Github Repo Link: https://github.com/tylernow/206-final.git

Changes made post-grading:

1. After discovering that adding 25 songs from the BillBoard Top 100 for each run of the code counted against the 25-add API rule. Because we also would add the top 5 tracks of each artist that is found in these songs which would add more than 25. We added another limiter that tracks the amount of top tracks added and if it reaches 25 will stop the code.

A. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)
    a. The goal of this project was to analyze the relationship between the Billboard Hot 100 rankings and Spotify popularity metrics. By gathering data from both sources, we aimed to identify correlations between song Billboard rankings and their Spotify popularity. We used two primary sources: the Billboard Hot 100 Website (https://www.billboard.com/charts/hot-100/) and the Spotify Web API (https://api.spotify.com). We planned to retrieve the name, artist(s), and ranking of the top 100 songs from the Billboard website to make a song database. Then we would use Spotify to collect the album name, release date and popularity score of each song to make another database to join with the Billboard song database. Afterwards we would make one more database from the Spotify top 5 tracks of each unique top 100 artist, which we planned to use for further analysis.

B. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)
    a. We successfully achieved our original project goals by gathering and integrating data from both the BillBoard Hot 100 website and the Spotify Web API. From the Billboard Hot 100 website we scraped the top 100 songs, including their names, artists, and rankings. Then using the Spotify API, we matched each song to its corresponding track in Spotify and retrieved additional metadata. Such as album name, album release date, and popularity score. We also gathered the top 5 tracks for each unique artist from the Spotify API and stored this information in a separate table to our primary artist table. All data was then stored and normalized in a SQLite database.

C. The problems that you faced (10 points)
    a. One problem faced was customizing the visuals to be different from those seen in class. This was solved by searching through the library documentation and troubleshooting extensively by trial and error. Another problem was making the scatterplot to see the correlation between the album release date of a popular song

and the song's ranking. This was due to the difficulty of having accurate and legible labeling, which was resolved by plotting multiple years over a timeline of "months" rather than graphing the full timespan continuously. We also faced minor difficulties in syncing the database with our Jupyter Notebook. Since SQLite can only be accessed by one process at a time, the file was occasionally locked due to open connections in Python scripts. This was resolved by ensuring that all connections were properly closed and no background processes were using the database while visualizing the data. Another problem was that our method of adding top songs by artists went over the API 25-add limit rule. While we already limited songs to 25 per run, we realized artist top tracks could still cause us to exceed the limit. We solved this by adding an additional limiter to cap the number of artist top tracks added to 25 entries per run as well. The program now tracks how many ArtistTopTracks entries are added and stops once the total hits 25.

D. The calculations from the data in the database (i.e. a screenshot) (10 points)

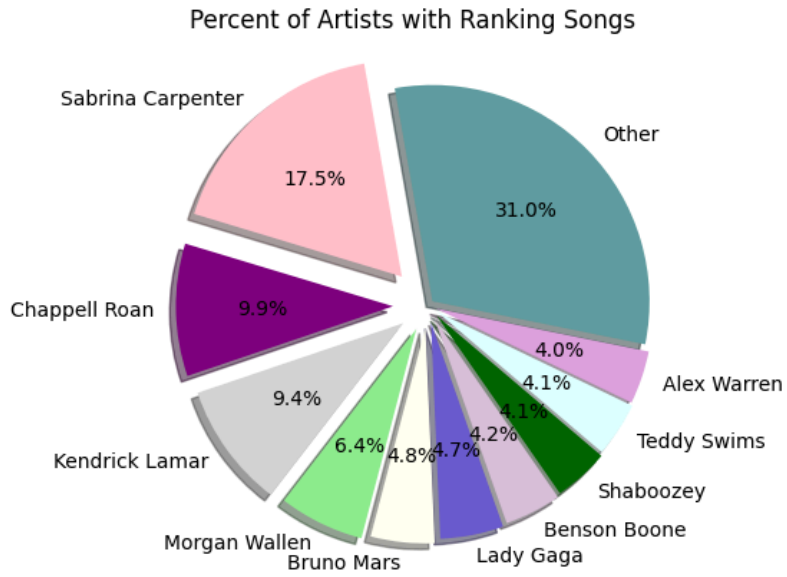
🔪 Top 10 Artists by # of Top 100 Songs:

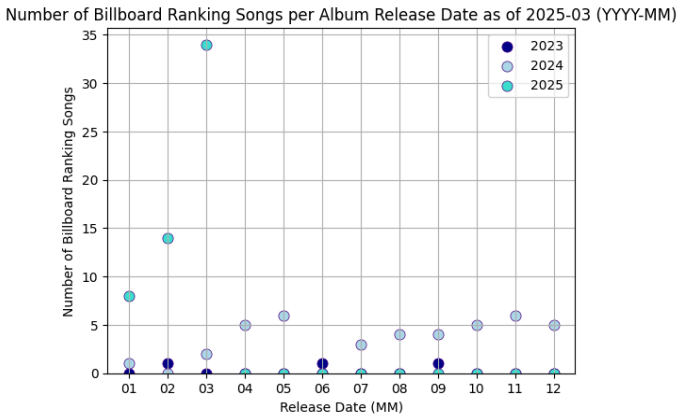
artist	song_count
Sabrina Carpenter	16
Chappell Roan	9
Kendrick Lamar	8
Morgan Wallen	6
Alex Warren	4
Benson Boone	4
Bruno Mars	4
Lady Gaga	4
Shaboozey	4
Teddy Swims	4

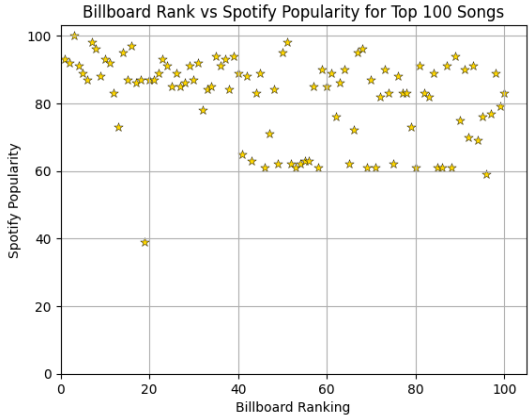
🔥 Top 10 Artists by Total Popularity:

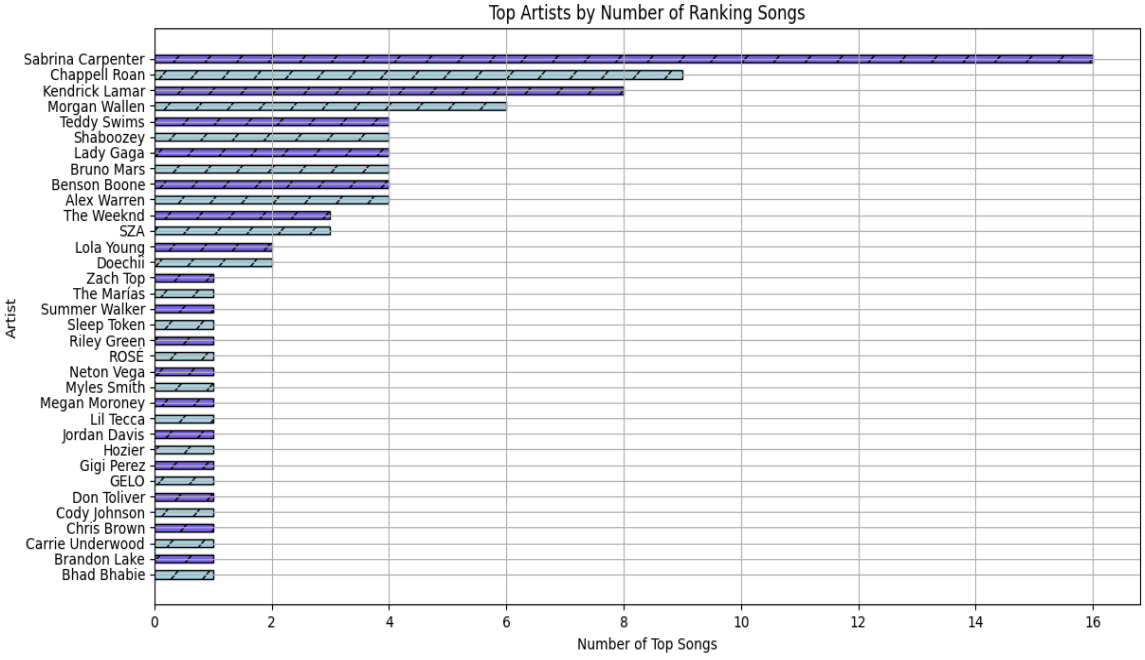
artist	total_popularity
Sabrina Carpenter	1440
Chappell Roan	813
Kendrick Lamar	768
Morgan Wallen	522
Bruno Mars	390
Lady Gaga	386
Benson Boone	348
Shaboozey	338
Teddy Swims	336
Alex Warren	326

