[SI 206] Final Project Report

Ashley Bouse, Terry Qi
https://github.com/ashlooz/SI206_FinalProject

Goals

Planned Goals: Use Plotly, create bar charts for favorite genre + artists from database

- Planned APIs/Websites: Spotify + one other related API to gather additional information
- Planned Data to Collect:
    - User playlists
    - Track information (i.e. title, artist name, genre)
    - Personalized song recommendations (from Spotify's recommendation engine)

Goals Achieved: Used matplotlib, displayed visualizations and calculations using song data pulled from Spotify and Last.fm APIs

- Actual APIs/Websites Used: Spotify API (spotipy) + Last.fm API
- Actual Data Collected:
    - Spotify track details + audio features
        - Artist name
        - Song title
        - Popularity
        - Valence
        - Danceability
        - Energy
    - Last.fm play count values

Problems Faced

1. **Duplicate string data:** We had to ensure there were no repetitive strings in the database.

- Solution: By using the 'INSERT or IGNORE' alongside 'name TEXT UNIQUE' ensured that any sort of attempt to add an existing artist name to the database was ignored.

2. **Shared integer keys:** There was a learning curve figuring out how to create a database that related a shared key between the Artist and Song tables.

- Solution: This was solved by implementing a system where each artist name was associated with a unique ID in the Artist table. The ID (artist_id) functioned as a foreign key in the Song table. This method made sure that we had consistency when entering the data while also preventing duplicate string data in the process.

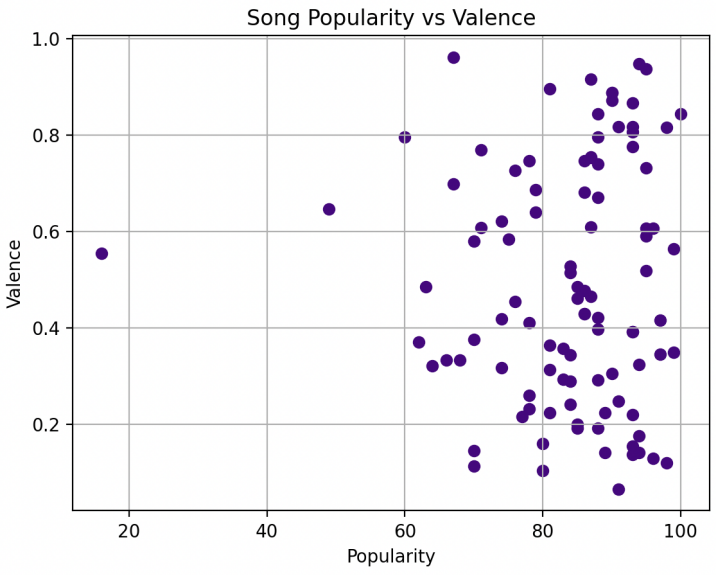
3. **429 Errors:** Figuring out how to handle API rate limits was challenging, especially with Spotify API.
- Solution: Reducing the frequency of requests and being more mindful of time lapsed in between requests to the APIs.
4. **Limiting the data to 25:** Figuring out how to limit the items being stored each time the program is run was difficult at first. The code would initially stored all 100 items into the database after one run.
- Solution: Implementing a logic in order to keep track of the number of entries that were already sitting in the database after a run. From here, we only processed and inserted a maximum of 25 new items each time the code was ran. This was achieved by utilizinf the 'SELECT COUNT(*) FROM Song' in order to figure out the range (of indexes) for the data that would be processed during each run.

