SI 201-Final Project Report

Team Musical.ly: Olivia Lombardo, Abbey Halabis, Audrey Kovtun

1. Desired Goals

The primary goal of this project was to investigate how musical popularity differs across platforms by combining scraped chart data with API data from music platforms. To accomplish this, the project aimed to gather data from **three independent data sources**, each with its own base URL:

1. **Billboard Hot 100 (BeautifulSoup web scraping)**
    - *Purpose:* Provide a real-world snapshot of weekly chart performance.
    - *Goal:* Scrape 100 songs: including title, artist, and chart rank, and store them in a normalized SQLite database.

2. **Spotify API (Spotipy)**
    - *Purpose:* Capture streaming-based audio metadata.
    - *Goal:* For each Billboard song, retrieve and store Spotify attributes such as track popularity, duration, and track ID directly into the songs table.

3. **Last.fm API**
    - *Purpose:* Measure listener engagement across a different streaming ecosystem.
    - *Goal:* Retrieve track-level listener counts and playcounts for all 100 Billboard songs. A dedicated Lastfm_track_stats table was created, populated in batches of 25 rows per run to satisfy the assignment requirement.

Together, these datasets enable a comprehensive comparison of **chart popularity (Billboard)**, **streaming popularity (Spotify)**, and **listener engagement (Last.fm)**.
The overall goal was to build a fully integrated music analytics database capable of supporting comparisons across platforms and answering questions such as:

- *How does a song's Billboard rank relate to its Spotify popularity?*
- *Do the most-streamed artists also have the most Last.fm listeners?*
- *Are certain genres more popular across all platforms?*

This project integrates multiple APIs and web scraping into a single relational database and prepares the data for calculated metrics and visualizations.

2. Achieved Goals

Most of our primary goals of the project were successfully achieved through the combination of web scraping, multiple APIs, and a database with several tables. The project effectively worked with three independent data sources, which included the Billboard Hot 100 website, the Spotify API (spotipy), and the Last.fm API, to collect, store, and relate music popularity data across platforms.

- APIs/Websites Specifics:
    - Billboard Hot 100: This website was successfully scraped to collect a complete snapshot of weekly chart performance. For each of the 100 songs on the chart, the code gathered:
        - Rank, title, artist
        - This data was stored in a normalized SQLite database and served as the foundation for all subsequent API calls.
    - Spotipy API: using the Spotipy package, each Billboard song was matched to its corresponding Spotify track. For matched songs, the following attributes were retrieved and stored:
        - Song name, Artist name, Album name, Spotify album ID, song ID, song popularity, song duration, explicitly flag, release year
        - This fulfilled the goal of capturing streaming-based metadata and enabling direct comparisons between Billboard chart rank and Spotify popularity metrics.
    - Last.fm API: this was used to collect listener engagement statistics for the same set of Billboard songs. A dedicated table named Lastfm_track_stats was created to store:
        - Listeners, playcount
        - The data was inserted incrementally in batches of 25 songs per run, resulting in a complete dataset of 100 unique tracks after multiple iterations.
- Overall Outcome:
    - By combining scraped chart data with Spotify and Last.fm API data, the project successfully created a fully functional music analytics database. The supports cross-platform comparisons and enables meaningful analysis of questions such as:
        - The relationship between Billboard rank and Spotify popularity
        - Differences between streaming popularity and listener engagement
        - Artist and album-level trends across multiple platforms

3. Problems/Challenges Faced

(1) Fixing the 25-entry limitation:

The project requires that each execution of our data-gathering script insert no more than 25 items, while still accumulating at least 100 unique rows across multiple runs.

Our initial approach always selected the first 25 songs from Billboard's Hot 100 list, which caused the same items to be re-inserted on every run. This duplicated data and prevented us from continuing to the remaining songs in the chart.

To fix this, we implemented incremental insertion with duplicate detection:

1. Scrape all 100 Billboard songs each time the program runs.
2. Query the existing database to retrieve all stored (title, artist) pairs.
3. Filter out any songs that already exist in the database.
4. Select only the next 25 *new* songs from the remaining dataset.
5. Insert those 25 new entries into the database.

This allows the script to progress through the full Billboard Hot 100 list over multiple runs without ever modifying the code. Each execution adds exactly 25 new, unique songs until the full dataset has been stored.

(2) <u>Ensuring all artist names were in a usable format</u>

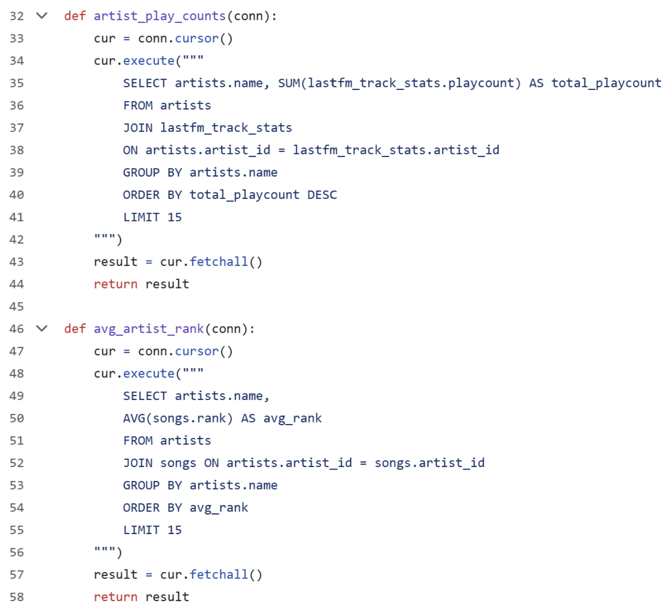
In order to incorporate the other two APIs after scraping the Billboard 100, it was necessary that all artist names were in a usable format to query other API searches. Many of the songs on the Hot 100 had multiple artists or were in a variety of formats. For example:

- Some were in “a” tags, while others were not
- HUNTR/X: EJAE, Audrey Nuna & REI AMI
- Morgan Wallen Featuring Tate McRae
- Mariah the Scientist & Kali Uchis

