Final Project Report - SI 206

Repository Link: https://github.com/Sadavar/si206-finalproject
Varun Sadasivam and Noah Feller

Project Goals

The primary goal of this project was to access several information sources to calculate and visualize insights related to the current music industry. Our initial plan was to scrape the song name, song ranking, and artist name from the Billboard Hot 100 charts. From that information, we could calculate the chart's most popular artist, least popular artist, and average popularity artist based on the rankings and artist song count.

Using the Bandsintown API, we planned to compare the artist rankings to how often artists went on tour and where they performed most frequently. We intended to use several tables to collect the information. A core goal was to create a dedicated artists table that provided each artist with an integer key, allowing a events data table and a song chart table to access the artists without storing duplicate artist name strings.

We planned to use Matplotlib to visualize how often Billboard Hot 100 artists went on tour and compare different artists' popularity. After encountering access issues with the Bandsintown API, we chose to use the Ticketmaster API instead to gather similar data. We also decided to access the Spotify API for extra credit.

Our final project retrieves the song name, song ranking, and artist name data of the 100 songs listed on the Billboard Hot 100. From the Spotify API, we gathered the genre and popularity of the artists on the Billboard Hot 100 rankings. From the Ticketmaster API, we collected the event name, artist, venue, and event date from recent and upcoming Ticketmaster music events. We successfully met our goal of creating an artists table with a unique integer key assigned to each artist. We were able to use the artist keys across all the tables that shared artists but collected information from different sources. This allowed us to avoid storing duplicate artist name string data.

We were also successful in our goal of calculating the most popular, least popular, and average popularity artists on the Billboard Hot 100. We utilized data from the Spotify API to find the popularity values. We were able to visualize this information in a bar chart using Matplotlib, matching another goal. We pivoted from the other visualization goal of plotting how often different artists went on tour. Instead, we created graphs showcasing the number of Ticketmaster events per month and the most popular venues based on data stored in our events table. Another visualization showcases the Spotify popularity of the top 10 artists on the Billboard rankings. Overall, our team was successful in accomplishing our primary goal of accessing music information sources to calculate and visualize insights. Further, we met other key project aims and adapted as challenges arose and our work evolved.

