# API Audio Fusion Final Project Report

Ethan Casler, Hady Farhat, Kenyatta Washington

SI 206: Data Oriented Programming

Dr. Barbara Ericson

April, 30, 2024

## **Project Goals**:

### Initial Goal:

Our initial goal with this project was to explore the different usage patterns for different music streaming platforms. For example, is R&B more popular on Spotify than Last FM? It would also be a cool thing, in addition to this cross comparison goal, to have a database archive for this information to be able to be loaded into. We would like to calculate the number of songs in each genre and also gather data to create relevant box plots and histograms showing this data. Additionally, we could extract all the relevant data( genre, track name, artists, popularity/play count) and put it on a tree map to show relevant frequencies and relationships among the data. We would utilize API data from Spotify and either Last FM https://www.last.fm/api/intro) or Soundcloud (https://developers.soundcloud.com/docs/api/guide), as well as use BeautifulSoup retrieving more data from spotify (https://open.spotify.com)

## **Goals Achieved:**

For our project, we conducted a comparison of music streaming services, focusing on extracting songs, identifying popular tracks, and exploring existing genres. We also investigated naming conventions for track, album, and radio recommendations. Specifically, we aimed to extract a sample of song data from Spotify, including genre, track name, artists, and popularity data. For web scraping, while initially our objective was to capture important row header titles such as "Made for Ethan" and "Popular Radio" on Spotify, we switched to doing web scraping for Soundcloud songs, specifically scraping the play counts for our 100 songs.

We were able to create a beautiful database without any string duplications, using extra tables to make unique ids as stand-ins for genres, artists, and albums. We were also able to make data visualizations, in the forms of box plots, which show the trends, variability and averages of artists vs their platform popularity (via popularity index for Spotify* and play count totals for Last FM and Soundcloud).

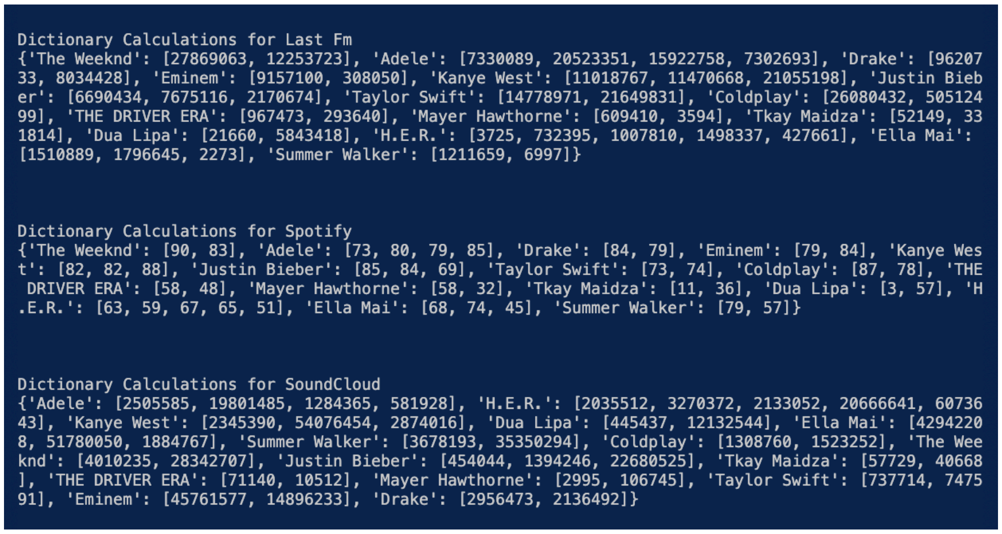
*According to loudlab.org, Spottify's "Popularity Index is majorly determined by recent stream count, other factors like save rate, the number of playlists, skip rate, and share rate can indirectly bump up or push down a song's popularity index" (https://www.loudlab.org/blog/spotify-popularity-leverage)

## **Problems We Faced:**

We faced challenges in comprehending Spotify's datasets and its proprietary coding system, which made it difficult to grasp how Spotify operates and is structured. This lack of understanding posed difficulties in anticipating outcomes. Additionally, we encountered issues when attempting to implement specific songs and artists, as the code did not yield any results. Consequently, we had to conduct further research to locate the required data.(Hidden Data)