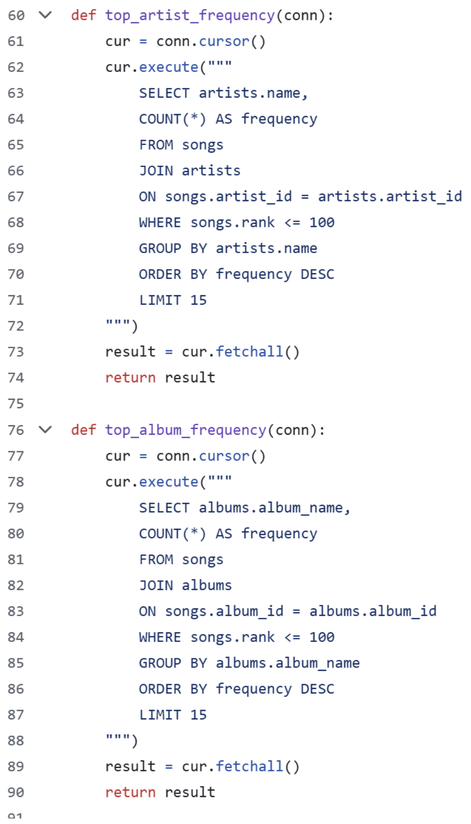
To fix this, we implemented several helper functions at the top of our code to clean and format all the artist names in matching formats. To do this, we utilized regular expressions. This allows the code to process all the names and output the primary artist on a track. This facilitates the rest of the project since the other API calls require an artist's name, and it was not feasible to search for two artists at the same time with words in between.

4. Calculations

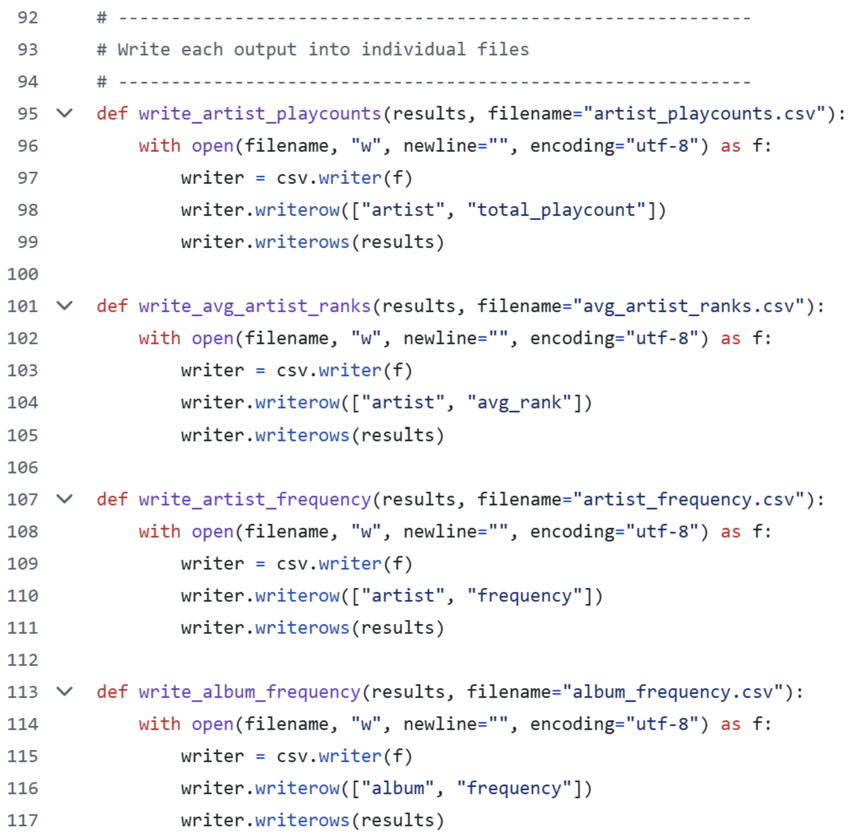
```python
# calculations
import sqlite3
import csv

def calculate_avg_duration(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT AVG(spotify_duration_ms)
        FROM songs
                """)
    result = cur.fetchone()[0]
    return result

def calculate_avg_plays(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT AVG(playcount)
        FROM lastfm_track_stats
                """)
    result = cur.fetchone()[0]
    return result

def calculate_avg_listeners(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT AVG(listeners)
        FROM lastfm_track_stats
                """)
    result = cur.fetchone()[0]
    return result
```



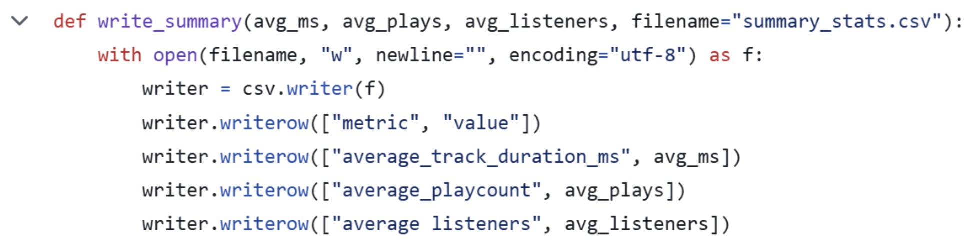
```python
def artist_play_counts(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT artists.name, SUM(lastfm_track_stats.playcount) AS total_playcount
        FROM artists
        JOIN lastfm_track_stats
        ON artists.artist_id = lastfm_track_stats.artist_id
        GROUP BY artists.name
        ORDER BY total_playcount DESC
        LIMIT 15
    """)
    result = cur.fetchall()
    return result

def avg_artist_rank(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT artists.name,
        AVG(songs.rank) AS avg_rank
        FROM artists
        JOIN songs ON artists.artist_id = songs.artist_id
        GROUP BY artists.name
        ORDER BY avg_rank
        LIMIT 15
    """)
    result = cur.fetchall()
    return result
```