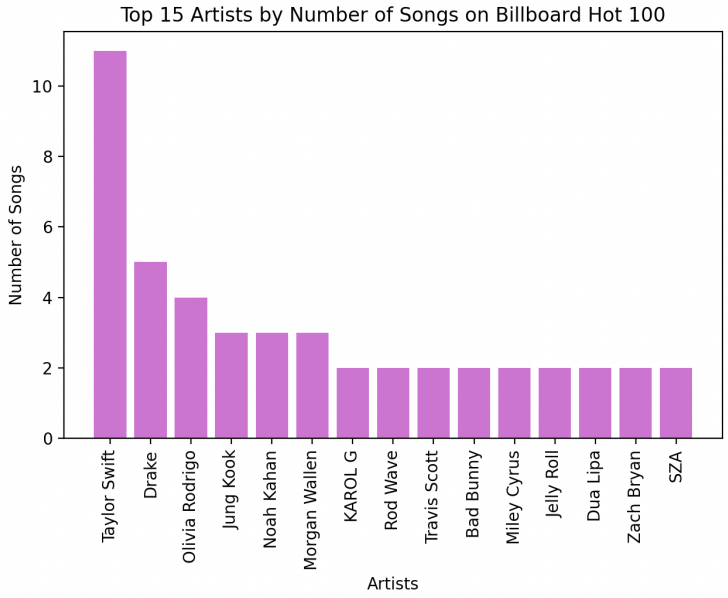
Calculations

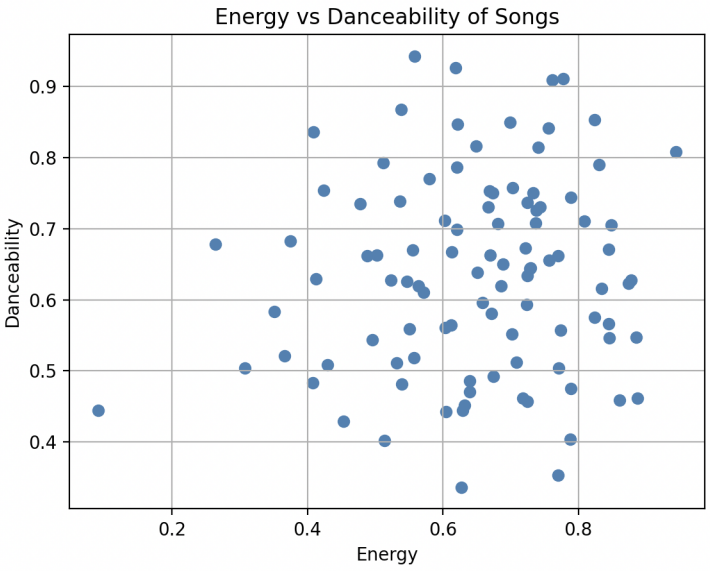
calculations_results.txt

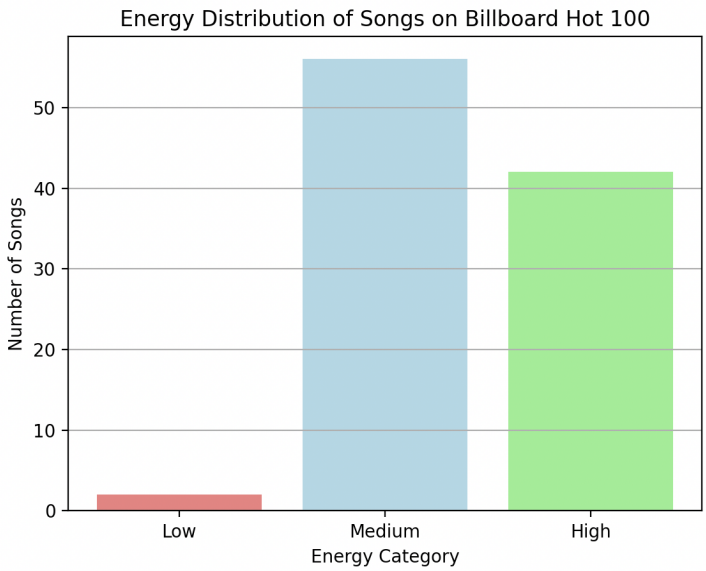
```
1  The valence standard deviation for the songs in the Billboard Top 100 is 0.24986038492637355
2  The average valence of songs in the Billboard Top 100 is 0.4917419999999999
3  The average play count of songs in the Billboard Top 100 is 4096427.4
```

Visualizations (4)









