twenty artists with popularity index.
- **plot_bar_chart_followers(data):** This function plots a bar chart for the top twenty most followed Spotify artists. It takes one parameter: data, which contains artist names and follower counts.

- **plot_bar_chart_popularity(data):** This function plots a bar chart for the top twenty Spotify artists with the highest popularity index. It requires one parameter: data, which contains artist names and popularity index.
- **plot_dual_bar_graph(follower_data, popularity_data):** This function plots a dual bar graph for the top twenty artists with follower count and popularity index. It takes two parameters: follower_data and popularity_data, containing artist names and corresponding data.
- **plot_dual_bar_graph_popularity(popularity_data, follower_data):** This function plots a dual line graph for the top twenty artists with popularity index and follower count. It requires two parameters: popularity_data and follower_data, containing artist names and corresponding data.
- **main(date):** This is the main function to process data and visualize it. It takes a date parameter (not used in the function, just a placeholder).

21. You must also clearly document all resources you used. The documentation should be of the following form: (20 points)

Date	**Issue Description**	**Location of Resource**	**Result (did it solve the issue)**
4/19/24	In Grammy.py, I needed to implement measures after a request to see if the request was successful or not, since occasionally code was not working.	https://realpython.com/python-requests/#:~:text=.status_code%20returned%20200%20%2C%20which%20means,data%20that%20you%20were%20requesting.	Yes, it ensured that if the request was not successful, nothing would be returned, and it would print out that the request was unsuccessful
4/10/24	In spotify_read_api.py, I was having trouble configuring my code in order to correctly use spotipy to call the Spotify API.	https://spotipy.readthedocs.io/en/2.22.1/	Yes, this resource clearly laid out the steps for accessing Spotify's API through the spotipy package for python, and it provided me with clear documentation of the API.

4/13/24	In create_grammy_billboard_table.py, I was having trouble properly indexing on the nested dictionary of grammy nominees/winners. I was also having trouble isolating winner/nominee names.	https://stackoverflow.com/questions/53164633/indexing-a-value-in-a-nested-dictionary	This resource helped me clean up my code and more properly define artist names from the large amount of information scraped from the Variety website on the Grammy winners/nominees.
4/10/23	In spotify_read_api.py, I was having trouble configuring my code in order to correctly use spotipy to call the Spotify API.	https://medium.com/@maxtingle/getting-started-with-spotifys-api-spotipy-197c3dc6353b	This resource acted as another model for utilizing spotipy to call on Spotify's API. This resource clearly provided visuals for where I could find important information and necessary Client IDs on the Spotify for Developers website, which was extremely helpful.
4/23/24	In create_grammy_table.py, we were running into the problem that when trying to split a string with the nominated piece of music and artist that composed it, it was splitting on newlines despite having a "-" split. Additionally, there are some nominations that have multiple artists responsible for one song, and was having difficulty separating these artists as well	Office hours	Yes, we were able to properly split the strings and avoid splitting on the newlines, and during office hours, we created a new dictionary (different from the ones created in grammy.py) that made it music easier to correctly iterate through and grab each artist, what they were nominated for, and what they won.

4/26/24	Trouble writing the query for retrieving data about a certain number of artists	https://stackoverflow.com/questions/41219058/how-to-write-queries-in-python-script	Yes, got to make clear statements and learned how to adjust because of the changes going through debating displaying top ten or top twenty in the function join_tables, top_twenty_followers, and top_twenty_popularity.
4/27/24	Couldnt figure out the line to make a single bar chart into a double	https://www.geeksforgeeks.org/plotting-multiple-bar-charts-using-matplotlib-in-python/	Yes, the third visualization turned out clean and easy to understand. Also helped with the last line graph because I just had to make tiny changes to complete it.
4/27/24	Debugging throughout when given unfamiliar errors	Chat gpt	Would give it my line of code and then the error I was getting when I could not seem to fix, helped most of the time to fix things that I overlooked

Github repository link - https://github.com/janejoycee/206_final_project