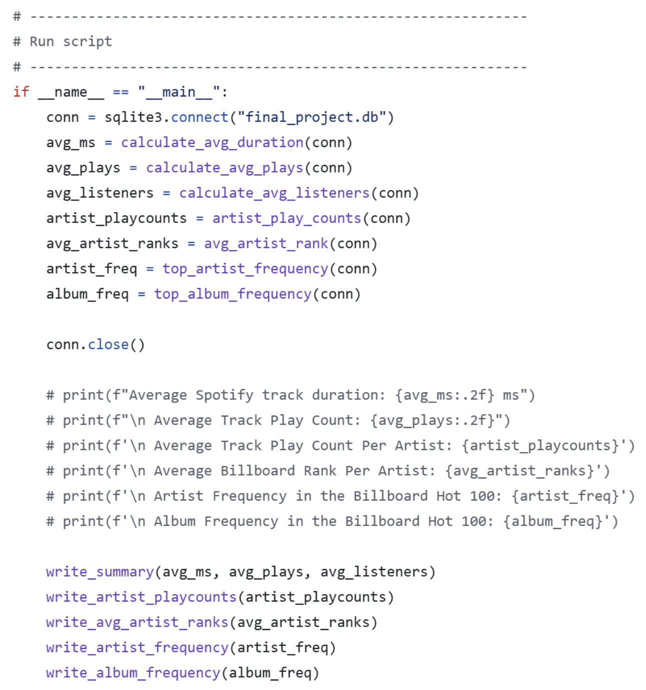
```python
def top_artist_frequency(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT artists.name,
        COUNT(*) AS frequency
        FROM songs
        JOIN artists
        ON songs.artist_id = artists.artist_id
        WHERE songs.rank <= 100
        GROUP BY artists.name
        ORDER BY frequency DESC
        LIMIT 15
    """)
    result = cur.fetchall()
    return result

def top_album_frequency(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT albums.album_name,
        COUNT(*) AS frequency
        FROM songs
        JOIN albums
        ON songs.album_id = albums.album_id
        WHERE songs.rank <= 100
        GROUP BY albums.album_name
        ORDER BY frequency DESC
        LIMIT 15
    """)
    result = cur.fetchall()
    return result
```



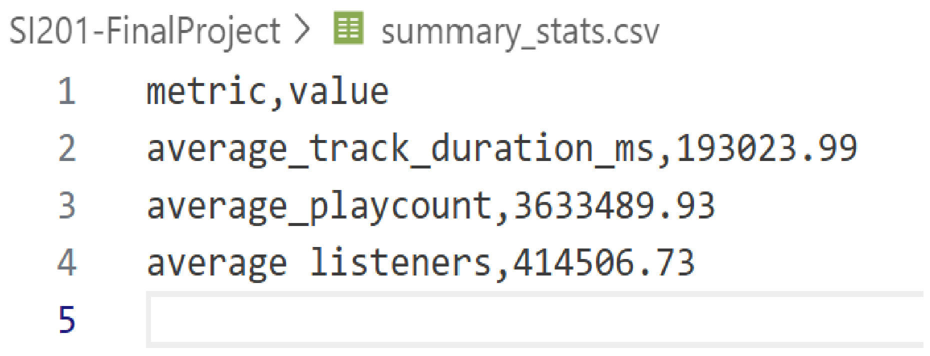
```python
# ------------------------------------------------------------
# Write each output into individual files
# ------------------------------------------------------------
def write_artist_playcounts(results, filename="artist_playcounts.csv"):
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["artist", "total_playcount"])
        writer.writerows(results)

def write_avg_artist_ranks(results, filename="avg_artist_ranks.csv"):
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["artist", "avg_rank"])
        writer.writerows(results)

def write_artist_frequency(results, filename="artist_frequency.csv"):
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["artist", "frequency"])
        writer.writerows(results)

def write_album_frequency(results, filename="album_frequency.csv"):
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["album", "frequency"])
        writer.writerows(results)
```



```python
def write_summary(avg_ms, avg_plays, avg_listeners, filename="summary_stats.csv"):
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["metric", "value"])
        writer.writerow(["average_track_duration_ms", avg_ms])
        writer.writerow(["average_playcount", avg_plays])
        writer.writerow(["average listeners", avg_listeners])
```

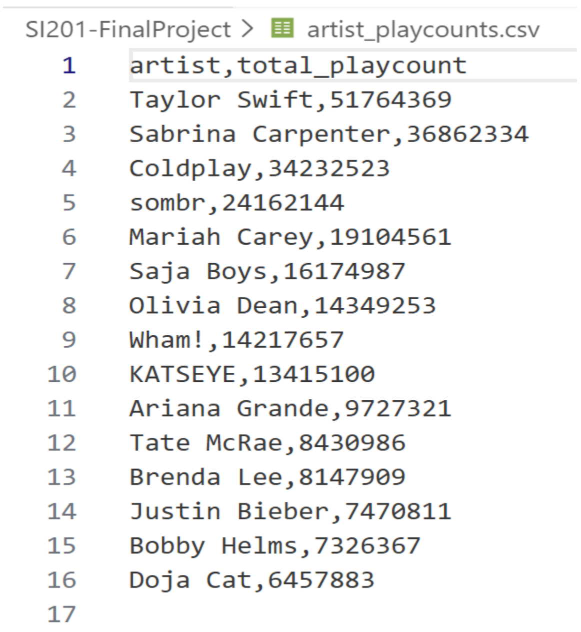


```python
# -------------------------------------------------------------
# Run script
# -------------------------------------------------------------
if __name__ == "__main__":
    conn = sqlite3.connect("final_project.db")
    avg_ms = calculate_avg_duration(conn)
    avg_plays = calculate_avg_plays(conn)
    avg_listeners = calculate_avg_listeners(conn)
    artist_playcounts = artist_play_counts(conn)
    avg_artist_ranks = avg_artist_rank(conn)
    artist_freq = top_artist_frequency(conn)
    album_freq = top_album_frequency(conn)

    conn.close()

    # print(f"Average Spotify track duration: {avg_ms:.2f} ms")
    # print(f"\n Average Track Play Count: {avg_plays:.2f}")
    # print(f'\n Average Track Play Count Per Artist: {artist_playcounts}')
    # print(f'\n Average Billboard Rank Per Artist: {avg_artist_ranks}')
    # print(f'\n Artist Frequency in the Billboard Hot 100: {artist_freq}')
    # print(f'\n Album Frequency in the Billboard Hot 100: {album_freq}')

    write_summary(avg_ms, avg_plays, avg_listeners)
    write_artist_playcounts(artist_playcounts)
    write_avg_artist_ranks(avg_artist_ranks)
    write_artist_frequency(artist_freq)
    write_album_frequency(album_freq)
```