## **Calculations & Visualizations:**

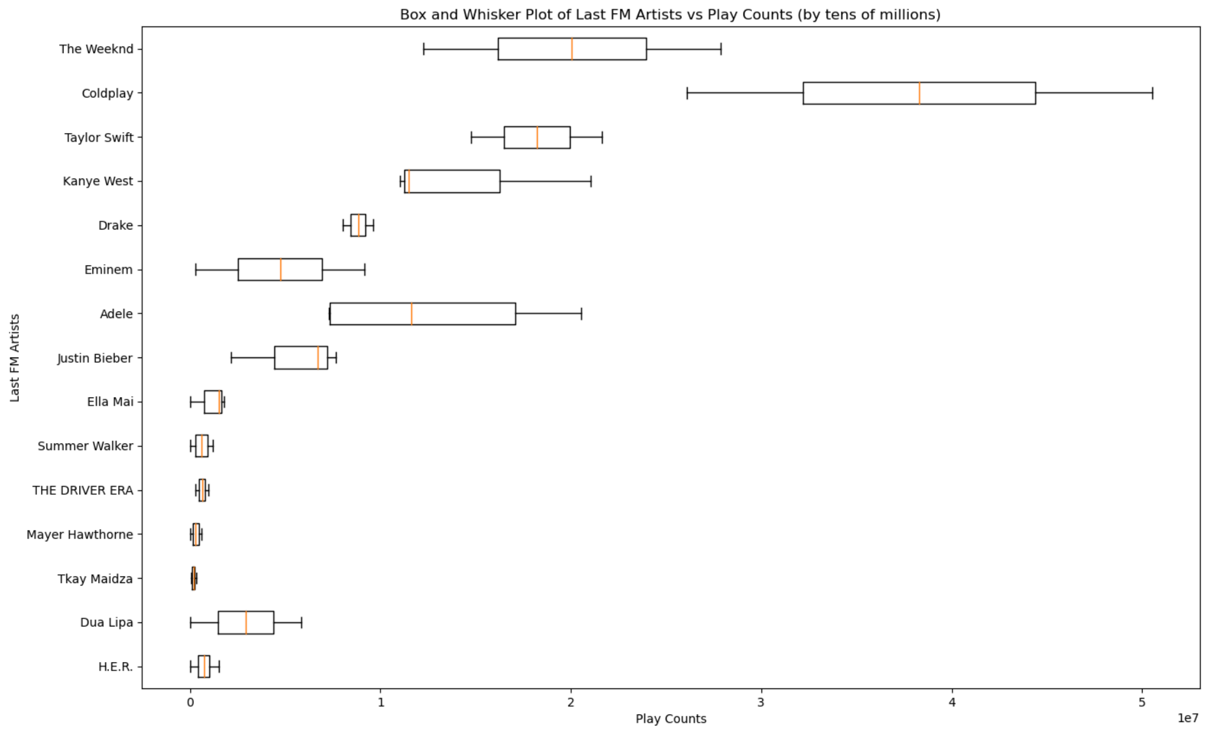
## Calculations for the Data



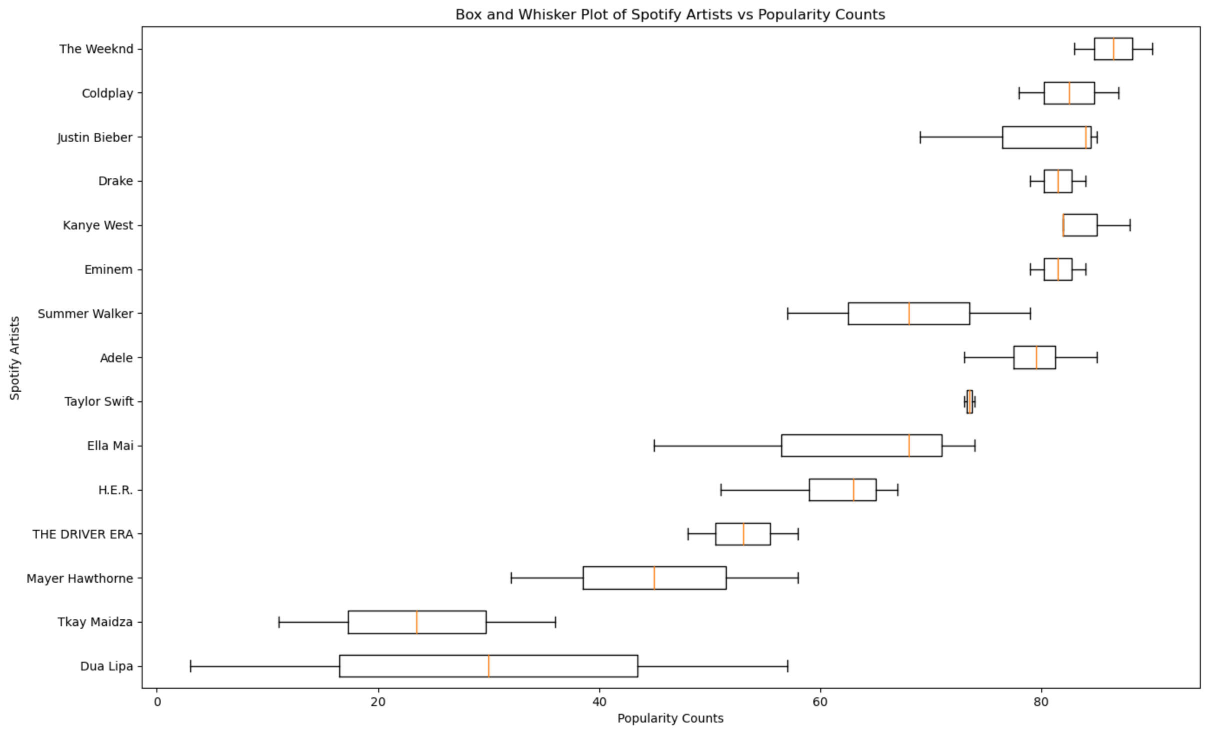
```
Dictionary Calculations for Last Fm
{'The Weeknd': [27869063, 12253723], 'Adele': [7330089, 20523351, 15922758, 7302693], 'Drake': [96207
33, 8034428], 'Eminem': [9157100, 308050], 'Kanye West': [11018767, 11470668, 21055198], 'Justin Bieb
er': [6690434, 7675116, 2170674], 'Taylor Swift': [14778971, 21649831], 'Coldplay': [26080432, 505124
99], 'THE DRIVER ERA': [967473, 293640], 'Mayer Hawthorne': [609410, 3594], 'Tkay Maidza': [52149, 33
1814], 'Dua Lipa': [21660, 5843418], 'H.E.R.': [3725, 732395, 1007810, 1498337, 427661], 'Ella Mai':
[1510889, 1796645, 2273], 'Summer Walker': [1211659, 6997]}


Dictionary Calculations for Spotify
{'The Weeknd': [90, 83], 'Adele': [73, 80, 79, 85], 'Drake': [84, 79], 'Eminem': [79, 84], 'Kanye Wes
