Ethan Hod & Ethan Udler
SI 206 Final Project Report & Instructions

Github Link: https://github.com/ethanhod05/si-206-final.git

Project Goals

The primary goal of this project was to explore the relationship between artist popularity and live concert patterns by integrating data from two major platforms: Spotify and Ticketmaster. Specifically, we aimed to gather artist-level metadata (ex: popularity score, followers) and pair it with real-world concert event information (ex: locations, frequencies) in order to uncover trends in how music popularity translates to live performance opportunities. By storing the data in a normalized SQLite database and applying analytical techniques, we sought to create multiple visualizations that reveal meaningful patterns in the music industry.

Achieved Goals

We successfully connected to the Spotify Web API to gather detailed artist information, including popularity score, follower count, and genre. Simultaneously, we used the Ticketmaster Discovery API to pull upcoming concert data for artists performing in the U.S. and internationally. Both datasets were cleaned and merged into an SQLite database using structured tables: SpotifyArtists, PopularityLevel, and UpcomingConcerts. We ensured at least 100 records were gathered from each source and that only 25 lines would print at a time and not repeat an artist's name twice. A key achievement was generating four thoughtful visualizations that each revealed a different aspect of the artist-performance relationship.

Problems Faced

Several challenges came up during the development of our code. One of the earliest roadblocks was authenticating with Spotify's Web API using the proper token key. For the first few days of the project, we struggled to get Spotify's Authorization working correctly. Whether it was trying to run a server on my computer or accessing it directly from the web, it wouldn't provide me with an API key. This delay was frustrating and impacted our ability to begin data collection on time. Eventually after sitting in office hours, we figured out how to correctly structure the token request and include it in our headers, which allowed us to start retrieving artist data.

Another major challenge was designing an integer key that would meaningfully connect both datasets. Initially, our Spotify and Ticketmaster data had no shared unique strings that could easily be joined with a relational key. This roadblock deterred us from our initial goals of finding the number of ticket sales, as this information wasn't available on the API. After brainstorming and receiving guidance from our GSI, we decided to categorize artists into popularity tiers—Low, Mid, and High—and created a new table to represent those tiers. We then used an integer key to associate each artist with a popularity level, which allowed us to join artist metadata with concert data meaningfully. This was a creative solution that made our analysis stronger and more cohesive.

Additionally, we encountered issues with duplicate data during API pulls, which we resolved by limiting fetches to 25 records at a time and implementing primary key constraints. Handling null entries, especially when artists had no upcoming concerts, also required extra logic to ensure our scripts didn't crash and our visualizations remained clean and accurate.

Calculations

We performed several key calculations that supported our analysis and visualizations. One important transformation was the use of a logarithmic scale (log10) for Spotify follower counts in our *popularity vs. followers* scatter plot. Because follower counts ranged from tens of thousands to tens of millions, plotting them on a linear scale would have distorted the data. Applying the log10 transformation allowed us to display the data more proportionately and made it easier to compare lesser-known and top-tier artists on the same chart.

Another significant calculation was our creation of an integer key system by grouping artists into popularity levels—Low, Mid, and High—based on their Spotify popularity score (0–100). This categorization enabled us to connect data across both APIs in a meaningful way, even though Spotify and Ticketmaster didn't share natural overlapping string fields. By assigning a unique ID to each popularity level and using that as a foreign key, we were able to join artist metadata with concert frequency and location data for cleaner analysis.

Finally, we calculated the average number of concerts by popularity level using a SQL GROUP BY query. This aggregation allowed us to visualize which popularity tier typically performs the most live events. Surprisingly, the results showed that mid-level artists often have more scheduled concerts than the highest-tier artists, possibly due to greater touring availability or promotional needs.

Documentation: & AI Help

One of the most unique and helpful aspects of this project was how we incorporated AI tools like ChatGPT to support our development process. From the very beginning, AI played a major role in helping us debug API authentication issues, especially when we struggled to connect to Spotify's token system. It also guided us through brainstorming solutions when our data sources didn't align naturally, like suggesting the use of popularity tiers to create an integer key.