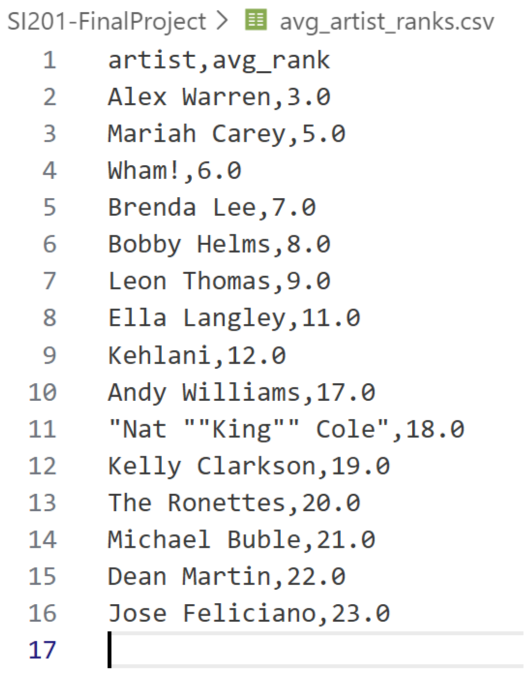


SI201-FinalProject > summary_stats.csv

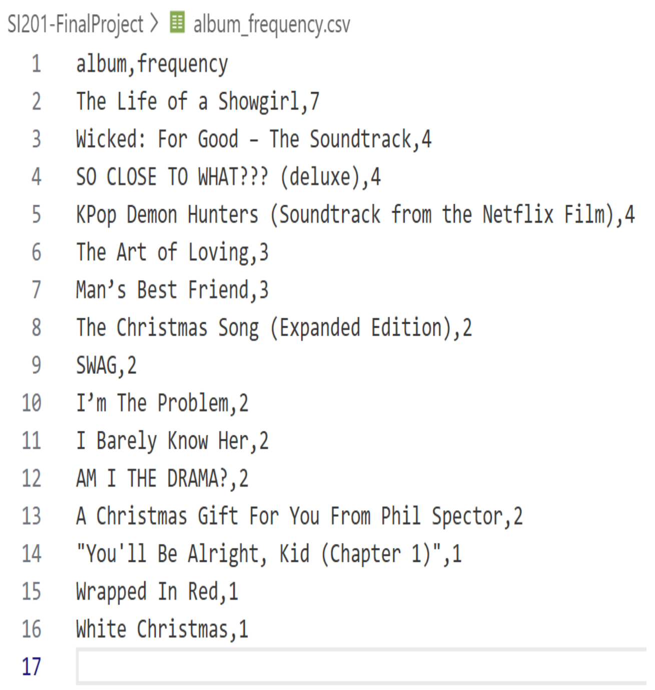
```
metric,value
average_track_duration_ms,193023.99
average_playcount,3633489.93
average listeners,414506.73

```


SI201-FinalProject > artist_playcounts.csv

```
artist,total_playcount
Taylor Swift,51764369
Sabrina Carpenter,36862334
Coldplay,34232523
sombr,24162144
Mariah Carey,19104561
Saja Boys,16174987
Olivia Dean,14349253
Wham!,14217657
KATSEYE,13415100
Ariana Grande,9727321
Tate McRae,8430986
Brenda Lee,8147909
Justin Bieber,7470811
Bobby Helms,7326367
Doja Cat,6457883

```

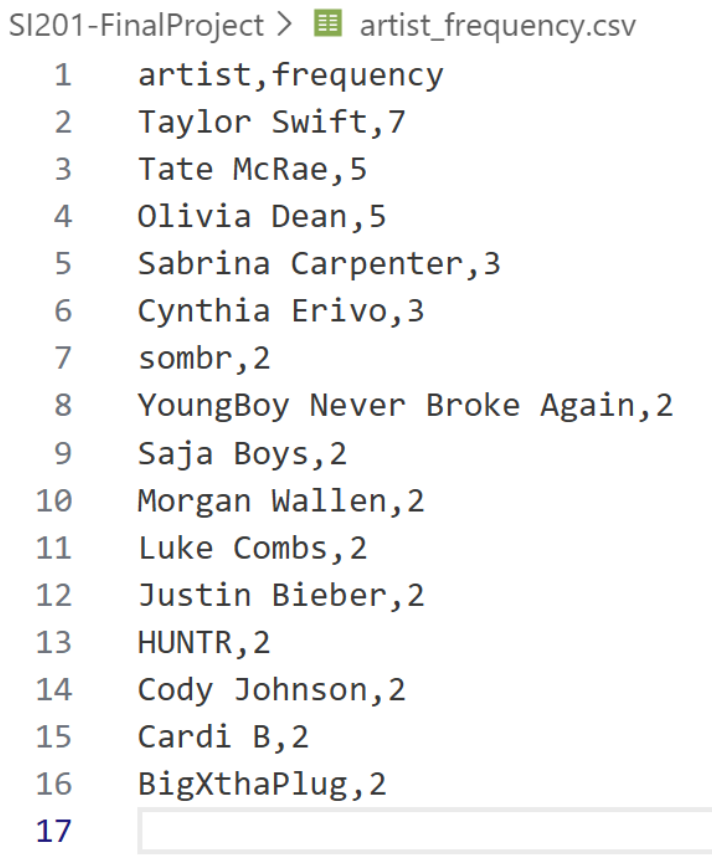

SI201-FinalProject > avg_artist_ranks.csv

```
artist,avg_rank
Alex Warren,3.0
Mariah Carey,5.0
Wham!,6.0
Brenda Lee,7.0
Bobby Helms,8.0
Leon Thomas,9.0
Ella Langley,11.0
Kehlani,12.0
Andy Williams,17.0
"Nat ""King"" Cole",18.0
Kelly Clarkson,19.0
The Ronettes,20.0
Michael Buble,21.0
Dean Martin,22.0
Jose Feliciano,23.0

