Spotify Play Count and Popularity Tracker

Jasper Tsai

1 Introduction

Spotify is one of the biggest audio streaming companies, with over 489 million monthly active users. Many up-and-coming artists hope to release their music onto the platform and have their music added to Spotify's curated playlists. Spotify has an internal value called the popularity index that ranks all artists and their tracks on a 0 to 100 scale. This popularity index is what Spotify mainly uses in its algorithm to create suggestions for listeners all over the world. A high popularity score is crucial to improve a track's discoverability within the platform and have them featured in any playlists created by Spotify's algorithm. Currently, this popularity index is only available through Spotify's official web API or a third-party service called Musicstax. Stream counts, save rate, number of playlists a track appears in, skip rate, and share rate calculates the popularity index. However, it is noted in the official documentation that recent stream counts have the most impact on popularity scores. Also, it is crucial to note that there are different levels of popularity scores. Each track has a popularity index. All tracks' popularity index in an album creates the popularity score of an album, and the popularity of all the artist's songs calculates the artist's popularity score. In this project, we will specifically look at the association between total play counts and the popularity score over a period of approximately 14 days for 12 different artists and highlight patterns found through exploratory data analysis. We will also compare all 12 artists to see if their popularity score is associated with the number of followers they each have. The final deliverable for this project will be an interactive dashboard showing a proof of concept of the data retrieval methods we have learned and potentially helping future researchers reverse engineer Spotify's popularity score index.

All scripts, notebook, and dashboard of this project may be found in this GitHub Repository: https://github.com/jttsai99/Spotify-PlayCounts

2 Methods and Approach Outline

The data for this project was primarily sourced from Spotify's official web API. Unfortunately, data such as play counts for each track is not provided by the official API, so we had to work around this challenge by finding an alternative source to retrieve the track play counts. Originally, we attempted to scrape the Spotify website directly but were only successful at scraping the play counts for the top 10 tracks of each artist. Eventually, we located an undocumented API that we utilized to retrieve all tracks' play counts. We decided to create an SQLite database to store all this information from both APIs and query it before making our plots and dashboard.

The overall workflow of this project is described as follows:

1. Retrieved information for the 12 selected artists and all their albums and track's info from Spotify's official web API [1].
2. Utilized identifiers from the official web API to navigate to the appropriate link and retrieve play counts with the unofficial API.

3. Constructed SQL database with multiple tables and have at least one relational column in each.
4. Wrote a separate script that updates the SQL database daily and repeat the process for approximately 14 days.
5. Produced interactive plots with Plotly [2] package and use Streamlit [3] package to create an interactive dashboard.

3 Data Acquisition and Processing

3.1 Spotify's official web API

To retrieve relevant information from Spotify's official API, we used multiple endpoints to successfully gather all 12 artists' information. However, before any request could be made to the API, we needed to create a developer application and write a function to acquire an authentication token to pass into each API request call. The process to get to each track's popularity score was intricate and required multiple prior requests to achieve the desired outcome.

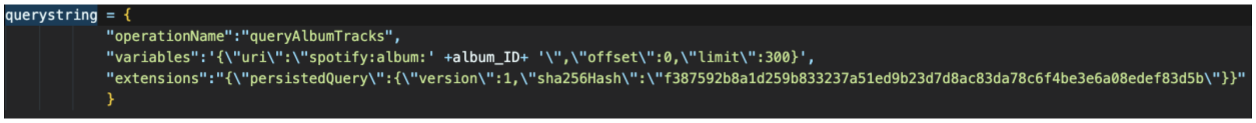
The general sequential request order to reach tracks' popularity score is described as follows:

1. Retrieved *artist_id* by using the search for item endpoint
2. With *artist_id*, we ran the function `get_all_singles_albums_by_artist()` and acquired all the *album_id* and other album related information such as release date and album name.
3. The *album_id* is then passed into the function `get_tracks_from_albums()` to retrieve all *track_id*.
4. Lastly, the last function `get_pop_by_tracks()` takes in each track_id and request the popularity score for each of the tracks.