Instructions for Running Code

1. Download and open up “data_visualizations.py” and “data_calculations.py” in your integrated development environment (IDE).
2. Generate your key for Spotify API:
    a. Go Spotify for Developers
    b. Click on Dashboard.
    c. Write down your Spotify ID and your Spotify secret key.
3. Generate your key for Last.FM API:
    a. Apply for a Last.fm API Key
    b. Write down your Last.fm API key.
4. Navigate to “data_collection.py”
    a. Replace the string on line 122 with your Spotify ID
    b. Replace the string on line 123 with your Spotify secret key
    c. Replace the string on line 124 with your Last.FM API key
5. Go to “data_collection.py” and click “Run Code” to run the python file.
    a. Billboard_hot_100.db will be created in the Explorer tab.
6. Now run “data_collection.py” three more times to populate the database with 100 items.
7. Navigate to “data_visualization.py” and click “Run Code.”
    a. You should see four visualizations display (only one will display at a time).
8. Navigate to “calculations.txt” to view the calculations (average play count, valence standard deviation, average valence)

- Note: If you would like to see these functions and visualizations run on another Spotify playlist, you can replace the ‘billboard_playlist_id’ on line 25 in ‘data_collection.py’ with any Spotify playlist ID of your preference.

Function Documentation

Data Collection:

```
def setup_database(database_name):
```

- Input: name of the database
- Output: SQL cursor and connection objects

```
def spotify_data_retrieval(spotify_id, spotify_secret):
```

- Input: Spotify client id (‘spotify_id’) and client secret (‘spotify_secret’)
- Output: four lists *track_data*, *valence_data*, *danceability_data*, and *energy_data*, each list contains respective information for every songs in the playlist

```
def create_artist_and_song_tables(cursor, connection):
```

- Input: SQL ‘cursor’ and ‘connection’ to the SQLite database
- Output: creates 'Artist' and 'Song' tables in the database
    - The Artist table has fields for artist ID and name, while the Song table includes song title, Spotify ID, artist ID, popularity, valence, danceability, energy, and Last.fm play count.

```
def get_lastfm_play_count(track_title, artist_name, lastfm_api_key):
```

- Input: ‘track_title’, ‘artist_name’, ‘lastfm_api_key’
- Output: returns play count (from Last.fm API) as an int

```
def insert_data_into_tables(track_data, valence, danceability, energy, cursor,
connection, lastfm_api_key):
```

- Input: ‘track_data’ , ‘valence’ (list), ‘danceability’ (list), ‘energy’ (list) – (these are all different audio features for the tracks), SQL ‘cursor’, ‘connection’, and ‘lastfm_api_key’
- Output: Adds the relevant data into the 'Artist' and 'Song' tables in the database. It links the artist and song data + includes the audio features and play counts.

Data Visualizations:

```
def popularity_valence_visual(cursor):
```

- Input: SQL ‘cursor’ which interacts with the db.
- Output: Creates and displays a scatter plot that shows the relationship between popularity of songs in the playlist to their respective valences.

```
def plot_artist_distribution(cursor):
```

- Input: SQL ‘cursor’
- Output: Creates and displays a bar graph that displays the distribution of the top 15 artists with songs in the playlist.

```
def energy_danceability_visual(cursor):
```

- Input: SQL ‘cursor’
- Output: Creates and displays a scatter plot visualizing the relationship between the energy and danceability of songs.

```
def energy_distribution_visual(cursor):
```

- Input: SQL ‘cursor’
- Output: Produces a bar chart showing the distribution of songs based on their energy levels (categorized into low, medium, and high).

Data Calculations:

```
def get_average_valence(cursor):
```

- Input: SQL ‘cursor’
- Output: The average valence of the songs in the Spotify playlist.

```
def get_valence_std_dev(cursor):
```

- Input: SQL ‘cursor’
- Output: The standard deviation of valence values of songs in the Spotify playlist.