```



SI201-FinalProject > album_frequency.csv

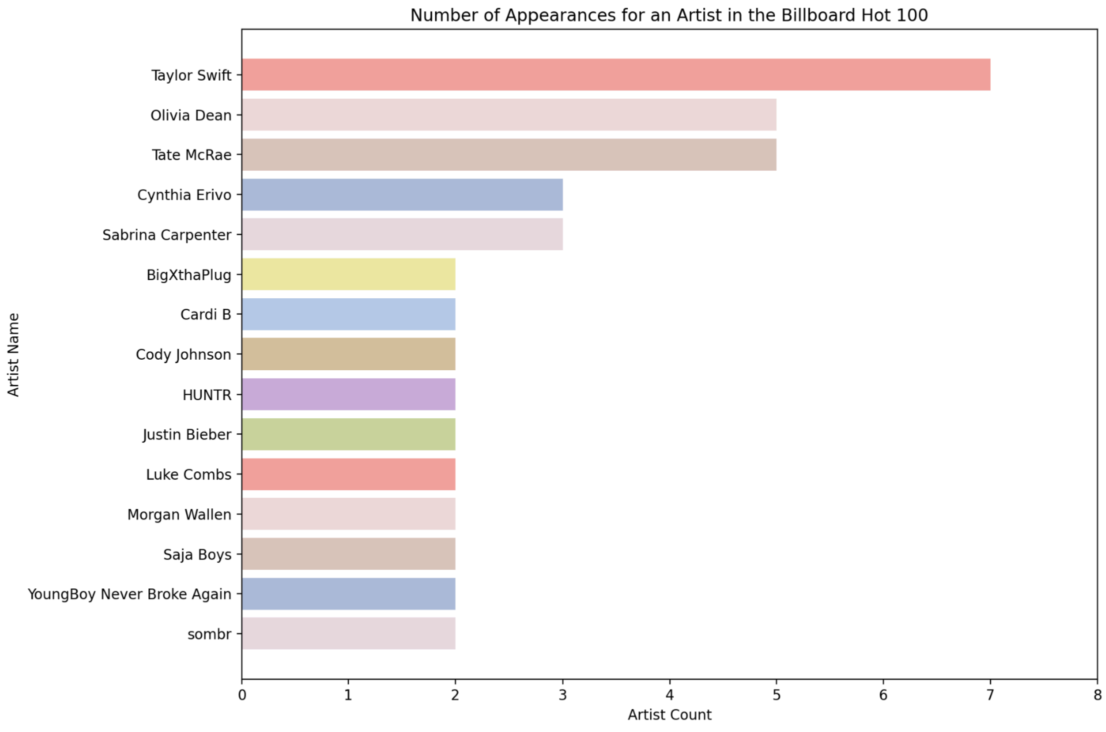
```
album,frequency
The Life of a Showgirl,7
Wicked: For Good - The Soundtrack,4
SO CLOSE TO WHAT??? (deluxe),4
KPop Demon Hunters (Soundtrack from the Netflix Film),4
The Art of Loving,3
Man’s Best Friend,3
The Christmas Song (Expanded Edition),2
SWAG,2
I’m The Problem,2
I Barely Know Her,2
AM I THE DRAMA?,2
A Christmas Gift For You From Phil Spector,2
"You'll Be Alright, Kid (Chapter 1)",1
Wrapped In Red,1
White Christmas,1

```

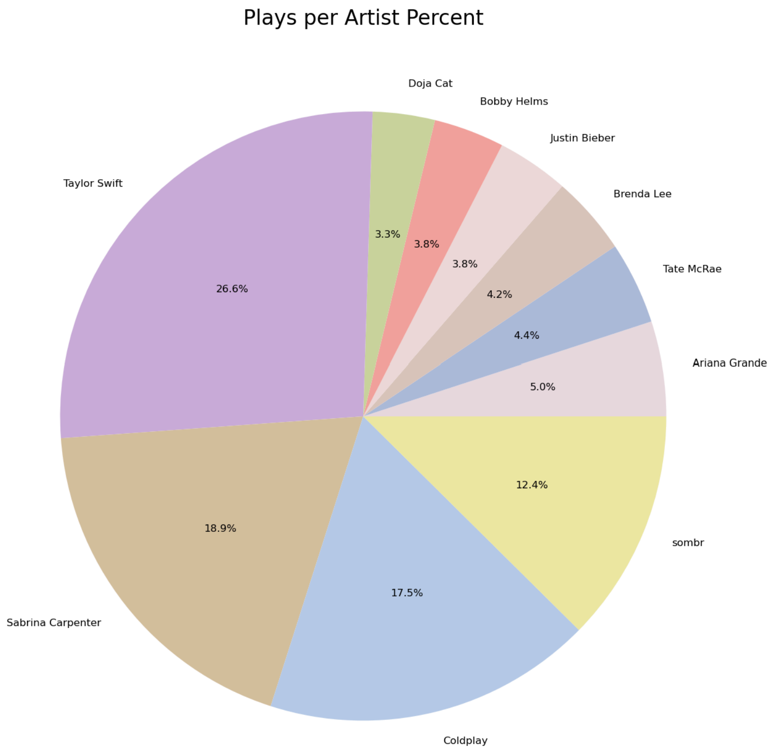


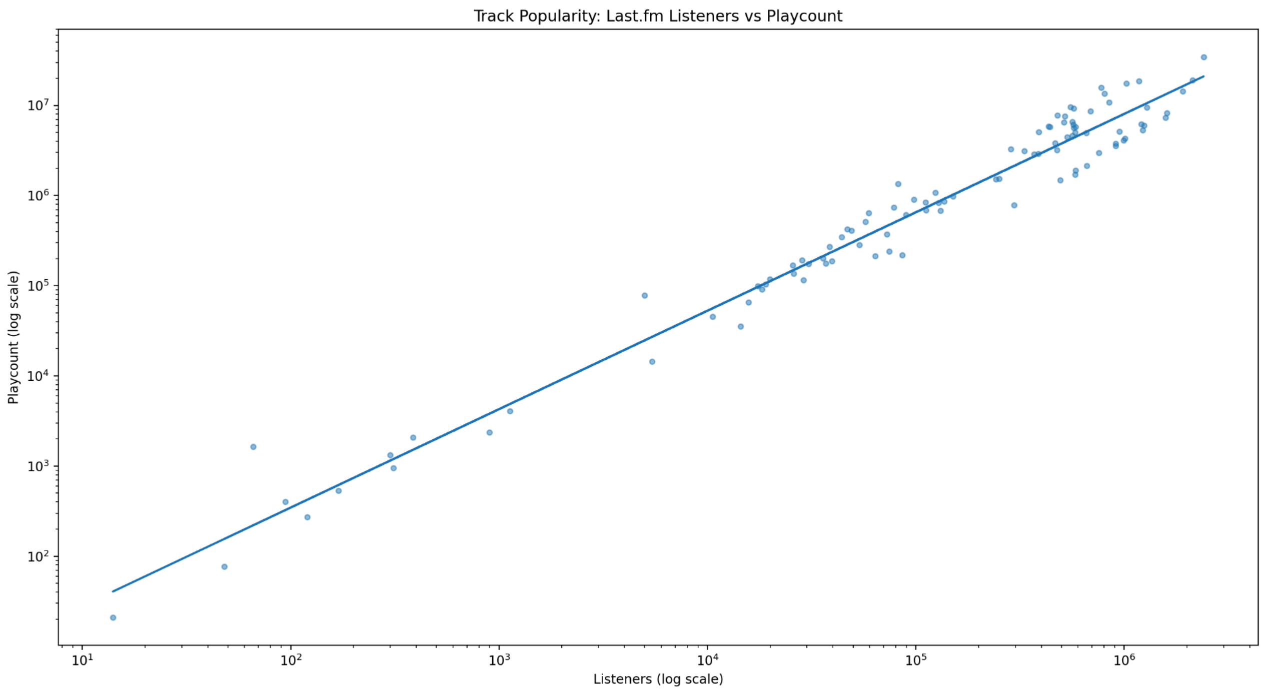
SI201-FinalProject > artist_frequency.csv