A couple of challenges arose when we initially implemented the above methods for individual albums and tracks one at a time. The large number of requests sent to Spotify was inefficient. The developer application eventually stopped retrieving Json data to be processed because we exceeded the API rate limit. To combat this issue, we had to rewrite our retrieval functions to retrieve a set of 20 albums at a time and 50 tracks at a time. Thankfully the Spotify official API has a parameter to offset the number of songs we start retrieving as a group. In the end, we shorten our 881 track requests to 76 requests.

3.2 Undocumented Spotify API

Using Google Chrome's developer Network tools, we identified an undocumented API request whenever we restarted the Spotify webpage. After carefully mimicking the request call through an app called Insomnia [4], an API development tool, we successfully located the play counts of each track in a particular Album. All we needed to provide was an endpoint with a modified link to request the play counts of all tracks in an album. The *album_id* we previously acquired in the documented API came in handy and helped us construct the endpoint link.



```
querystring = {
            "operationName":"queryAlbumTracks",
            "variables":'{\"uri\":\"spotify:album:' +album_ID+ '\",\"offset\":0,\"limit\":300}',
            "extensions":"{\"persistedQuery\":{\"version\":1,\"sha256Hash\":\"f387592b8a1d259b833237a51ed9b23d7d8ac83da78c6f4be3e6a08edef83d5b\"}}"
            }
```

Request Parameter for Undocumented API

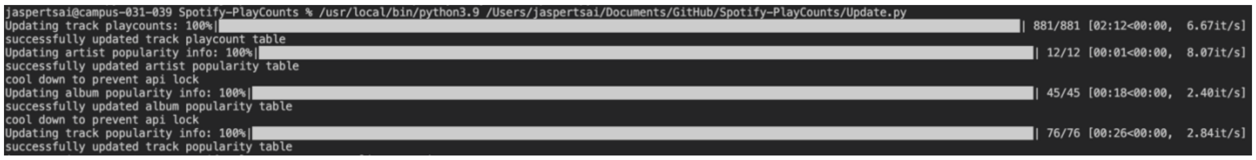
Currently, the implementation for the undocumented API has one minor flaw that requires additional changes every time we choose to run the request. The authentication and client token for the request header has an expiration of 1 hour, which means that after every hour, we must refresh the Spotify webpage manually to retrieve new authentication and client token. The temporary workaround for this problem is to refresh the Spotify page, locate the specific outward request from the Network developer tools, copy the link as cURL, and paste it into Insomnia to get the new authentication and client token. In the future, we hope to automate this process using Selenium[5] to eliminate such a manual task and streamline the requests from the undocumented API.

3.3 Data Processing and SQL Database

We decided to maintain an SQLite database to store all the acquired data and utilize the relational properties between different tables to combine data when we later query the data to draw plots for our exploratory data analysis and interactive dashboard. The following describes all the SQL tables and sample columns of what each table entails:

1. **Artists**: *artist_id*, *artist_name*
    a. Static table that will be used to reference artist id to their respective artist name.
2. **Artists_Popularity**: *artist_id*, *artist_followers*, *artist_popularity*, *date_tracked*
    a. Updated daily
3. **Albums**: *album_group*, *album_name*, *album_id*, *artist_id*, etc.
    a. Static table with all albums and their identifier during the period of tracking.
4. **Albums_Popularity**: *album_id*, *album_popularity*, *date_tracked*
    a. Updated daily
5. **Tracks**: *track_name*, *track_id*, *album_id*, etc.
    a. Static table with all tracks and their identifier during the period of tracking
6. **Tracks_Playcount**: *track_id*, *track_playcount*, *date_tracked*
    a. Updated daily
7. **Tracks_Popularity**: *track_id*, *track_popularity*, *date_tracked*
    a. Updated daily

