Final Project Report

Kendall Margolis and Molly Dworkin

SI 206

December 16, 2024

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

Goals

Our original goal was to explore trends in music and concerts by interacting with the Spotify API (Spotipy) and the Ticketmaster API (Apify). We aimed to gather data from Spotify's Billboard Hot 100 playlist, including song details, artist information, and genre, and to collect concert-related data from Ticketmaster, such as event prices and locations. The primary objective was to analyze the relationships between popular music genres and concert pricing trends across different cities, to reveal insights into how music preferences might influence concert pricing and event distributions

To achieve these goals, we duplicated the Billboard Hot 100 playlist to create our own copy because we could only access our personal Spotify data through the API, and the public playlist was not accessible. We then used the Spotipy API to gather data from the playlist such as song names, artists, genres, and artist popularity, storing this information in an SQLite database. Using this data, we calculated how many songs each artist contributed to the playlist and determined the top four genres based on the number of songs in each genre. Additionally, we fetched a multitude of data from Ticketmaster to collect information on concerts such as the artist name, genre, ticket price (minimum and maximum), the event name, and the event location (city and state). We then stored this information in another SQLite database and used it to calculate the average ticket price for each of the top four genres, as well as to analyze how events are distributed based on ticket prices and their corresponding cities

We structured the data into two tables for each API:

- **Spotipy API**
    - **Artists Table:** Includes each artist's ID number, the artist's name, the genre of their music, and the artist's popularity score
    - **Songs Table:** Includes the song's name and the artist ID number (linked to the Artists table)
- **Ticketmaster API**
    - **Artists Table**: Includes each artist's ID number, the artist's name, the genre of their music, the minimum event ticket price, and the maximum event ticket price

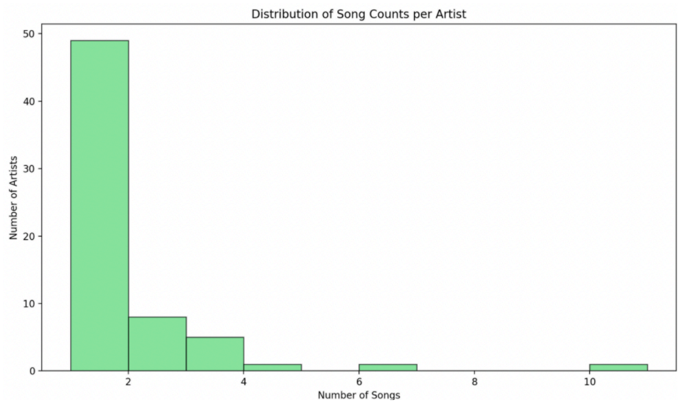
- **Events Table**: Includes the name of the event, the artist ID number (linked to the Artists table), the city where the event is, and the state where the event is

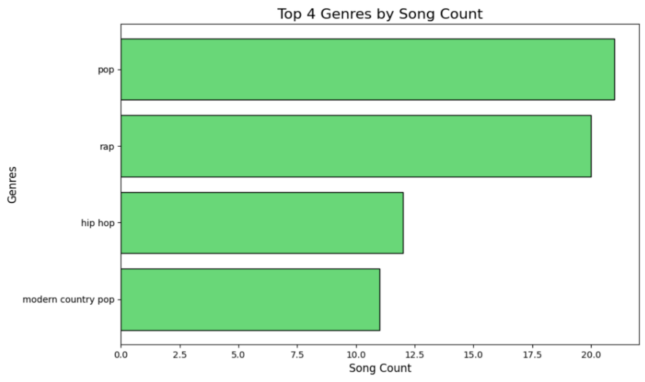
Problems Faced

We encountered a variety of problems throughout this project. The first issue we encountered was with accessing the official Billboard Hot 100 playlist on Spotify. We initially aimed to analyze data from the official playlist, but encountered a significant challenge due to restricted access to public playlists through the Spotify API, which requires proper user permissions to retrieve data from playlists that aren't owned by the user. Without the necessary access, we were unable to access data from the official playlist. To work around this limitation, we manually created a copy of the Billboard Hot 100 playlist on our own Spotify account. This allowed us to access the same information and data we would have obtained from the official playlist, simply by using the copy we had created on our own account. Another significant challenge we came across was with the Ticketmaster API, as many of the events added to the table lacked price details. The API would return "null" for missing prices, and since we were specifically analyzing concert pricing trends, these events couldn't be used. To solve this, we excluded any entries with null or invalid price information from the table. While this drastically reduced the number of events available for analysis, it more importantly ensured the data we used was accurate and relevant to our goal.