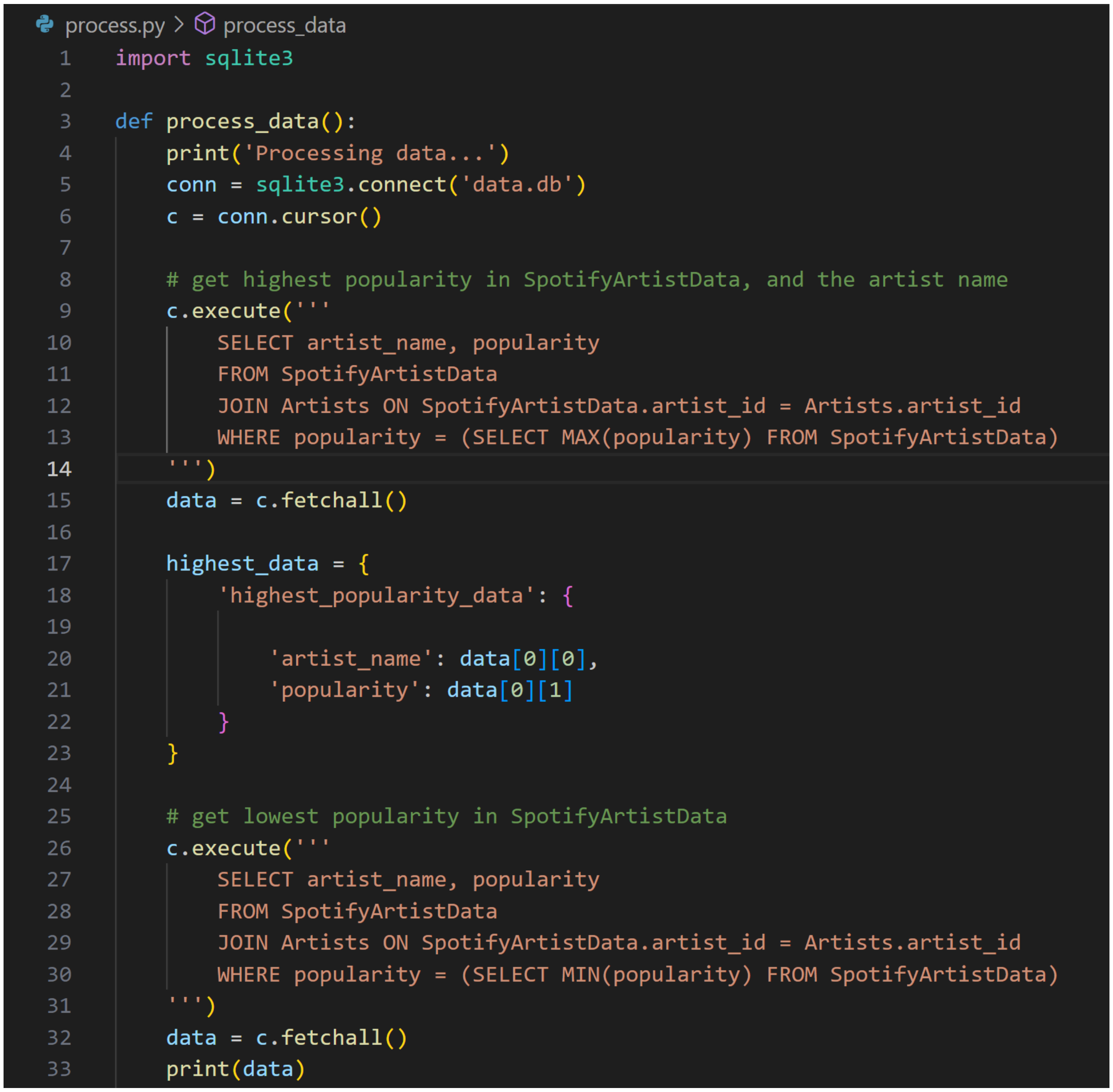
Problems

1. **Trouble accessing the Bandsintown API**
   We initially planned to use data from the site Bandsintown. Due to the licensing structure of the Bandsintown API, our team had difficulty gaining access to the API's data without approval from the company. We were able to instead use the Ticketmaster API to gather similar data.