For all the tables labeled “Updated daily”, we created a python script `Update.py` to update all tables simultaneously.



```
jaspertsai@campus-031-039 Spotify-PlayCounts % /usr/local/bin/python3.9 /Users/jaspertsai/Documents/GitHub/Spotify-PlayCounts/Update.py
Updating track playcounts: 100%|████████████████████████████████| 881/881 [02:12<00:00,  6.67it/s]
successfully updated track playcount table
Updating artist popularity info: 100%|██████████████████████████| 12/12 [00:01<00:00,  8.07it/s]
successfully updated artist popularity table
cool down to prevent api lock
Updating album popularity info: 100%|███████████████████████████| 45/45 [00:18<00:00,  2.40it/s]
successfully updated album popularity table
cool down to prevent api lock
Updating track popularity info: 100%|███████████████████████████| 76/76 [00:26<00:00,  2.84it/s]
successfully updated track popularity table
```

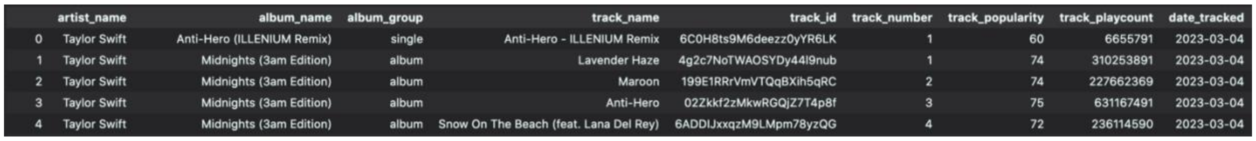
An example of running `Update.py`

4 Exploratory Data Analysis

4.1 Visualization Techniques

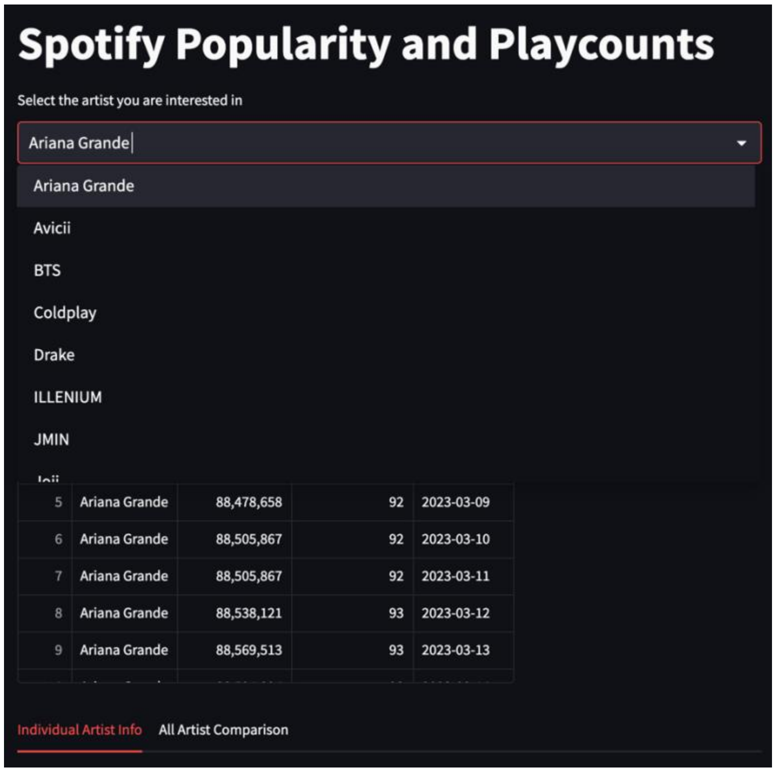
The previously constructed SQL database allowed us to query a data frame ideal for plotting and grabbing elements from different tables using relational columns (See Table 1).