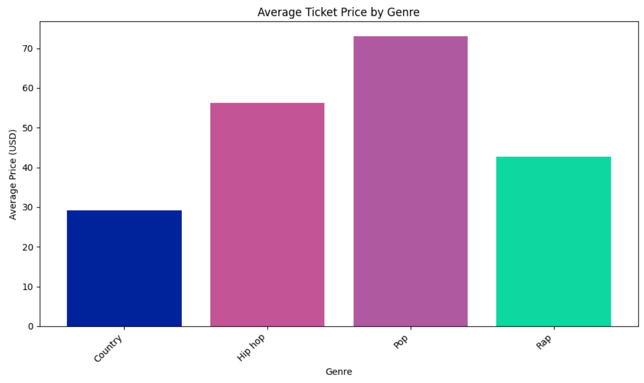
Visualizations

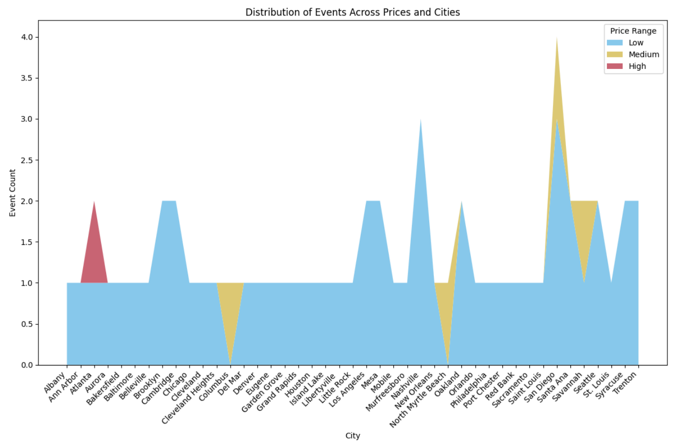
Spotipy API



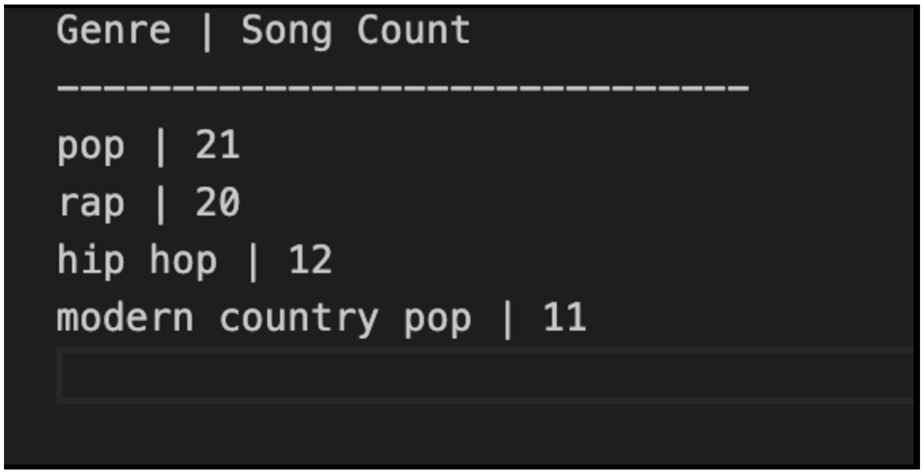


Ticketmaster API

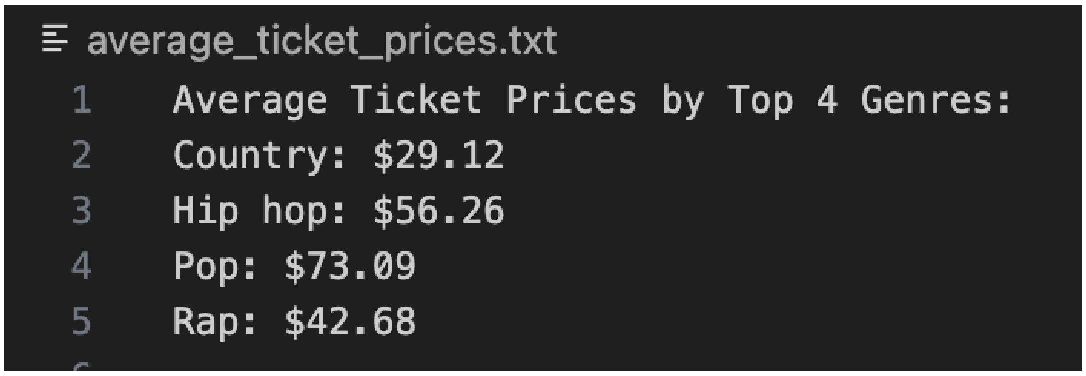




Calculations from the Data in the SQLite Database



```
Genre | Song Count
------------------------------
pop | 21
rap | 20
hip hop | 12
modern country pop | 11
```



average_ticket_prices.txt

```
Average Ticket Prices by Top 4 Genres:
Country: $29.12
Hip hop: $56.26
Pop: $73.09
Rap: $42.68
```