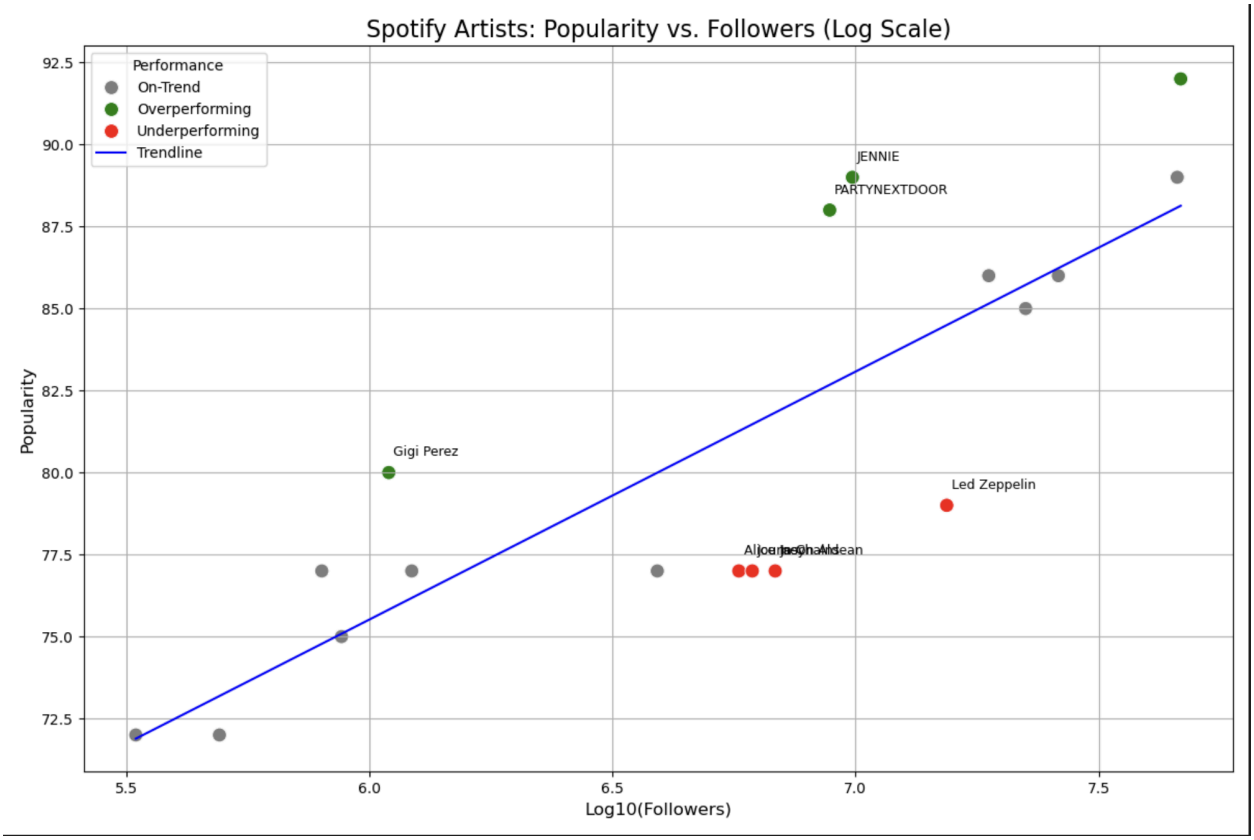
Beyond troubleshooting, AI helped us write cleaner, more readable code by generating reusable functions, suggesting SQL queries, and even explaining complex error messages in ways that made sense. Overall, using AI didn't take away from the learning process — it enhanced it. It gave us a way to work more independently, ask questions without hesitation, and get unstuck quickly.

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
04/14	Needed official endpoint details and parameters for artist search	Spotify Web API Docs	Used to build Spotify data-fetching logic
04/14	Needed to retrieve concert data for artists	Ticketmaster Discovery API Docs	Used to fetch event counts and locations
4/14	Trouble authenticating with Spotify API (token not returned)	Spotify Auth Flow Guide	Helped fix token header
4/15	Needed to prevent duplicate artist data in database	Stack Overflow – SQLite INSERT IF NOT EXISTS	Used conditional insert logic
4/16	Difficulty integrating integer key across unrelated APIs	Guidance from GSI during lab session	Used popularity level as creative join key
4/17	JSONDecodeError when Spotify API token request failed	Requests JSONDecodeError Docs	Added error handling for failed token calls
4/18	Heatmap not displaying events accurately by state	Matplotlib Hexbin Docs	Switched to hexbin by lat/lon
4/20	Needed log scaling for scatter plot to better visualize artist follower ranges	Matplotlib Log Scale Docs	Successfully applied log10 to y-axis