Table 1. Data frame from SQL query to combine different tables



	artist_name	album_name	album_group	track_name	track_id	track_number	track_popularity	track_playcount	date_tracked
0	Taylor Swift	Anti-Hero (ILLENIUM Remix)	single	Anti-Hero - ILLENIUM Remix	6C0H8ts9M6deezz0yYR6LK	1	60	6655791	2023-03-04
1	Taylor Swift	Midnights (3am Edition)	album	Lavender Haze	4g2c7NoTWAOSYDy44l9nub	1	74	310253891	2023-03-04
2	Taylor Swift	Midnights (3am Edition)	album	Maroon	199E1RRrVmVTQqBXih5qRC	2	74	227662369	2023-03-04
3	Taylor Swift	Midnights (3am Edition)	album	Anti-Hero	02Zkkf2zMkwRGQjZ7T4p8f	3	75	631167491	2023-03-04
4	Taylor Swift	Midnights (3am Edition)	album	Snow On The Beach (feat. Lana Del Rey)	6ADDIJxxqzM9LMpm78yzQG	4	72	236114590	2023-03-04

For this project we created all the plots using Ploty[2] and integrated with Streamlit [3] to make a dashboard that allows the user to explore the data themselves.



Spotify Popularity and Playcounts

Select the artist you are interested in

Ariana Grande

5	Ariana Grande	88,478,658	92	2023-03-09
6	Ariana Grande	88,505,867	92	2023-03-10
7	Ariana Grande	88,505,867	92	2023-03-11
8	Ariana Grande	88,538,121	93	2023-03-12
9	Ariana Grande	88,569,513	93	2023-03-13

Snapshot of the dashboard. (dark theme)

For a full experience follow the instructions in the readme of the GitHub Repository: [*https://github.com/jttsai99/Spotify-PlayCounts*](https://github.com/jttsai99/Spotify-PlayCounts)

4.2 Comparing Followers and Popularity score

In this portion of the analysis, we attempt to identify if there is any direct association between the number of followers an artist has with their artist's popularity score. From Figure 1 to Figure 3 below, we notice that despite having different numbers of followers, there is no clear pattern in which the popularity score increase or decreases. One notable observation from the plots is that for artists with fewer followers (less than 1 million), there is more fluctuation in the follower count histogram. It is interesting to see an artist's popularity score remain unaffected with a noticeable follower count decline (See Jmin in Figure 2). The two artists with a change in popularity score were Ariana Grande and Illenium. Both had an increase in follower count, but Illenium's popularity score fluctuated back and forth. Ariana Grande's popularity score increased by one overall (See Figure 1 and Figure 2).