Instructions for Running the Code

There are two different files, one for each API. To run the code for the **Spotify API** first open the file titled spotifyfinal.py and then run the code. Then, you will be prompted with the options: “What would you like to do?” followed by two choices:

1. Add 25 rows to the database
2. Calculate and visualize data

If you type **1**, the program will ask for a Spotify playlist URL or ID (for this project we used the playlist ID “1yuH8vUahw61OKnxo3Ykym”), and the script will fetch 25 songs from the playlist, store them in the SQLite database, and save your progress. Each time the user runs the code and types **1**, 25 more songs (and their respective artists) are added to the database until all 100 songs have been loaded. The user should run the code and type **1** a total of 4 times to load the entirety of the data. The user should then run the code a 5th time, but this time type **2**. The program will then process the stored data and display a second menu:

1. Top 4 Genres by Song Count
2. Distribution of Song Counts per Artist

If you type **1**, the script will generate a horizontal bar chart showing the top 4 genres based on song count. If you type **2**, it will create a histogram displaying the distribution of song counts per artist. The visualizations will be displayed on your screen and saved as image files (top_genres_by_song_count_cleaned.png and artist_popularity_histogram.png) in the project folder.

Run the file titled ticketmaster.py to run the **Ticketmaster API** code. The user will then be prompted to enter the genre of events they want to analyze (e.g., Pop, Rock, Jazz). We entered each of the top 4 genres by song count that we got from running the Spotify API, but the code allows the user to enter any genre of their choosing as many times as they want. Each time the code is run, it fetches data for 25 or fewer events related to the specified genre from the Ticketmaster API, stores the data in a SQLite database, and generates visualizations for the events currently in the database. These include a bar chart for average ticket prices by genre and a stack plot showing the distribution of events across price ranges. The visualizations will be displayed on your screen and saved as image files (genre_price_bar_chart.png and price_city_distribution_stackplot.png) in the project folder. Everytime the user runs the code, the visualizations update, incorporating the data newly added to the database. For example, running the code once generates visualizations for the first 25 events. Running it a second time updates the database to include 50 events, and the visualizations now reflect this larger dataset. By running the code 4–5 times, you can collect data for 100+ events, with the final visualizations representing the entire dataset.

Function Documentation

spotifyfinal.py

- initialize_spotify_client
    - Initializes the Spotify client using OAuth to securely access the Spotify API.
    - Input:
        - client_id(str): The client ID for the Spotify application
        - client_secret(str): The client secret for the Spotify application
        - redirect_uri(str): The redirect URI set in the Spotify application
    - Output:
        - Returns the authenticated Spotify client
- fetch_playlist_tracks
    - Fetches tracks from a Spotify playlist using the Spotify API and returns the track information along with the total number of tracks in the playlist.
    - Input:
        - sp: An authenticated Spotify API client created using the Spotipy library

        - playlist_id: The unique identifier of the Spotify playlist from which tracks are to be retrieved
        - limit (default = 25): Specifies the maximum number of tracks to fetch in a single request
        - offset (default = 0): Specifies the starting position for fetching tracks
    - Output:
        - tracks: A list of dictionaries, each containing:
            - song_name: The name of the song
            - artists: A list of the names of artists for that song
        - total: The total number of tracks in the playlist, retrieved from the API response (results.get('total', 0)).
- initialize_database
    - Creates and initializes an SQLite database by creating two tables, Artists and Songs, to store playlist data
    - Input: None
    - Output:
        - conn (sqlite3.Connection): A connection object for interacting with the SQLite database
        - cur (sqlite3.Cursor): A cursor object for executing SQL commands within the database
- save_artist_to_database
    - Inserts an artist's details (name, genres, and popularity) into the Artists table in the SQLite database and returns the unique ID of the artist.
    - Input:
        - cur (sqlite3.Cursor): A cursor object used to execute SQL queries
        - artist_name (str): The name of the artist
        - genres (list of strs): A list of genres associated with the artist
        - popularity (int): The popularity score of the artist
    - Output:
        - artist_id (int): The unique ID of the artist in the database, retrieved after the artist is inserted or ignored if they already exist.
- save_song_to_database
    - Inserts a song's details (name and associated artist ID) into the Songs table in the database.
    - Input:
        - cur (sqlite3.Cursor): A cursor object used to execute SQL queries.
        - song_name (str): The name of the song.
        - artist_id (int): The unique ID of the artist associated with the song.
    - Output:
        - None