t': [82, 82, 88], 'Justin Bieber': [85, 84, 69], 'Taylor Swift': [73, 74], 'Coldplay': [87, 78], 'THE
 DRIVER ERA': [58, 48], 'Mayer Hawthorne': [58, 32], 'Tkay Maidza': [11, 36], 'Dua Lipa': [3, 57], 'H
.E.R.': [63, 59, 67, 65, 51], 'Ella Mai': [68, 74, 45], 'Summer Walker': [79, 57]}


Dictionary Calculations for SoundCloud
{'Adele': [2505585, 19801485, 1284365, 581928], 'H.E.R.': [2035512, 3270372, 2133052, 20666641, 60736
43], 'Kanye West': [2345390, 54076454, 2874016], 'Dua Lipa': [445437, 12132544], 'Ella Mai': [4294220
8, 51780050, 1884767], 'Summer Walker': [3678193, 35350294], 'Coldplay': [1308760, 1523252], 'The Wee
knd': [4010235, 28342707], 'Justin Bieber': [454044, 1394246, 22680525], 'Tkay Maidza': [57729, 40668
], 'THE DRIVER ERA': [71140, 10512], 'Mayer Hawthorne': [2995, 106745], 'Taylor Swift': [737714, 7475
91], 'Eminem': [45761577, 14896233], 'Drake': [2956473, 2136492]}
```