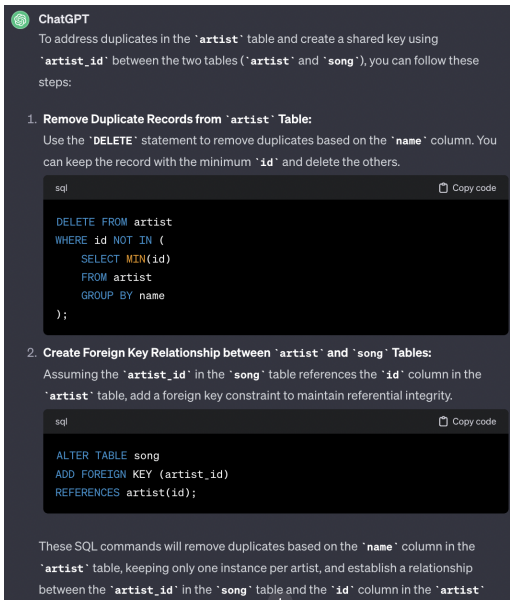
```
def get_average_play_count(cursor):
```

- Input: SQL ‘cursor’
- Output: Calculates and returns the average play count for the songs in the database.

Documentation of Resources

Date Issue	Description	Location of Resource	Result (did it solve the issue?
12/7	Initial resources for finding APIs + Resources for learning about spotipy and lastfm APIs	https://developer.spotify.com/ https://github.com/public-apis/public-apis https://www.last.fm/api#getting-started https://developer.spotify.com/documentation/web-api https://developer.spotify.com/documentation/web-api/reference/get-an-artist	Yes, learned more in depth about spotipy and lastfm APIs

		https://www.last.fm/api/authentication https://www.last.fm/api/intro	
12/9	429 Error	https://community.spotify.com/t5/Spotify-for-Developers/Too-Many-Requests-429-Error-blocked-for-13-hours/td-p/5664805	Yes - but had to start a new project on the Spotify developers dashboard + generate a new API id and secret key
12/10	Using spotipy and oauth_manager + import SpotifyClientCredentials as SCC	https://spotipy.readthedocs.io/en/2.22.1/#client-credentials-flow	Yes

12/10	How do I get rid of duplicate string data in the name column of the Artist table in SQLite?	ChatGPT
		 
		ChatGPT
		To address duplicates in the `artist` table and create a shared key using `artist_id` between the two tables (`artist` and `song`), you can follow these steps:
		1. **Remove Duplicate Records from `artist` Table:**
		Use the `DELETE` statement to remove duplicates based on the `name` column. You can keep the record with the minimum `id` and delete the others.
		sql Copy code
		`DELETE FROM artist`
		`WHERE id NOT IN (`
		`SELECT MIN(id)`
		`FROM artist`
		`GROUP BY name`
		`);`
		2. **Create Foreign Key Relationship between `artist` and `song` Tables:**
		Assuming the `artist_id` in the `song` table references the `id` column in the `artist` table, add a foreign key constraint to maintain referential integrity.
		sql Copy code
		`ALTER TABLE song`
		`ADD FOREIGN KEY (artist_id)`
		`REFERENCES artist(id);`
		These SQL commands will remove duplicates based on the `name` column in the `artist` table, keeping only one instance per artist, and establish a relationship between the `artist_id` in the `song` table and the `id` column in the `artist`
			No - DELETE statement addition did not work when implemented into the 'Artist' table code

12/11	Same problem: How do I get rid of duplicate string data in the name column of the Artist table in SQLite?	https://www.w3schools.com/sql/sql_unique.asp	Yes - added UNIQUE to the 'name' column in the Artist table to prevent duplicate artist names from being inserted. Artist_id now properly functions as the shared key

		SQL UNIQUE Constraint on CREATE TABLE
		The following SQL creates a `UNIQUE` constraint on the "ID" column when the "Persons" table is created:
		SQL Server / Oracle / MS Access:
		`CREATE TABLE Persons (`
		`  ID int NOT NULL UNIQUE,`
		`  LastName varchar(255) NOT NULL,`
		`  FirstName varchar(255),`
		`  Age int`
		`);`
		MySQL:
		`CREATE TABLE Persons (`
		`  ID int NOT NULL,`
		`  LastName varchar(255) NOT NULL,`
		`  FirstName varchar(255),`
		`  Age int,`
		`  UNIQUE (ID)`
		`);`
		To name a `UNIQUE` constraint, and to define a `UNIQUE` constraint on multiple columns, use the following SQL syntax:	