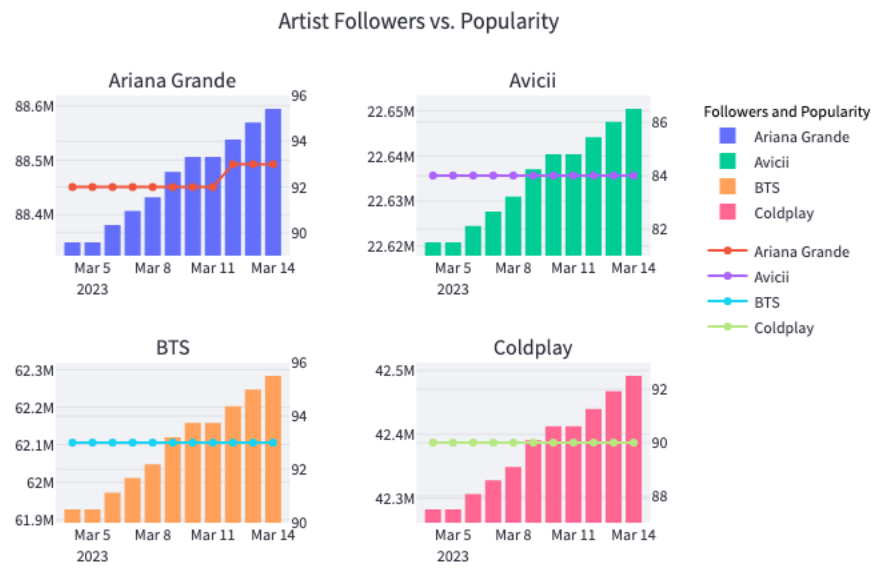


Figure 1. Followers vs. Popularity of Ariana Grande, Avicii, BTS, and Coldplay

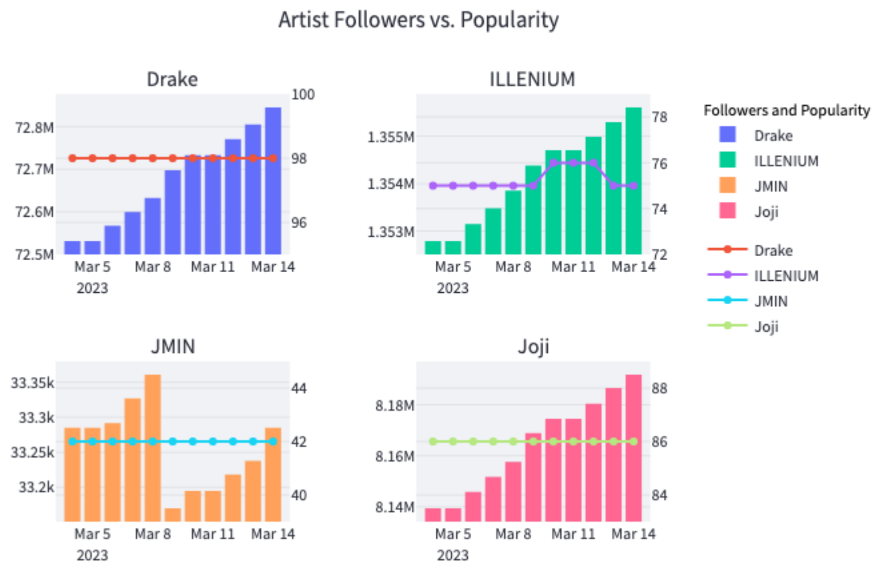


Figure 2. Followers vs. Popularity of Drake, Illenium, Jmin, and Joji

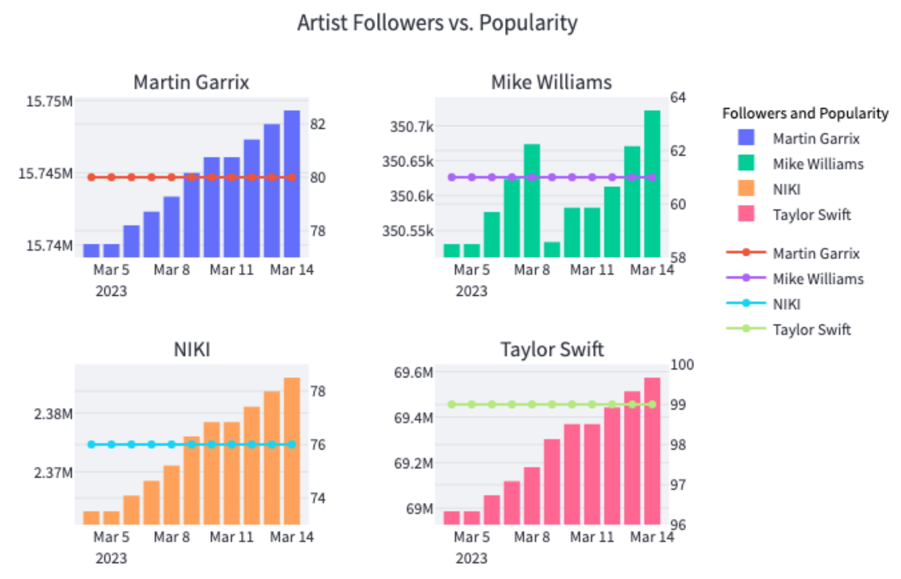


Figure 3. Followers vs. Popularity of Martin Garrix, Mike Williams, NIKI, and Taylor Swift

4.3 Comparing Play Count and Popularity Score

This section aims to answer the primary question we posed in the introduction, "Were there any observable patterns for play counts and popularity of each artist's tracks?" We found that the majority of songs that were released previously on Spotify did not have significant changes to their popularity score. For example, figure 4 shows BTS's Dynamite, which was top 1 on the Billboard Hot 100 chart for three weeks when it was initially released, is now barely increasing in popularity.