- process_and_store_playlist_data
  - Processes playlist tracks, retrieves artist details, and stores them along with song data in the database.
  - Input:
    - sp: An authenticated Spotify API client created using the Spotipy library
    - playlist_id (str): The unique ID of the Spotify playlist to fetch tracks from.
    - cur (SQLite cursor): A database cursor object used to execute SQL commands.
    - conn (SQLite connection): A connection object representing the database used to commit changes to the database after processing and storing the data.
    - limit (int, default=25): Specifies the number of tracks to fetch in a single API call
    - offset (int, default=0): Specifies the starting point for fetching tracks from the playlist.
  - Output:
    - A boolean (status) indicating success or failure and the total number of playlist tracks (total).
      - status (boolean)
        - True: Indicates that the function successfully fetched and processed some tracks from the playlist.
        - False: Indicates that no tracks were fetched (e.g., the playlist is empty, or all tracks have been processed).
      - total (int)
        - The total number of tracks in the Spotify playlist (as fetched by the fetch_playlist_tracks function).
- process_data_top_genres
  - Processes data from the database to calculate and rank the top genres based on the number of songs associated with each genre.
  - Input:
    - cur (sqlite3.Cursor): A cursor object used to execute SQL queries on the database.
  - Output:
    - sorted_genres (list of tuples): A list of tuples where each tuple contains a genre (str) and its corresponding song count (int).
- process_artist_popularity
  - Computes the popularity and song count for each artist based on database entries.
  - Input:
    - cur (sqlite3.Cursor): A cursor object used to execute SQL queries on the database.

    - Output:
        - results (list of tuples): A list of tuples where each tuple contains:
            - Artists.name (str): The name of the artist.
            - song_count (int): The number of songs associated with the artist.
            - popularity (int): The popularity score of the artist.
- write_top_genres_to_file
    - Writes the top genres and their respective song counts to a text file.
    - Input:
        - data (list of tuples): A list where each tuple contains:
            - genre (str): The name of the genre.
            - song_count (int): The number of songs associated with the genre.
    - Output:
        - None: Writes the genre data to a file named top_genres_data.txt
- write_artist_popularity_to_file
    - Writes artist popularity data (artist name, song count, and popularity) to a text file.
    - Input:
        - data (list of tuples): A list where each tuple contains:
            - artist_name (str): The name of the artist.
            - song_count (int): The number of songs by the artist.
            - popularity (int): The artist's popularity score.
    - Output:
        - None: Writes the artist data to a file named artist_popularity.txt
- create_bar_chart_top_genres
    - Generates a horizontal bar chart visualizing the top 4 genres by song count and saves it as an image.
    - Input:
        - data (list of tuples): A list where each tuple contains:
            - genre (str): The name of the genre.
            - song_count (int): The number of songs associated with the genre.
    - Output:
        - None: Saves the bar chart as top_genres_by_song_count_cleaned.png and displays it.
- create_histogram_artist_popularity
    - Creates a histogram displaying the distribution of song counts per artist and saves it as an image.
    - Input:
        - data (list of tuples): A list where each tuple contains:
            - song_count (int): The number of songs associated with an artist (extracted from the input data).
        - Output:

            - None: Saves the histogram as artist_popularity_histogram.png and displays it.

- fetch_and_store_25_rows
    - Fetches up to 25 songs from a Spotify playlist and stores them in the database, updating progress for pagination.
    - Input:
        - sp: An authenticated Spotify API client created using the Spotipy library.
        - playlist_id (str): The ID of the playlist to fetch data from.
        - conn (sqlite3.Connection): A connection object to interact with the database.
        - cur (sqlite3.Cursor): A cursor object used to execute SQL queries.
    - Output:
        - None: Updates the database with fetched data and manages the offset for subsequent fetches.
- calculate_and_visualize
    - Processes data to generate text files and visualizations based on user input, such as genre or popularity distribution.
    - Input:
        - conn (sqlite3.Connection): A connection object to interact with the database.
        - cur (sqlite3.Cursor): A cursor object used to execute SQL queries.
    - Output:
        - None: Writes processed data to text files and generates visualizations based on user input.
- extract_playlist_id
    - Extracts the playlist ID from a Spotify playlist URL or input string.
    - Input:
        - playlist_url_or_id (str): A Spotify playlist URL or a raw playlist ID.
    - Output:
        - playlist_id (str): The extracted playlist ID.
- main
    - The main function that serves as the entry point of the program. Allows the user to fetch and store data or calculate and visualize data.
    - Input:
        - User Input:
            - choice (str): Determines whether to fetch and store 25 rows of playlist data ("1") or to calculate and visualize data ("2").
            - If choice is "1", prompts for:
                - raw_playlist_id (str): A Spotify playlist URL or ID entered by the user.