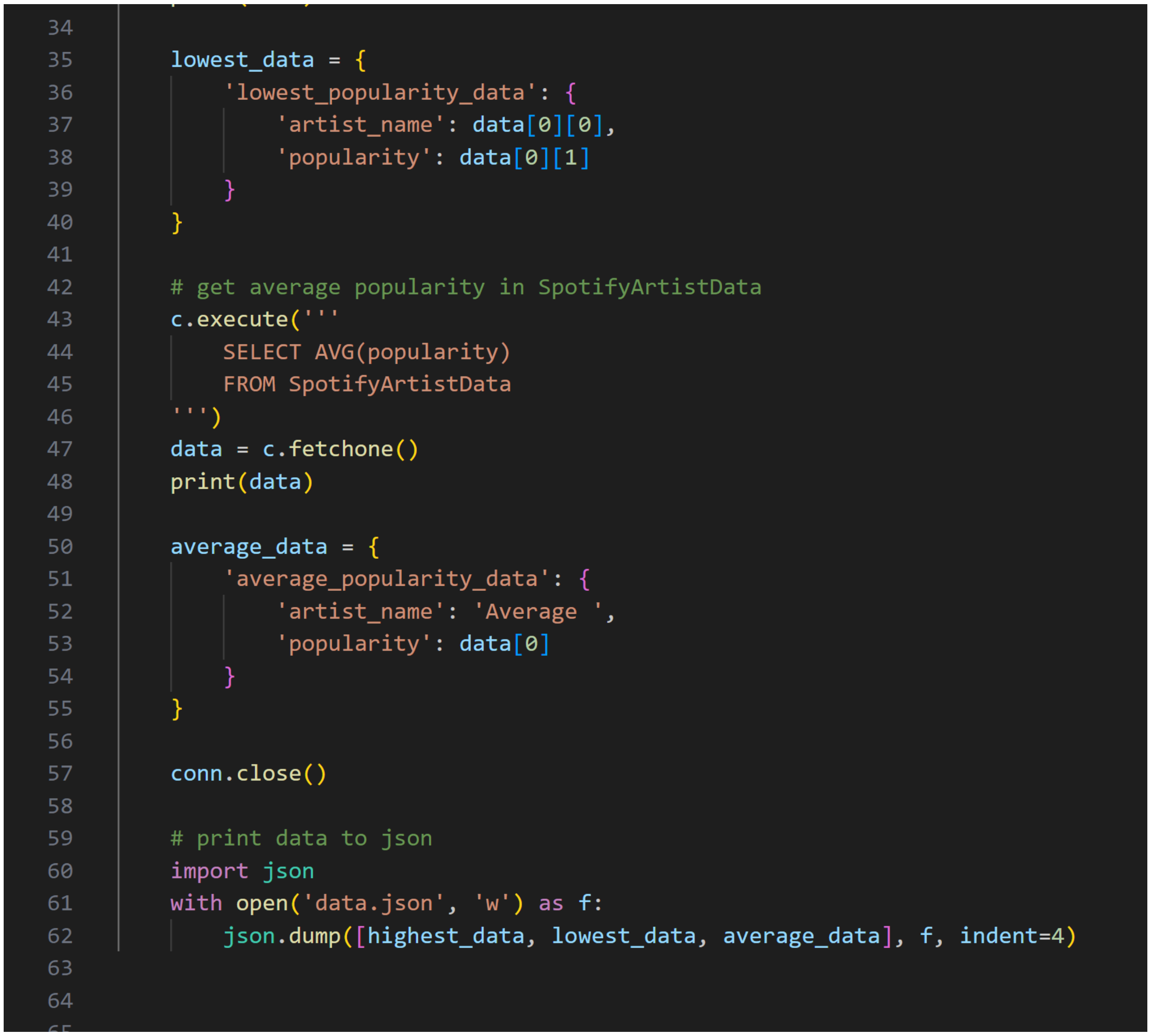
2. **Handling multiple artists listed on songs and performing at concerts**
   We encountered a lot of complexity in structuring our tables and creating visualizations when multiple artists were listed for each song or for a specific concert. It was especially difficult to visualize artist ranks if more than the lead artist were considered in calculations. In cases where there were several artists or featured artists, we solved the complexity by using regex and boolean logic to only store the first artist collected from the Billboard website artist string field. With the Ticketmaster API, an event's artist or artists are provided in a list, allowing us to apply simple data cleaning to store only the first artist listed in our database.

3. **Regex challenges with the cleanBillboard function**
   The cleanBillboard function takes in the list named billboard_list from get_billboard_data(), which collects data from the Billboard Hot 100 website. The list contains each song title and a string containing the song's artist or artists as written on the website. When there were multiple artists on a song, it proved challenging to parce just the first artist's name from the string due to the variation in how multiple artists were listed. We were not able to find a way to just use regex to both check if there were multiple artists in the string and then only extract the first name. To solve this, the cleanBillboard function uses a combination of if statements and regex to check if there is more than one name. If there is, the regex captures only the text before "Featuring", a comma, "And", "With", or a similar separator.