## Data Visualizations

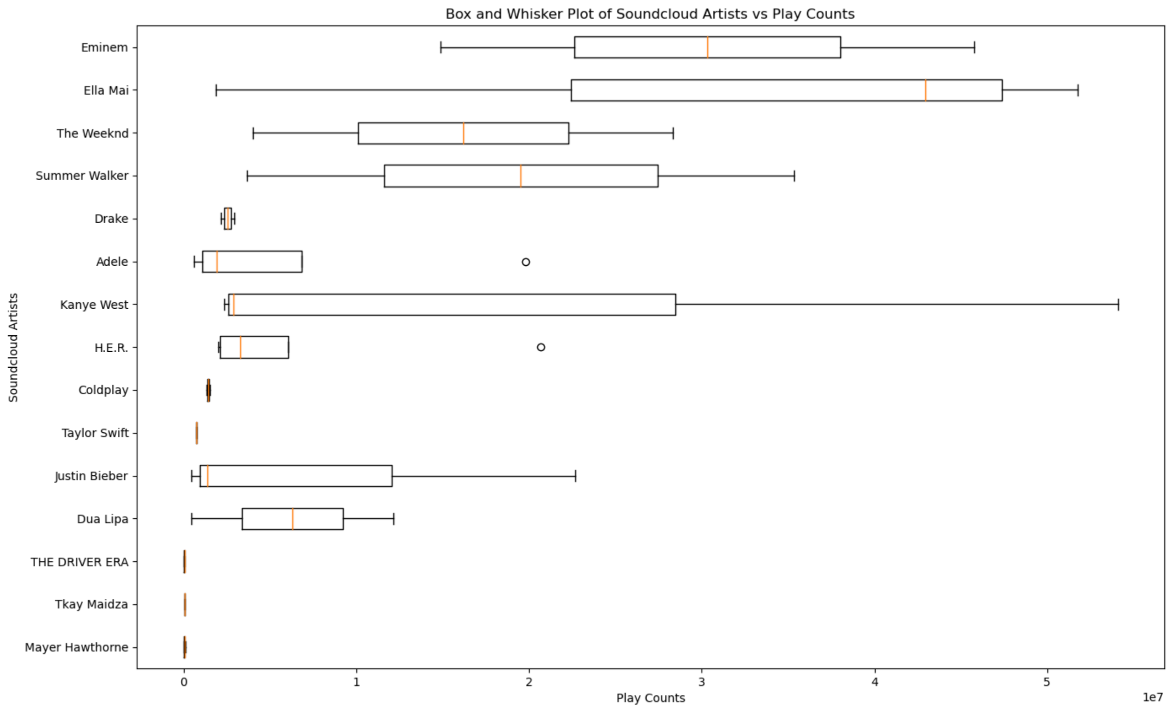
### Last FM:



### Spotify:


Box and Whisker Plot of Spotify Artists vs Popularity Counts
Spotify Artists
The Weeknd
Coldplay
Justin Bieber
Drake
Kanye West
Eminem
Summer Walker
Adele
Taylor Swift
Ella Mai
H.E.R.
THE DRIVER ERA
Mayer Hawthorne
Tkay Maidza
Dua Lipa
0
20
40
60
80
Popularity Counts

## Soundcloud:


Box and Whisker Plot of Soundcloud Artists vs Play Counts
Soundcloud Artists
Eminem
Ella Mai
The Weeknd
Summer Walker
Drake
Adele
Kanye West
H.E.R.
Coldplay
Taylor Swift
Justin Bieber
Dua Lipa
THE DRIVER ERA
Tkay Maidza
Mayer Hawthorne
0
1
2
3
4
5
1e7
Play Counts