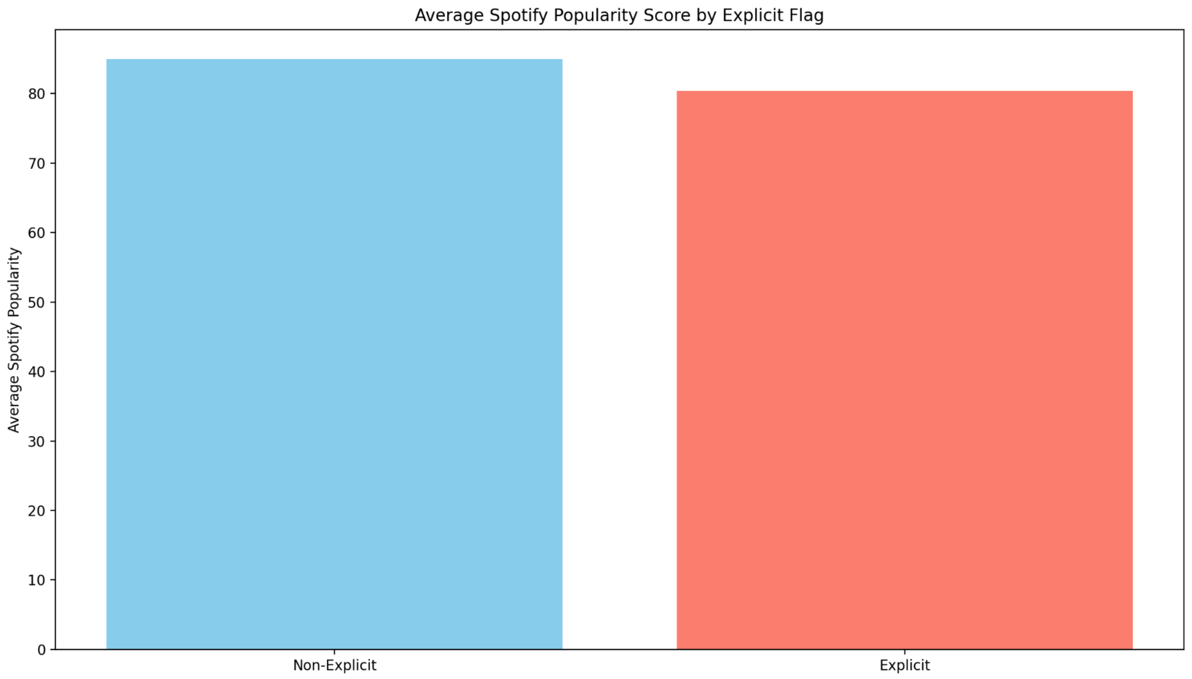
```
artist,frequency
Taylor Swift,7
Tate McRae,5
Olivia Dean,5
Sabrina Carpenter,3
Cynthia Erivo,3
sombr,2
YoungBoy Never Broke Again,2
Saja Boys,2
Morgan Wallen,2
Luke Combs,2
Justin Bieber,2
HUNTR,2
Cody Johnson,2
Cardi B,2
BigXthaPlug,2

```

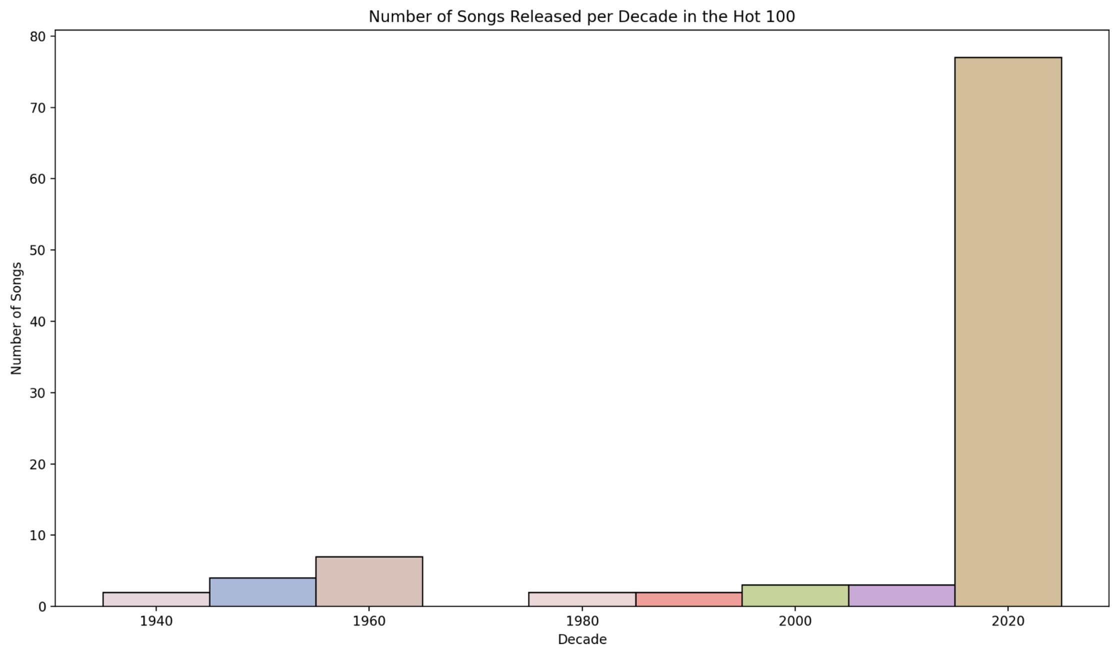
5. Visualizations






Track Popularity: Last.fm Listeners vs Playcount
Playcount (log scale)
Listeners (log scale)
10^1
10^2
10^3
10^4
10^5
10^6
10^7