Calculations

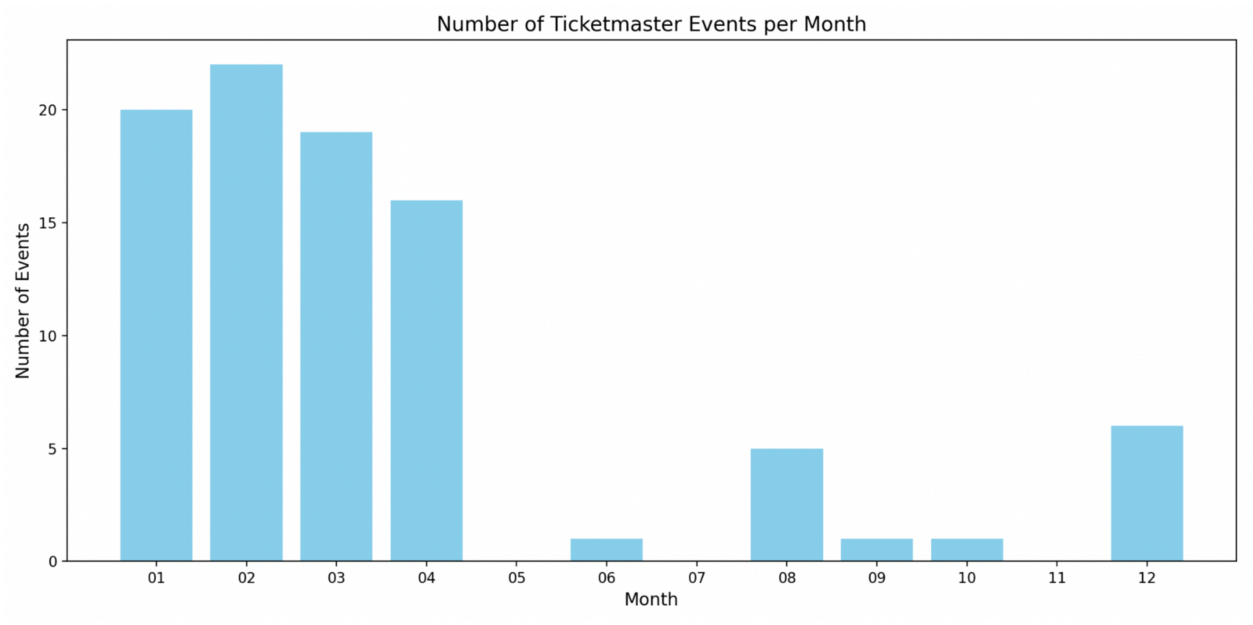

process.py > process_data

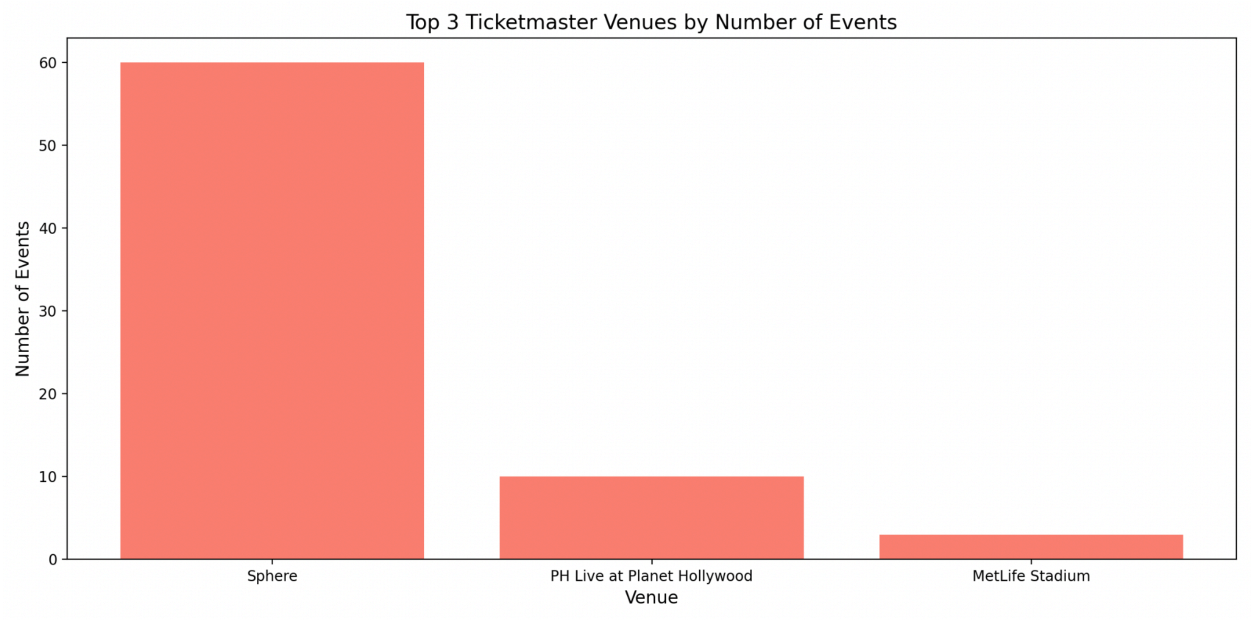
```python
import sqlite3

def process_data():
    print('Processing data...')
    conn = sqlite3.connect('data.db')
    c = conn.cursor()

    # get highest popularity in SpotifyArtistData, and the artist name
    c.execute('''
        SELECT artist_name, popularity
        FROM SpotifyArtistData
        JOIN Artists ON SpotifyArtistData.artist_id = Artists.artist_id
        WHERE popularity = (SELECT MAX(popularity) FROM SpotifyArtistData)
    ''')
    data = c.fetchall()

    highest_data = {
        'highest_popularity_data': {

            'artist_name': data[0][0],
            'popularity': data[0][1]
        }
    }

    # get lowest popularity in SpotifyArtistData
    c.execute('''
        SELECT artist_name, popularity
        FROM SpotifyArtistData
        JOIN Artists ON SpotifyArtistData.artist_id = Artists.artist_id
        WHERE popularity = (SELECT MIN(popularity) FROM SpotifyArtistData)
    ''')
    data = c.fetchall()
    print(data)
```