12/11	Circular import	https://levelup.gitconnected.com/escaping-the-python-import-maze-tips-for-avoiding-circular-imports-in-python-9046b0be5cb	Yes

12/11	I want to restructure my code from one main file into three files but my import statements are causing issues.	ChatGPT
		ChatGPT
		Certainly! Based on your code structure, here are the corrected import statements for each file:
		data_operations.py:
		python Copy code
		`import json`
		`import sqlite3`
		`import os`
		`import numpy as np`
		`import spotipy`
		`import requests`
		`from spotipy.oauth2 import SpotifyClientCredentials as SCC`
		`import matplotlib.pyplot as plt`
		visualizations.py:
		python Copy code
		`import sqlite3`
		`import matplotlib.pyplot as plt`
		spotify_lastfm_api.py:
		python Copy code
		`from data_operations import (`
		`  setup_database,`
		`  spotify_data_retrieval,`
		`  lastfm_data_retrieval,`
		`  create_artist_and_song_tables,`
		`  insert_data_into_tables`
		`)`
		Ensure that all the necessary functions called in each file are properly defined within the respective modules (`data_operations.py`, `visualizations.py`) to avoid any undefined function errors.
			Yes and no - The way the response corrected the import statements is correct but I still needed to add “from visualizations import plot_artist_distribution, popularity_valence_visual” in main.py for proper functionality/for the code to run

12/12	How can I restructure my code to fit the project	ChatGPT	Yes - now the code can run 4x on the data_collection.py to

	requirements: “How can I structure one file that independently runs to create tables and add data + another file that runs independently for visualizations”	**ChatGPT** To reorganize your code into two separate files as per your requirements, we will create one file for data collection and insertion into the database, and another file for data visualization and analysis. Each file will have its `main()` function to orchestrate the process. **File 1: Data Collection and Database Management (`data_collection.py`)** This file will handle the setup of the SQLite database, data retrieval from Spotify and Last.fm APIs, creation of database tables, and insertion of data into these tables. It will ensure that only 25 items are added to the database each time the script is run.	collect the 100 items and run 1x on data_visualizations.py to output the plots/txt files
	What is a scrobble… And what is causing every single item for “play count” data column in the Songs table to default to 0.	https://www.businessinsider.com/guides/tech/what-is-last-fm-scrobbling **Last.fm uses 'scrobbling' to track what music you listen to** Open any app with Last.fm integration, and you're going to see the word "scrobble" more times than you count. At this point, it's probably more recognizable than the name "Last.fm." In short, **"scrobble" is just the word that Last.fm uses to describe how it automatically tracks what music you listen to, and when**. The act of tracking your music with Last.fm is called "scrobbling," and apps that connect to Last.fm are sometimes called "scrobblers." When you listen to a song, Last.fm scrobbles the song by recording the song's title, its artist, its genre, and when you listened to it. All this information gets saved to your account, and Last.fm uses it to both generate lists of your favorite songs (both recently and of all-time) and **ChatGPT** In Last.fm, "scrobbling" and "play count" are two related concepts, but they refer to slightly different things: 1. **Scrobble**: A "scrobble" on Last.fm is a record of a user listening to a track. It includes information about the track (such as the artist, album, and track name), and the time the track was played. Scrobbling is the act of sending this information to Last.fm from a music player or app. The term is unique to Last.fm and comes from the original software application, "Audioscrobbler," which was developed to automatically log (or scrobble) what a user was listening to. 2. **Play Count**: The "play count" on Last.fm refers to the total number of times a particular track has been scrobbled by all users on the platform. It's a cumulative count that gives an idea of the overall popularity or frequency of plays a track has received from the Last.fm user community. In summary, a scrobble is an individual instance of a user listening to a track, while the play count is the total number of these instances (scrobbles) for a track across all users. Scrobbling contributes to the play count: each time a track is scrobbled by any user, its play count increases by one.	Yes - I had accidentally been trying to add “user scrobbles” to the table instead of overall song “play counts.” I figured this out because all of the values had been defaulted to zero, since it was referencing individual play counts (aka “scrobbles”) instead of overall plays from all users.