Visualizations Created

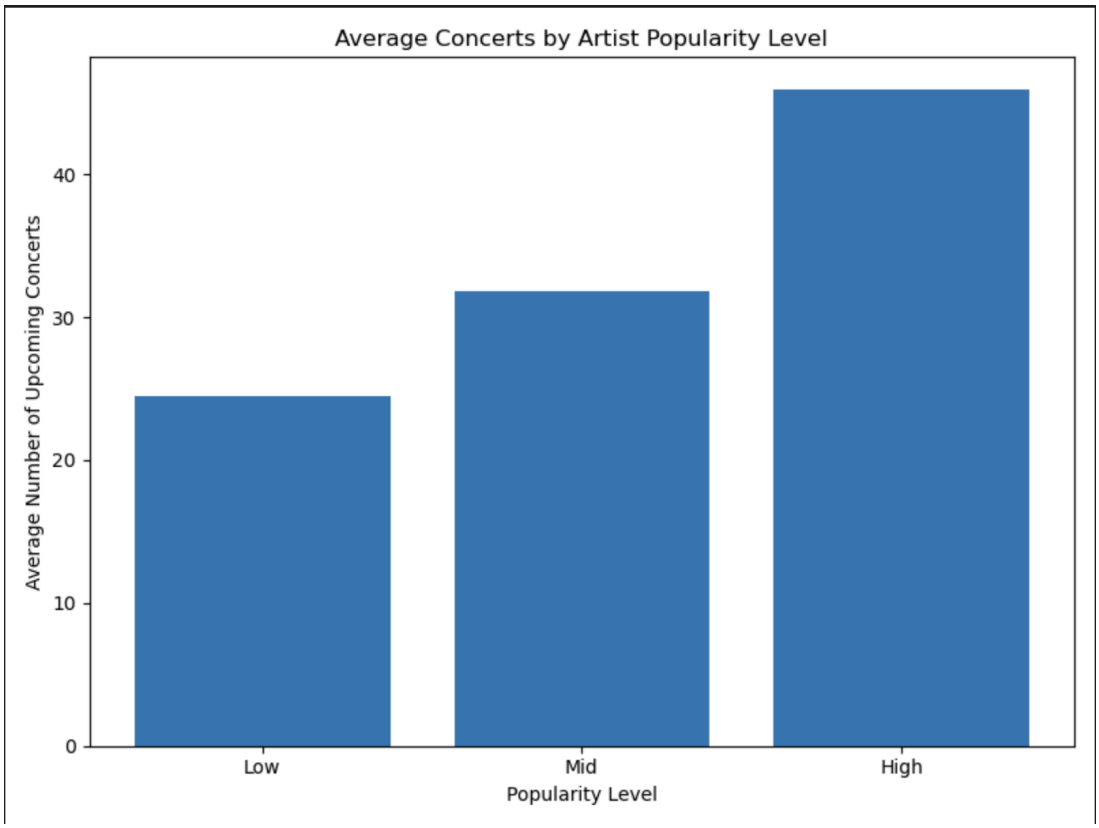
We created four distinct visualizations. These visualizations are automatically generated and organized into a dedicated folder when create_visualizations.py is run, making them easy to access and manage for review or presentation purposes:

Popularity vs. Follower Count Scatter Plot



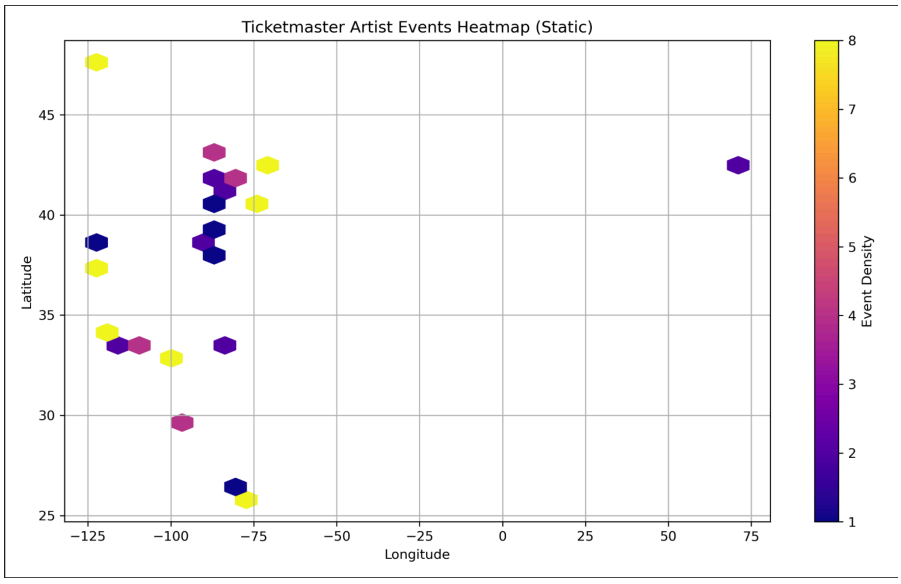
This chart plotted artist popularity (0–100) on the x-axis against their Spotify follower count on the y-axis. Each point represented an artist, and we used clean dot markers to enhance readability. This visualization made it easy to spot outliers—such as artists with niche popularity but high loyal followings—and gave insight into how fame does or does not correlate with fanbase size.