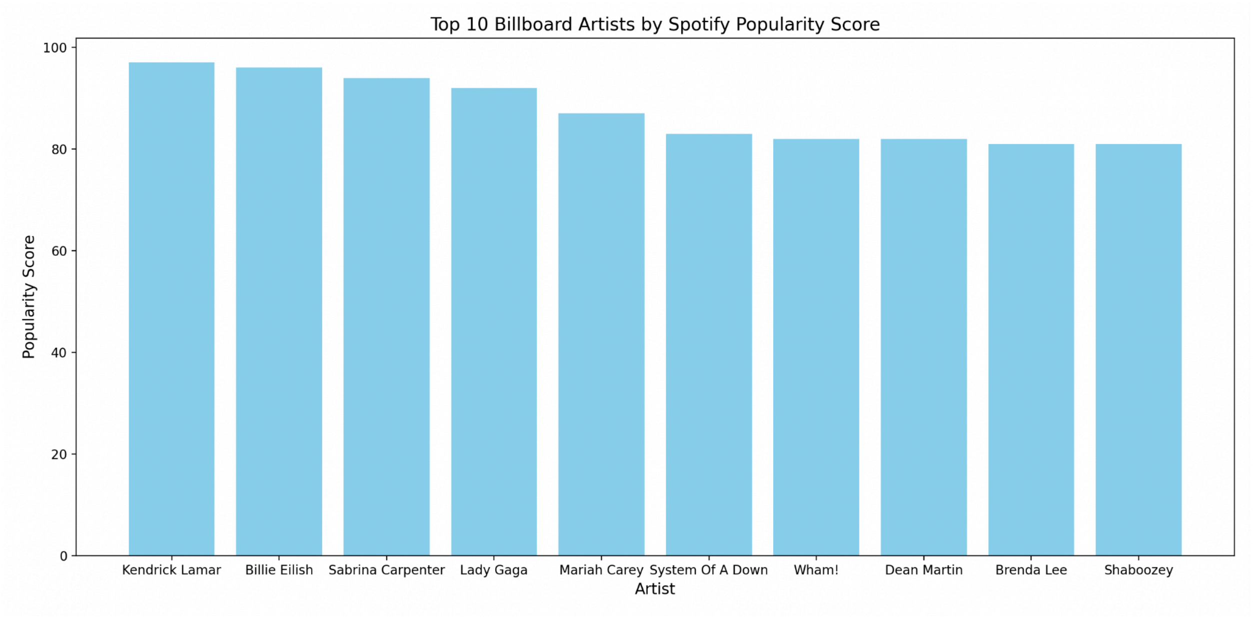


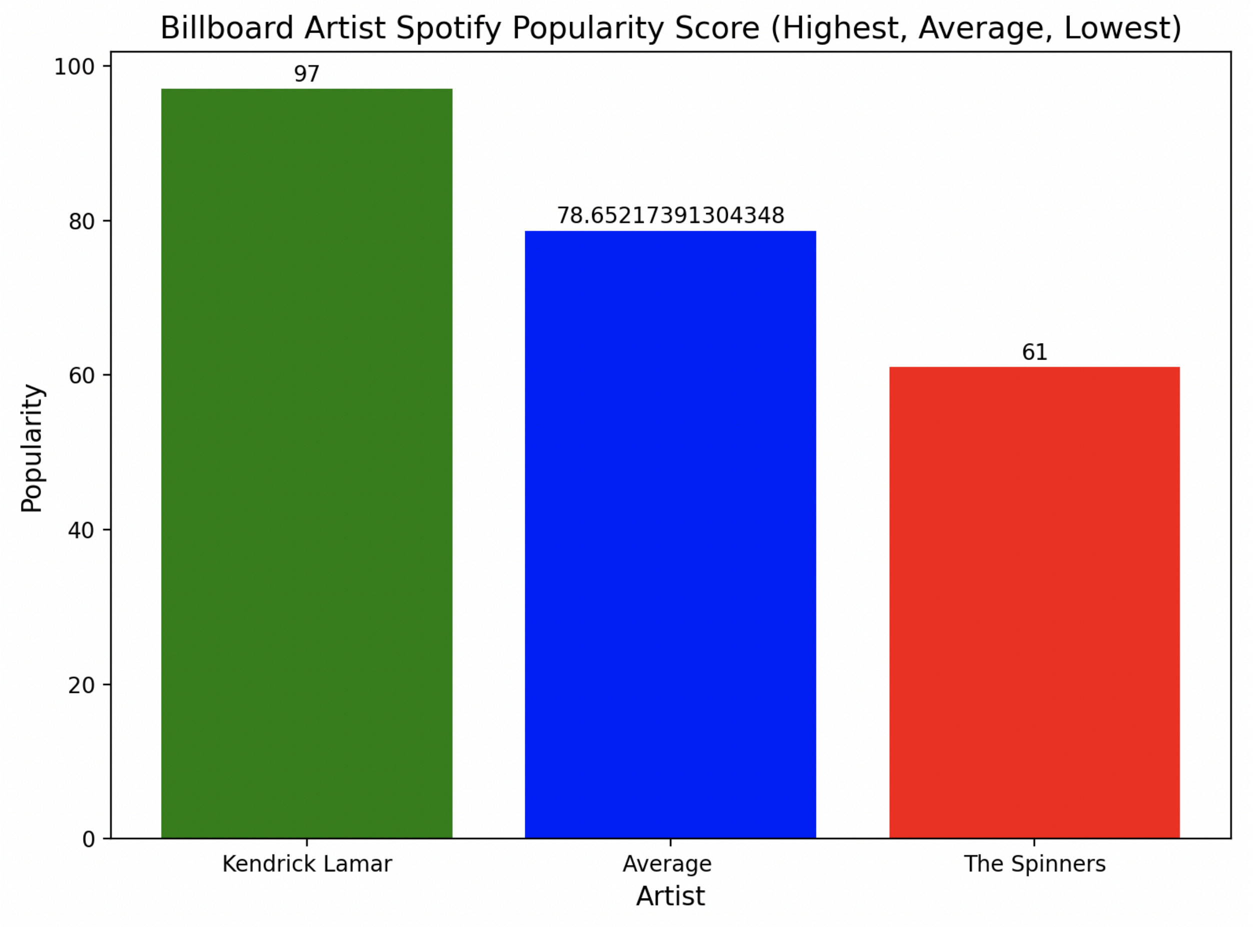
```python

    lowest_data = {
        'lowest_popularity_data': {
            'artist_name': data[0][0],
            'popularity': data[0][1]
        }
    }

    # get average popularity in SpotifyArtistData
    c.execute('''
        SELECT AVG(popularity)
        FROM SpotifyArtistData
    ''')
    data = c.fetchone()
    print(data)

    average_data = {
        'average_popularity_data': {
            'artist_name': 'Average ',
            'popularity': data[0]
        }
    }

    conn.close()

    # print data to json
    import json
    with open('data.json', 'w') as f:
        json.dump([highest_data, lowest_data, average_data], f, indent=4)


```

Visualizations






Top 10 Billboard Artists by Spotify Popularity Score
Popularity Score
Artist
Kendrick Lamar
Billie Eilish
Sabrina Carpenter
Lady Gaga
Mariah Carey
System Of A Down
Wham!
Dean Martin
Brenda Lee
Shaboozey
0
20
40
60
80
100


Billboard Artist Spotify Popularity Score (Highest, Average, Lowest)
Popularity
Artist
Kendrick Lamar
Average
The Spinners
97
78.65217391304348
61
0
20
40
60
80
100

Code Instructions

Before running, make sure that all required python packages are installed. They are listed in requirements.txt. Best practice would be to make a virtual environment and install using these steps in a unix compatible terminal or WSL. If you can't set up a terminal, feel free to use any other python environment tool like Anaconda. Just make sure to make a .env and install dependencies.