**Instructions For Running Code:**

Create a database if it does not exist

Open entire project file ‘SI-206-FINAL-PROJECT-Ethan-Hady-Kenyatta’ in VSCode

Run the spotify.py to add spotify api data and tables to the database

Run last_fm.py to add last fm api data and tables to database

Run soundcloud_soup.py to add web-scraped data from Soundcloud to the database

Finally, run the spotify_calculations.py file to make calculations (for all apis not just spotify) and generate the graph

**Function Documentation**:

## **spotify.py**

**receive_access_token** sends over to Spotify servers our client id and secret id to receive back a response that contains within it our access token, aka our API Key so that we may process the spotify json data.

**oauth_header** creates a header that will be required for future api calls. They are an argument required for any requests.get call, as well a params.

**track_search** takes in the token received from 'receive_access_token', any track name, and any track artist name, and then retrieves the first listed search result on spotify returning the name requested, the artist request and their (genre, track_popularity, explicit, album) for the requested song and artist. Also returns song name and artist name originally inputted.

**Create_artists_table** takes song_lst, a list of songs in which each song is represented as a tuple containing the song name and artist name, and the database,creates the table 'spot_artists' in the database and loops thru the the provided list adding unique ids with unique artists to the newly created table.

**Create_albums_table** takes in song_info_lst, a compiled list of tuples which contain all relevant data pulled from track_id_search, and the database, then creates the table 'albums' which, similarly to create_artists table, loops thru the data of the songs in a list adds the unique albums and their unique id numbers to the table.

**Create_genre_table** takes in song_info_lst, a compiled list of tuples which contain all relevant data pulled from track_id_search, and the database,then creates the table 'spotify_genres'. It then loops thru the provided list extracting the album information, and adding unique albums with a unique id assigned to spotify_genres table.

**Create_explicit_tables** takes in only the database as its argument and makes a subtable 'explicit' for explicitness in spotify songs. 1 represents yes, the song is explicit, and 2, represents, no the song is not explicit.

**Add_info_to_database** takes in an id_number, a tuple of song information, and the creates 'spotify_songs' table in the database which represents the main table for data collected on spotify songs, The data is constructed in away so that there is no duplicate string data. The columns in this table are the generated song id, song name, artist name (represented by id from spot_artists table, genre_id (represented by id from spot_genres table) add a row of tuple song data to the database, using id's to not have duplicate string data.

## Last_fm.py

**Get_lastfm_API_Key** takes in the filename of a file representing the .txt file which contains our granted API Key for Last FM. It returns the string contents of the file

**Get_track_info** takes in the API_key, the requested song_name, and requested artist_name,and returns a tuple containing the requested song, artist, the genre (the first genre tag listed in the json response), playcount (how many times the song/track has been played) and listener_count (how many users listen to the track on Last FM). The request similarly uses a series of required headers and params, in the form of a payload dictionary

**Create_genre_table_lastfm** takes in as an argument song_info_lst, a compiled list of tuples which contain all relevant data pulled from get_track_info and the database, and creates the lastfm_genre_tags table which houses the unique genre tags pulled by the API data search and extraction in get_track_info. Uses an accumulator to create unique ids for each unique genre.

**Add_info_to_database** takes in an id_number, a tuple of song information, and the creates 'lastfm_songs' table in the database which represents the main table for data collected on last fm songs, The data is constructed in away so that there is no duplicate string data. The columns in this table are the generated song id,song name, artist name (represented by id from spot_artists table, genre_id (represented by id from spot_genres table) add a row of tuple song data to the database, using id's to not have duplicate string data

## Soundcloud_soup.py

**Scrap** takes in a filename of an html file which contains the soundcloud webpages for all of the songs and then uses beautiful soup to fetch the play counts of each song/track. It then returns the id of the song pulled from the data base and an integer representing the play count number. For example, if you inputed '38_Greedy_ArianaGrande.html' as the argument, the it would return (38, [290212])

**Add_info_to_database** takes in the info, a tuple (the one generated in Scrap) and the database and creates the soundcloud_songs table. To the table it inserts the song name, artist name, and play counts for the song (Selecting the info based of the ID number)