Average Spotify Popularity Score by Explicit Flag
Average Spotify Popularity
0
10
20
30
40
50
60
70
80
Non-Explicit
Explicit

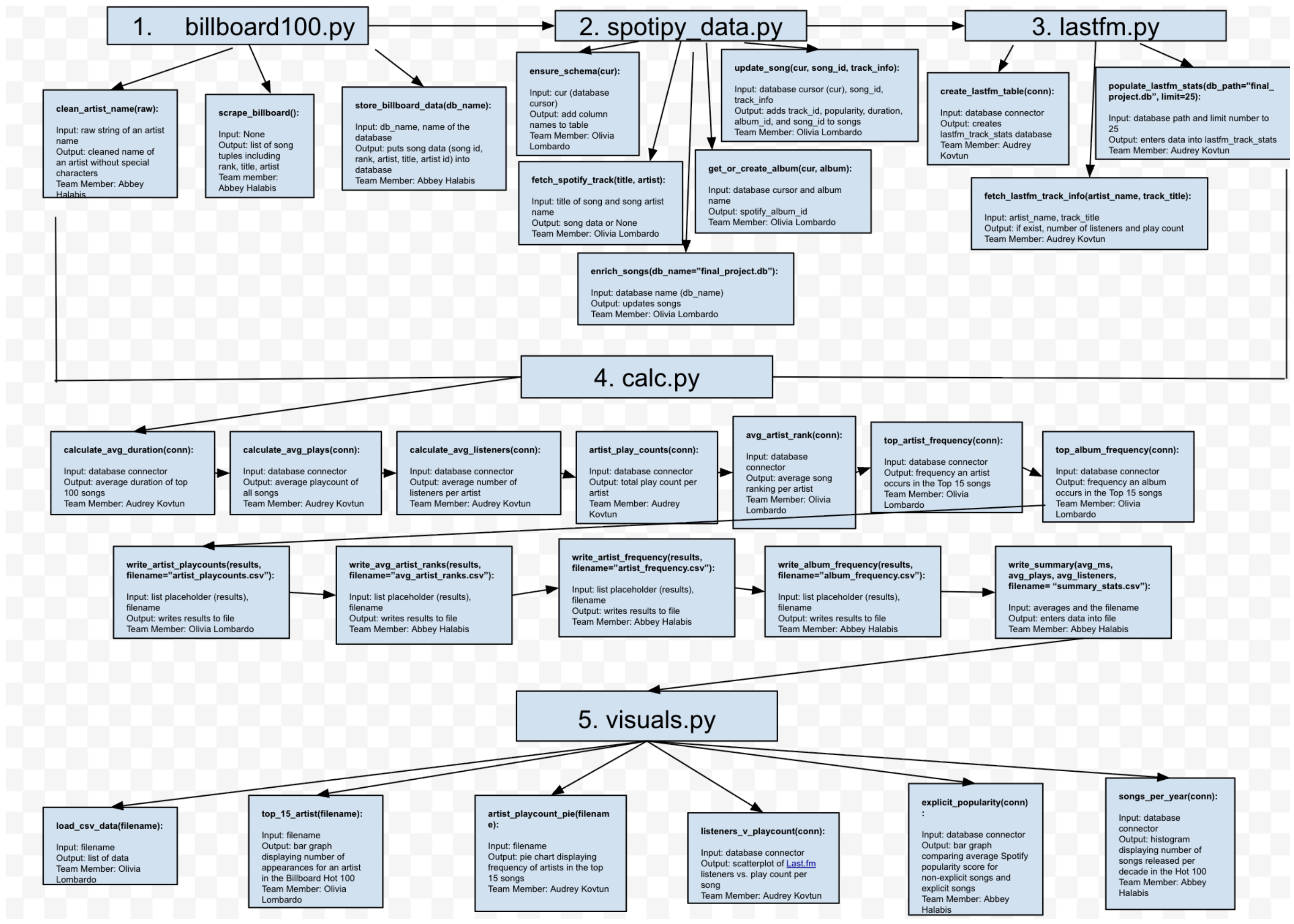


6. Instructions for Running Code

1. If any database exists, this needs to be deleted to refresh the 100 Billboard songs & gather metrics for an updated list. I do this by closing the SQLite browser and deleting the database from the sidebar in my VS Code.
2. Navigate to the "billboard100.py" file and run the code 4 times. Each time you run the code, it should output:
    a. "Inserted 25 new songs."
3. On the last run, it should say "Inserted 0 new songs."
4. Next, go to the file named "spotipy_data.py." This file contains information utilizing the Spotify API, but through the Spotipy package
5. Run this code 4 times through. Each run, it should output:
    a. "Searching for track: How Far Does A Goodbye Go - Jason Aldean
    b. → Found: How Far Does A Goodbye Go [17U2M7HB14yGe9QSAmbyyB]
6. This will be formatted the same way for each song within the Billboard artist data
7. At the end of each run in this file, it should output:
    a. "Run complete - run again for the next 25 songs."
8. At the final run, after all 100 songs were searched, it will say "All songs already enriched!"
9. After running the code in "spotipy_data.py," navigate to the last API file named "lastfm.py."
10. Similar to the last two, run the file, and it should collect only 25 rows of data each time. It will print this output for each song endpoint:
    a. "Fetching: Justin Bieber - Yukon
    b. → Saved: listeners=365348, playcount=2793054
11. After 4 successful runs, it will say "All tracks already processed - no new items to add."