Terminal commands:

- - **python3 -m venv env**
  - **source env/bin/activate**
  - **pip install -r requirements.txt**
- Next make a file named .env file to load api keys
  - Make a .env file
    - **touch .env or File - > New File > .env**
  - Then right these values to .env:
    - **SPOTIFY_CLIENT_ID = "fd5eae8a9f5d4390b801c425afb2105b"**
    - **SPOTIFY_CLIENT_SECRET = "bf6010fd039045aabb9fdc7db9df333b"**
    - **TICKETMASTER_KEY = "UQaRoP5fLCVe2MRA3UrheYt3GjeP9Smb"**
- Then, our code is run through one command: **python3 presentation.py**
- Each time that command is run, <= 25 rows are added to the database and visualizations are presented
- If you want more accurate visualizations based on more data, please run the command more times
- That's it! Functions are defined in different files and imported in

Improvements since Grading Session

1. Eliminated duplicate rows in tables by creating a Venues and Events table and using those ids to link with the TicketMasterEvents table. Also added artists from ticketmaster events to the artists table and used that for the artist ID.
2. Changed data column from TEXT to DATE in TicketMasterEvents table, cleaning and inputting date formats accordingly.
3. Changed DB processing to have <= 25 rows of data added per use of the script. The amount of rows added to the table (calculated by counting the rows across all tables before and after running the script) is printed at the end.

These should fix the criticism we received and help create a more efficient DB with no duplication.

Functions

presentation.py

get_data()
Input: None
Output: None
Description: Coordinates data retrieval from multiple sources. Calls functions to get_billboard_data, save_billboard_data, get_spotify_artist_data, save_spotify_artist_data, get_billboard_track_spotify_data, save_billboard_track_spotify_data, get_ticketmaster_data, and save_ticketmaster_data.

main()
Input: None
Output: None
Description: Orchestrates the entire data collection and visualization workflow:

- Initializes the database
- Records the initial number of rows in the database
- Calls get_data() to collect data from various sources
- Calls process_data() to analyze and process the collected data
- Prints the number of rows added to the database
- Calls visualize_data() to create visualizations
- Executes when the script is run directly

schema.py

get_num_rows()
Input: None
Output: total (integer)
Description: Connects to the SQLite database, retrieves all table names, and counts the total number of rows across all tables. Returns the total row count.

initialize_db()
Input: None
Output: None
Description: Creates database tables if they don't exist, including:

- Artists
- BillboardTracks
- SpotifyArtistData
- BillboardTrackData
- Events
- Venues
- TicketmasterEvents

Each table is created with appropriate columns and foreign key relationships.

billboard.py

cleanBillboard(billboard_list)
Input: billboard_list
Output: billboard_list
Description: Takes in a list containing the top 100 Billboard songs and the corresponding artists from the Billboard Hot 100 website. Iterates through the list and modifies the artist field to only include the first artist listed for each song. Returns the modified list.

get_billboard_data()
Input: None
Output: billboard_list
Description: Uses BeautifulSoup to scrape the song ranking, song name, and artist names of the 100 songs on the Billboard Hot 100 website. Stores the information in a list of lists "billboard_list." Passes the list into cleanBillboard() so only the first artist listed is included with each song. Returns billboard_list.

save_billboard_data(billboard_list)
Input: billboard_list
Output: None
Description: Connects to the database. Counts how many items are already in the BillboardTracks table. Checks if there are fewer than 100 tracks in the table to continue. Iterates through billboard_list input and adds artists not already in the database to the Artists table. For tracks from the inputted list, finds the corresponding artist IDs from the Artists table, adds new tracks with artist IDs to the BillboardTracks table. Only 3 tracks can be added at a time in order to comply with the <=25 database insert restriction. Closes the connection to the database.

spotify.py

get_spotify_token()
Input: None
Output: token
Description: Connects to the Spotify API to get an API token to access the platform's data. Returns unique token.

get_spotify_artist_data()
Input: None
Output: spotify_artist_data
Description: Retrieves Spotify artist data for artists in the database. Connects to the Spotify API using the obtained token, searches for each artist, and collects their genre and popularity. Limits the data collection to 3 new artists not already in the SpotifyArtistData table. Returns a list of dictionaries containing artist name, genre, and popularity.