        - Predefined Configuration:
            - CLIENT_ID (str): Spotify API client ID.
            - CLIENT_SECRET (str): Spotify API client secret.
            - REDIRECT_URI (str): Spotify API redirect URI.
    - Output:
        - None: Performs actions such as database updates, visualization generation, and data writing.

tickmaster.py

- initialize_database
    - Initializes a SQLite database and creates two tables, Artists and Events, to store artist and event data.
    - Input:
        - None (database creation and table setup are predefined).
    - Output:
        - (conn, cur): A tuple containing the database connection (conn) and cursor (cur) objects.
- save_artist_to_database
    - Inserts an artist's name, genre, and ticket price range into the Artists table and returns the artist's ID.
    - Input:
        - cur (sqlite3.Cursor): The cursor used to execute SQL queries.
        - artist_name (str): The name of the artist.
        - genre (str): The genre of the artist.
        - min_price (float): The minimum ticket price for the artist’s events.
        - max_price (float): The maximum ticket price for the artist’s events.
    - Output:
        - int: The ID of the artist that was saved or retrieved.
- fetch_and_store_data_by_genre
    - Fetches event data from the Ticketmaster API for a specified genre, processes the data, and stores it in the database.
    - Input:
        - genre (str): The genre entered by the user for which to fetch event data (e.g., "Pop", "Rock").
    - Output:
        - None: Saves event and artist data to the database and updates the page number for API fetching.
- save_event_to_database
    - Inserts event details, including the name, artist ID, city, and state, into the Events table.

- Input:
  - cur (sqlite3.Cursor): The cursor used to execute SQL queries.
  - event_name (str): The name of the event.
  - artist_id (int): The ID of the artist associated with the event.
  - city (str): The city where the event is held.
  - state (str): The state where the event is held.
- Output:
  - None: Saves event data to the database.

- analyze_genre_prices
  - Analyzes the database to calculate the average ticket price by genre and the distribution of events across cities and price ranges, saving the results to text files.
  - Input:
    - None: Uses data already stored in the database.
  - Output:
    - None: Writes analysis data to text files and generates visualizations.
- main
  - The main function that drives the program. Accepts user input for the genre, fetches event data, stores it in the database, and performs data analysis and visualization.
  - Input:
    - genre (str): A user input specifying the genre for event data fetching (e.g., "Pop", "Rock").
  - Output:
    - None: Based on user input, either fetches event data for a genre or analyzes and visualizes data.

Documentation of Resources Used

Date	**Issue Description**	**Location of Resource**	**Result** **(did it solve the issue)**
11/20/24	Fetching Spotify API data	Spotify Web API Documentation	Yes, solved fetching data issues
11/20/24	OAuth with Spotify	Spotify OAuth Guide	Yes, enabled successful implementation of the authentication code
11/21/24	Scope needed to	Spotify Scopes Guide	Yes, granted access to the

	implement the authorization grant		private playlist we made for this project
11/21/24	Top 100 Songs	Billboard Top 100	Yes, we used it to make the playlist of the top 100 songs
11/24/24	Fetching Ticketmaster API data	Ticketmaster Discovery API Guide	Yes, allowed the program to fetch event data based on a specific genre
11/24/24	Needed help making sure the program did not allow duplicate data in ticketmaster.py	ChatGPT → explained that our 'INSERT OR IGNORE' statement was in the wrong place, thus allowing duplicate data, and showed where to put it instead	Yes, ensured that the database will not insert duplicate entries for artists or events based on their unique fields by moving the statement to the correct place
11/25/24	Unsure how to visualize data using a stack plot	Matplotlib Stackplot Guide	Yes, gave clear instructions on how to create a stack plot that we then used to visualize ticket prices and event distributions
12/01/24	Generating random colors for plots	ChatGPT → suggested using Python's built-in random module in combination with Matplotlib. Specifically, it showed how to generate random hexadecimal color codes, which could be applied to plot elements in Matplotlib.	Yes, successfully added random color generation for plot elements