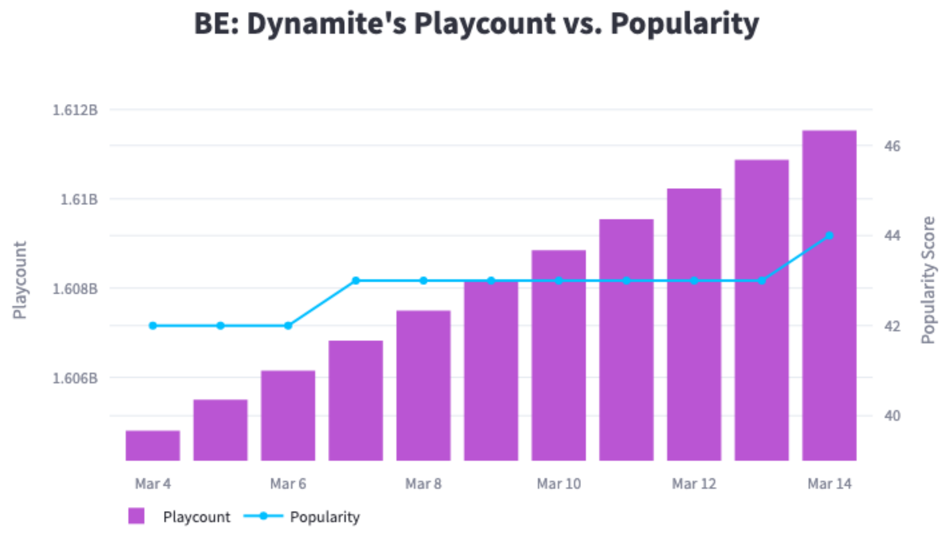


Figure 4. BTS's "Dynamite" released on Nov 20, 2020.

However, more recently released tracks saw their popularity index change drastically as play counts increased (See Figure 5). Songs released near the start of the tracking period of March saw the most variability in their popularity score.

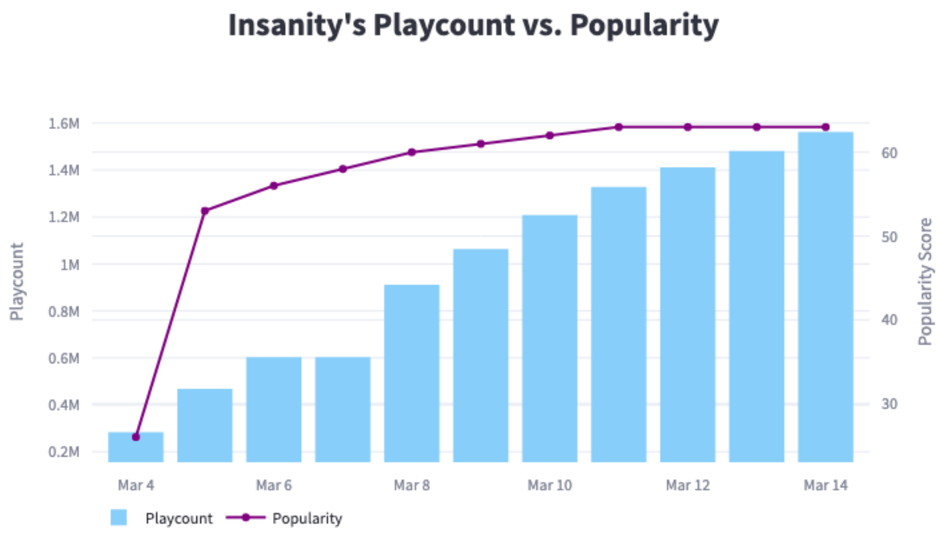


Figure 5. ILLENIUM's "Insanity" released on Mar 3, 2023

Another interesting observation that seems to be a reoccurring pattern is that tracks relating to holidays such as Christmas start to reduce in popularity during the tracking period of March.