save_spotify_artist_data(spotify_artist_data)
Input: spotify_artist_data (list of dictionaries)
Output: None
Description: Saves the Spotify artist data to the SpotifyArtistData table in the database. For each artist in the input list, it finds the corresponding artist_id from the Artists table and inserts the genre and popularity data. Commits the changes and closes the database connection.

get_billboard_track_spotify_data()
Input: spotify_artist_data (list of dictionaries)
Output: None
Description: Saves the Spotify artist data to the SpotifyArtistData table in the database. For each artist in the input list, it finds the corresponding artist_id from the Artists table and inserts the genre and popularity data. Commits the changes and closes the database connection.

save_billboard_track_spotify_data(spotify_data):
Input: spotify_data (list of dictionaries)
Output: None
Description: Saves the Spotify track data to the BillboardTrackData table in the database. For each track in the input list, it inserts the billboard track ID, popularity, and duration. Commits the changes and closes the database connection.

ticketmaster.py

cleanArtists(event_list)
Input: event_list (list of dictionaries)
Output: event_list (modified)
Description: Cleans the artist names in the event list by removing any commas from the first artist's name and ensuring only the first artist is included in the list of artists for each event.

get_us_music_events(api_key, pages)
Input: api_key (string), pages (integer)
Output: event_list (list of dictionaries)
Description: Retrieves music events from the Ticketmaster API for the specified number of pages. Extracts event name, artist name, venue name, and event date for each event. Cleans the artist list using the cleanArtists() function and returns the list of events.

save_ticketmaster_data(event_list)
Input: event_list (list of dictionaries)
Output: None
Description: Saves the Ticketmaster event data to the database. Iterates through the event list, inserting or fetching IDs for artists, venues, and event names. Limits the insertion to 3 new events. Inserts the event details into the TicketmasterEvents table, maintaining relationships with other tables.

get_ticketmaster_data()

Input: None
Output: events_with_artists (list of dictionaries)
Description: Retrieves the Ticketmaster API key from the environment and calls get_us_music_events() to fetch music events. Returns the list of events with artists.

process.py

process_data()
Input: None
Output: None
Description: Processes and analyzes Spotify artist popularity data:

- Finds the artist with the highest popularity score
- Finds the artist with the lowest popularity score
- Calculates the average popularity across all artists

Writes the results to a JSON file ('data.json') with three entries:

- Highest popularity artist
- Lowest popularity artist
- Average popularity data

visualize.py

plot_hml_artist_popularity()
Input: None
Output: None
Description: Reads data from a JSON file containing the highest, lowest, and average artist popularity. Creates a bar chart showing the Spotify popularity scores for these three artists, with different colors representing their popularity levels. Saves the plot as an image.

plot_top_10_spotify_artists_popularity()
Input: None
Output: None
Description: Retrieves the top 10 artists by Spotify popularity from the database. Creates a bar chart displaying these artists' popularity scores, with artists sorted in descending order. Saves the plot as an image.

plot_ticketmaster_top3venues()
Input: None
Output: None
Description: Retrieves the top 3 venues by number of Ticketmaster events from the database. Creates a bar chart showing these venues and their event counts. Saves the plot as an image.

plot_events_per_month()
Input: None
Output: None

Description: Retrieves event dates from the TicketmasterEvents table. Counts the number of events per month, ensuring all months are represented (with 0 for months without events). Creates a bar chart showing the number of events for each month. Saves the plot as an image.

visualize_data()
Input: None
Output: None
Description: Calls the visualization functions in sequence to generate multiple plots: artist popularity, top 10 Spotify artists, top 3 Ticketmaster venues, and events per month.

Resources

Date	**Issue**	**Description**	**Resource Location**	**Result**
12/5/2024	DB design	Needed guidance for the best way to structure the database	Stack Overflow	Solved
12/5/2024	Regex issues	How to use regex to capture only the first artist listed on a song.	Stack Overflow and Regex 101	Solved
12/8/2024	Spotify API guidance	Was unsure what data the API provided, consulted docs for info.	Spotify Developer Docs	Solved
12/9/2024	Ticketmaster API guidance	Used to setup API and understand the formatting of the data collected.	Ticketmaster API Website	Solved
12/9/2024	Matplotlib ideas	Needed ideas for interesting ways to graph the collected data.	U-M GPT	Solved
12/11/2024	Matplotlib issue	Venue data wasn’t visualizing properly, required small code tweak.	U-M GPT	Solved