Bar Chart of Concert Counts by Popularity Level



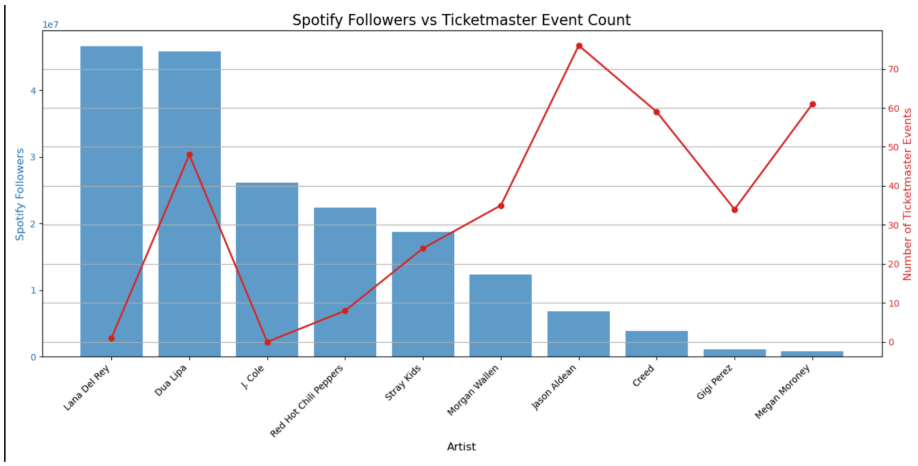
Artists were divided into "Low," "Mid," and "High" popularity categories. We computed the average number of concerts scheduled per group using SQL queries and then visualized the result using a bar chart. This helped illustrate that mid-tier artists, surprisingly, often had more tour events than top-tier ones, possibly due to scheduling flexibility or smaller venues.

Geographic Hexbin Heatmap of Artist Concert Events



We used latitude and longitude coordinates from the Ticketmaster API to generate a hexbin heatmap that shows where artist events are most densely clustered. This static heatmap provides a spatial view of concert activity across the U.S. and abroad, with color intensity representing event density. Yellow indicates the highest concentration of events in a given region, while darker blues represent areas with fewer events. This visualization reveals hotspots such as the U.S. West Coast and Northeast, where artist activity is most prevalent.

Dual-Axis Line Graph: Popularity vs. Concert Count



This line graph showed two trends side-by-side: average popularity and number of concerts for artists in both our datasets. Using two vertical axes, one for each metric, we were able to compare how these two variables rise and fall over the same artist groupings. The result was a nuanced view of the balance between fame and touring.

Instructions to Run the Code

1) Start with Artists_in_the_world.py

   This file uses BeautifulSoup to scrape data from Billboard's Top 100 Artists chart.

   Extract Artist Data

   The scraped artist names are returned as a list from this file.

2) Use the List in Spotify_fetch.py

   The list of artists from Artists_in_the_world.py is imported into Spotify_fetch.py to retrieve Spotify data for each artist.

Run Data Collection:

- Spotify_fetch.py: Fetches artist data from the Spotify API (in batches of 25).
    - This code prints 25 rows of unique artists into This_one_works.db
        - Run Spotify_fetch.py 4 times to get a list of 100 total different ID, Names, Followers, Popularity, and Popularity Level

3) Use the list in ticketmaster_fetch.py, which collects upcoming concert data from the Ticketmaster API.

   Each run will store new, non-duplicate data into this_one_works.db.