## Spotify_calculations

**calculate_artist_by_popularity_lastfm** takes in the database as an argument, and extracts the last fm artist names and respective play counts using database JOINS, it then parses thru the data adding only the artists that have more than 1 track appearing in the data (for data viz purposes) then returns a dictionary where the keys represent the artists and the values represent each play count integer from the songs we pulled from that artist. For example: {'The Weeknd': [27869063, 12253723]. . . }

**calculate_artist_by_popularity_spotify** takes in the database as an argument, and extracts the spotify artist names and respective play counts using database JOINS, it then parses thru the data adding only the artists that have more than 1 track appearing in the data (for data viz purposes), then returns a dictionary where the keys represent the artists and the values represent each popularity rating () from the songs we pulled from that artist. For example:  {'The Weeknd': [90, 83]. . .}

**calculate_artist_by_popularity_soundcloud** takes in the database as an argument, and extracts the soundcloud artist names and respective play counts using database JOINS, it then parses thru the data adding only the artists that have more than 1 track appearing in the data (for data viz purposes), then returns a dictionary where the keys represent the artists and the values represent each play count integer from the songs we pulled from that artist. For example: {'The Weeknd': [4010235, 28342707] . . .}

**Create_lastfm_viz_artist**, **create_spotify_viz_artist**, and **create_soundcloud_viz_artist** each take in a dictionary representing the returned output of the calculation functions above, and they create the matplotlib visualizations for the respective streaming services. We made Boxplots for each of them to make a comparison for how frequent each of the artist's selected songs are on different streaming service platforms; noted for play counts for last fm and soundcloud, and popularity metrics for Spotify.

## Resource Documentation:

Date Issue Description Location of Resource Result (did it

| Date | Issue Description | Location | Result (did it resolve issue) |
|---|---|---|---|
| Apr 1, 2024 | Getting authorization to make the API/OAuth requests from spotify, using secret client and client id’s | https://www.youtube.com/watch?v=WAmEZBEeNmg&t=1267s | Yes<br><br>We were able to begin making spotify API calls reliably. |

| | | | |
|---|---|---|---|
| Apr 1, 2024 | Getting code examples and formatting for Spotify | https://developer.spotify.com/documentation/web-api | Yes<br><br>We were able to create API functionality for searching for songs within our python file |
| Apr 18, 2024 | Understanding Last Fm’s API request structure and format. How to make the calls to them, what to put in the request.get argument | https://www.dataquest.io/blog/last-fm-api-python/ | Yes<br><br>We were able to create the necessary header and payload params |
| Apr 23, 2024 | Turning the list of songs that we had in google drive into a list of tuples in Python | ChatGpt:<br><br>“hey chat! can you format this list of songs, where each row contains the first item, the song name, and the second itme is the artist name into a python list of tuples. Make the first item of the tuple the song name and the second item in the tuple the artists name: for example Cashmere by Tkay Maidza. ---> [('Cashmere', Tkay Maidza) . . . ]” | Yes<br><br>It beautifully made this list into a python list of tuples so that we would not have to do the painstaking work. |
| Apr 23, 2024 | Debugging code errors, like tuple type errors | File "/Users/ethancasler/Desktop/206_python/Final_Project/SI-206-FINAL-PROJECT-Ethan-Hady-Ke | Helped to illuminate the mistake i made with the tuple arrangements, and in indexing it, to put in the tables. |

| | | | |
|---|---|---|---|
| | | nyatta/spotify.py", line 133, in create_artists_table<br><br>cur.execute('INSERT OR IGNORE into spot_artists (id, artist_name) VALUES(?,?)', (count, song_lst[i])) sqlite3.ProgrammingError: Error binding parameter 2: type 'tuple' is not supported<br><br>what's wrong with my python code here, why isn't it taking the values?” | Here’s the code snippet it gave us:<br><br>for song in songs:<br><br>cur.execute('INSERT OR IGNORE INTO spot_artists (id, artist_name) VALUES (?, ?)', (count, song[1])) |