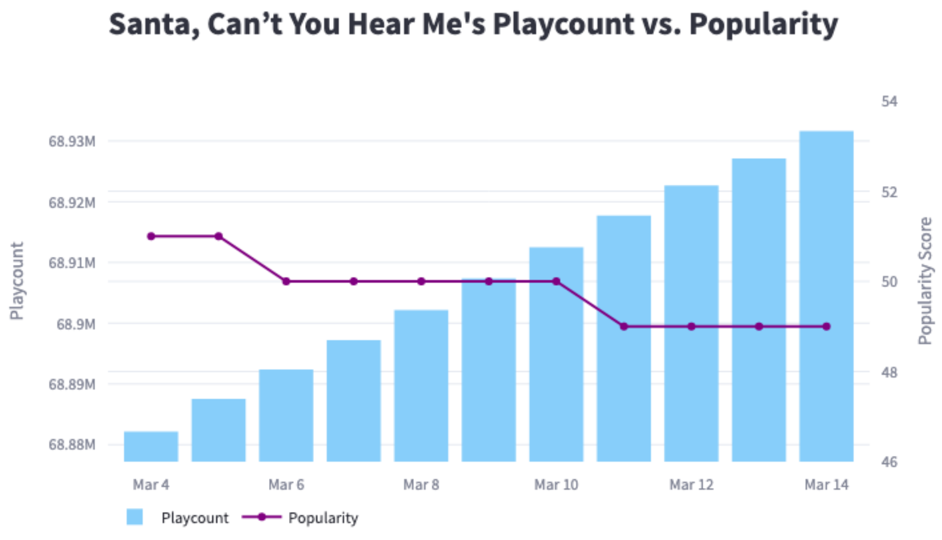


Figure 6. Ariana Grande's "Santa, Can't You Hear Me" released on Dec 14, 2022

The graph in figure 6 demonstrates a continual decline in popularity score for a seasonal track. We expect this trend to continue throughout the year. It will be a question of interest to observe if these tracks will increase in popularity again once the holiday season approaches.

5 Conclusions and Discussions

In this project, we gathered data from Spotify's official API and combined it with data collected from Spotify's undocumented webpage API. We also maintained a SQL database by updating it daily for approximately two weeks. In the end, we created an interactive dashboard to allow future users to explore the data themselves. Using the data visualization tool produced, we conducted a simple exploratory data analysis to see if there are any patterns in an artist's number of followers and popularity index of the artist as well as the track's play count and corresponding popularity score.

Our key findings include:

1. It is difficult to see any patterns for whether an artist's followers affect their popularity score, and we cannot make any inference based on the limited sample size we have in our dataset. We might find a more noticeable pattern in the data if we provided a lengthier tracking period.
2. We noticed that older released tracks' popularity scores change less dramatically compared to recently released songs. We also believe that Spotify's algorithm initially boosts new songs' popularity scores to help listeners discover them in various playlists.
3. Lastly, there is a clear declining trend in popularity index for seasonal songs. We suspect this trend will continue throughout the year until holiday season returns.

This project is a proof of concept to demonstrate the data retrieval methods we learned in STA 220. However, this program may prove more useful to those interested in reverse engineering Spotify's internal algorithm if we maintain the database for a lengthier period. Plans to improve this project include automating the procedure to retrieve play count data using Selenium and allowing users to input the desired artists they wish to track. Once more data is collected, we could implement prediction methods to forecast how well an artist's new track will perform under a similar genre of music they produce.

6 References

1. https://developer.spotify.com/documentation/web-api/reference/#/
2. https://plotly.com/python/
3. https://streamlit.io/
4. https://insomnia.rest/
5. https://www.selenium.dev/

GitHub Repository:

https://github.com/jttsai99/Spotify-PlayCounts