   - Ticketmaster_fetch.py fetches data on the number of upcoming concerts for 25 unique artists around the world at one time
       - Run ticketmaster_fetch.py 4 times to get a list of 100 unique artists and the number of upcoming concerts each one of them has

4) Run Visualizations:
   Execute create_visualizations.py to automatically generate and save all four visualizations:

   - Popularity vs. Followers (Log Scale) Scatter Plot
   - Average Concerts by Popularity Level Bar Chart
       - (This is the visualization that uses the JOIN function in SQL)
   - Dual-Axis Line Graph (Popularity vs. Concert Count)
   - Geographic Hexbin Heatmap of Concert Locations

     All images will be organized into a dedicated **output folder (Visualizations)** for easy access.

5) View Calculations:write_calculations_to_text.py
   The file write_calculations_to_text.py outputs key statistics such as average concerts by popularity level and log-scaled follower data into results.txt.

6) Database:
   The SQLite database file this_one_works.db contains three tables: SpotifyArtists, PopularityLevel (integer key), and UpcomingConcerts, with 100+ rows of live data from each API.

Documentation for Each Function We Wrote

	Purpose	Input	Output
get_top_100()	Scrapes Billboard's Top 100 artist list using BeautifulSoup	None	List of top 100 artist names
Spotify_fetch.py	Fetches Spotify data (followers, popularity) for each artist in list	List of artist names	Spotify metadata (JSON, DB entries)
sqlite3.connect("...")	Connects to the SQLite database	Database file name	SQLite connection object
CREATE TABLE IF NOT EXISTS	Creates the UpcomingConcerts table if it doesn't already exist	Table schema	Table schema
SELECT COUNT(Name_i)	Gets the number of artists already inserted into the database	SQL query	Integer (row count)
requests.get(...)	Fetches concert data for each artist from the Ticketmaster API	Artist name, API key, query params	JSON response with event info
INSERT INTO UpcomingConcerts	Stores artist name and number of concerts into the database	Artist name, event count	Data inserted into SQLite database
spotify_artist_visualization.py			

generate_visualization()	Main function that orchestrates token fetching, data retrieval, DB insert, and plotting	None	JSON file, SQLite entries, and visualization image
get_token()	Fetches an access token from Spotify's API using client credentials	None (uses CLIENT_ID and CLIENT_SECRET)	Spotify access token (string)
get_artist_data()	Queries Spotify API for an artist's followers and popularity	Artist name (string), Spotify token	Dictionary with name, followers, and popularity
sqlite3.connect()	Connects to SQLite database and prepares for data insertion	"SpotifyArtists.db"	SQLite connection object
CREATE TABLE IF NOT EXISTS	Creates a SpotifyArtists table for storing artist data	Table schema	SQLite table if not already created
INSERT INTO SpotifyArtists	Inserts artist metadata (name, followers, popularity) into the database	Artist data dictionary	Inserts rows into SQLite DB
pd.DataFrame(...)	Loads artist data into a Pandas DataFrame for analysis	List of artist dictionaries	Pandas DataFrame
LinearRegression(). fit()	Fits a linear model predicting popularity based on	X = log_followers, y = popularity	Regression model and predictions stored in new DataFrame columns

	log-transformed followers		
sns.scatterplot()	Plots a scatterplot of popularity vs. log followers, colored by performance	DataFrame with computed fields	Scatterplot saved as PNG
plt.savefig(...)	Saves the generated plot as an image	File path	PNG file in Visualizations/ directory

Bonus / Extra Credit

+10 presentation - Our group presented early compared to other groups allowing us to get this EC
+30 Additional API sources (Beautiful soup scraping of billboard top 100)
+30 2 extra visualizations

What We learned from the API

Through this project, we've learned a lot about working with APIs, web scraping, and managing databases. We gained hands-on experience with authenticating APIs like Spotify, retrieving data, and handling potential errors like rate limits. Scraping data from websites using BeautifulSoup taught us how to extract useful information from HTML, but we also learned how fragile scraping can be when website structures change.

Working with SQLite databases helped us understand how to structure and store data efficiently, ensuring it's clean and free of duplicates. We also practiced writing SQL queries to analyze and join different data sources. The data visualization part was particularly fun, we learned how to turn raw data into clear and meaningful charts using Matplotlib and Seaborn.

Overall, this project taught us how to integrate data from various sources, solve technical issues as they arose, and present everything in an easy-to-understand format. It was a great learning experience that gave us a deeper understanding of how to work with data in real-world scenarios.