12. Once all three BSoup and API files have been run to put information into a database, you can navigate to the file labeled "calc.py."
    a. This file contains all calculations done based on the tables from our final database. It selects some form of data from all four tables, including utilizing at least one database JOIN.
13. Run this file once, and if successful, 5 files should appear in the SI201-FinalProject folder labeled:
    a. "Artist_playcounts.csv"
        i. Displays the total number of plays for artists among the top 15 in terms of plays
    b. "Avg_artist_ranks.csv"
        i. Calculates every artist's average rank and returns a list of the top 15 artists and their associated average rank
    c. "Artist_frequency.csv"
        i. Calculates the frequency of an artist appearing in the Billboard Hot 100 and returns the top 15 artists and their frequency count
    d. "Album_frequency.csv"
        i. Similar to "artist_frequency.csv," this calculates the frequency of an album appearing in the Billboard Hot 100 and returns the top 15 albums and their frequency count
    e. "Summary_stats.csv"
        i. Includes summary information, including: Average_track_duration_ms, average_playcount, average listeners
14. Finally, go to the file named "visuals.py" to create the visualizations.
15. Run this file once, and a pop-up window will appear for matplotlib, and it will display five graphs, one at a time.

7. Function Diagram


1. billboard100.py
clean_artist_name(raw):
Input: raw string of an artist name
Output: cleaned name of an artist without special characters
Team Member: Abbey Halabis
scrape_billboard():
Input: None
Output: list of song tuples including rank, title, artist
Team member: Abbey Halabis
store_billboard_data(db_name):
Input: db_name, name of the database
Output: puts song data (song id, rank, artist, title, artist id) into database
Team Member: Abbey Halabis
2. spotipy_data.py
ensure_schema(cur):
Input: cur (database cursor)
Output: add column names to table
Team Member: Olivia Lombardo
update_song(cur, song_id, track_info):
Input: database cursor (cur), song_id, track_info
Output: adds track_id, popularity, duration, album_id, and song_id to songs
Team Member: Olivia Lombardo
fetch_spotify_track(title, artist):
Input: title of song and song artist name
Output: song data or None
Team Member: Olivia Lombardo
get_or_create_album(cur, album):
Input: database cursor and album name
Output: spotify_album_id
Team Member: Olivia Lombardo
enrich_songs(db_name="final_project.db"):
Input: database name (db_name)
Output: updates songs
Team Member: Olivia Lombardo
3. lastfm.py
create_lastfm_table(conn):
Input: database connector
Output: creates lastfm_track_stats database
Team Member: Audrey Kovtun
populate_lastfm_stats(db_path="final_project.db", limit=25):
Input: database path and limit number to 25
Output: enters data into lastfm_track_stats
Team Member: Audrey Kovtun
fetch_lastfm_track_info(artist_name, track_title):
Input: artist_name, track_title
Output: if exist, number of listeners and play count
Team Member: Audrey Kovtun
4. calc.py
calculate_avg_duration(conn):
Input: database connector
Output: average duration of top 100 songs
Team Member: Audrey Kovtun
calculate_avg_plays(conn):
Input: database connector
Output: average playcount of all songs
Team Member: Audrey Kovtun
calculate_avg_listeners(conn):
Input: database connector
Output: average number of listeners per artist
Team Member: Audrey Kovtun
artist_play_counts(conn):
Input: database connector
Output: total play count per artist
Team Member: Audrey Kovtun
avg_artist_rank(conn):
Input: database connector
Output: average song ranking per artist
Team Member: Olivia Lombardo
top_artist_frequency(conn):
Input: database connector
Output: frequency an artist occurs in the Top 15 songs
Team Member: Olivia Lombardo
top_album_frequency(conn):
Input: database connector
Output: frequency an album occurs in the Top 15 songs
Team Member: Olivia Lombardo
write_artist_playcounts(results, filename="artist_playcounts.csv"):
Input: list placeholder (results), filename
Output: writes results to file
Team Member: Olivia Lombardo
write_avg_artist_ranks(results, filename="avg_artist_ranks.csv"):
Input: list placeholder (results), filename
Output: writes results to file
Team Member: Abbey Halabis
write_artist_frequency(results, filename="artist_frequency.csv"):
Input: list placeholder (results), filename
Output: writes results to file
Team Member: Abbey Halabis
write_album_frequency(results, filename="album_frequency.csv"):
Input: list placeholder (results), filename
Output: writes results to file
Team Member: Abbey Halabis
write_summary(avg_ms, avg_plays, avg_listeners, filename= "summary_stats.csv"):
Input: averages and the filename
Output: enters data into file
Team Member: Abbey Halabis
5. visuals.py
load_csv_data(filename):
Input: filename
Output: list of data
Team Member: Olivia Lombardo
top_15_artist(filename):
Input: filename
Output: bar graph displaying number of appearances for an artist in the Billboard Hot 100
Team Member: Olivia Lombardo
artist_playcount_pie(filename):
Input: filename
Output: pie chart displaying frequency of artists in the top 15 songs
Team Member: Audrey Kovtun
listeners_v_playcount(conn):
Input: database connector
Output: scatterplot of Last.fm listeners vs. play count per song
Team Member: Audrey Kovtun
explicit_popularity(conn):
Input: database connector
Output: bar graph comparing average Spotify popularity score for non-explicit songs and explicit songs
Team Member: Abbey Halabis
songs_per_year(conn):
Input: database connector
Output: histogram displaying number of songs released per decade in the Hot 100
Team Member: Abbey Halabis

8. Documentation

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
11/29	Working on scraping the Billboard Hot 100. Could not get my code to only retrieve 25 data points at a time. I was getting the first 25 results over and over again.	Generative AI- Chat GPT Lecture Slides from week 14	Yes, it now scrapes all 100 Billboard songs each time the program runs, queries the existing database to retrieve all stored (title, artist) pairs, filters out any songs that already exist in the database, selects

			only the next 25 new songs from the remaining dataset, and inserts those 25 new entries into the database.
11/30	When working with the Billboard 100 data, it was difficult to obtain the artist in a simple format since it is needed to search the other API's. I had to use a lot of helper and cleaning functions at first to only grab the primary artist.	Generative AI- Chat GPT	Yes, the function now only grabs the primary artist and is a usable format for the other APIs. Example: "Riley Green Featuring Ella Langley" is now only "Riley Green"
12/1	We were not sure how to select data from our database and turn it into calculations.	GeeksForGeeks	Yes, I read over the SQLite SUM() function article to learn how to use the built-in functions.
12/3	Since I have not officially created any plots in Visual Studio Code before, I wasn't sure how to approach and begin building my subplots and how they would appear within the code.	Discussion 13 slides, assignment, matplotlib documentation	Yes, completing the discussion assignment was super helpful in learning how to access information from a CSV and create a plot. My section leader mentioned how easy it was to access documentation per type of graph, which has been helpful.