E. The visualization that you created (i.e. screenshot or image file) (10 points)


Percent of Artists with Ranking Songs
Sabrina Carpenter
17.5%
Other
31.0%
Chappell Roan
9.9%
4.0%
4.1%
Alex Warren
4.1%
4.2%
Teddy Swims
Kendrick Lamar
9.4%
6.4%
4.8%
4.7%
Shaboozey
Benson Boone
Morgan Wallen
Bruno Mars
Lady Gaga


Number of Billboard Ranking Songs per Album Release Date as of 2025-03 (YYYY-MM)
2023
2024
2025
Number of Billboard Ranking Songs
Release Date (MM)
01
02
03
04
05
06
07
08
09
10
11
12


Billboard Rank vs Spotify Popularity for Top 100 Songs
Spotify Popularity
Billboard Ranking


Top Artists by Number of Ranking Songs
Artist
Sabrina Carpenter
Chappell Roan
Kendrick Lamar
Morgan Wallen
Teddy Swims
Shaboozey
Lady Gaga
Bruno Mars
Benson Boone
Alex Warren
The Weeknd
SZA
Lola Young
Doechii
Zach Top
The Marías
Summer Walker
Sleep Token
Riley Green
ROSÉ
Neton Vega
Myles Smith
Megan Moroney
Lil Tecca
Jordan Davis
Hozier
Gigi Perez
GELO
Don Toliver
Cody Johnson
Chris Brown
Carrie Underwood
Brandon Lake
Bhad Bhabie
Number of Top Songs

F. Instructions for running your code (10 points)
    a. Run main.py
        i. Collects up to 25 new songs from the Billboard Hot 100 and adds Spotify metadata
        ii. Re-run multiple times to collect all Billboard 100 songs and their artist metadata. Each run adds at most 25 songs and up to 25 artist top tracks, preventing API overuse.
        iii. This script updates the music_data.sqlite database automatically
    b. Run analyze.py
        i. Performs SQL joins and calculations across the database.
        ii. Writes a text summary to analysis_summary.txt
    c. Run visuals.py
        i. Generates and saves all 4 visualizations
        ii. All plots are saved as .png files in the current directory and also displayed

G. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)
    a. main.py
        i. Purpose: Builds the full data pipeline, combining Billboard scraping and Spotify API queries, and inserting results into the SQLite database.
        ii. Logic:
            1. Scrapes the Billboard Hot 100 list (top 100 songs).
            2. Filters out songs already present in the database.
            3. Fetches metadata for the next 25 unprocessed songs from Spotify.
            4. Limits song inserts to 25 per run.
            5. Limits top track inserts (from Spotify artist data) to 25 entries per run.
            6. Inserts albums, songs, artists, and artist top tracks into normalized database tables.
            7. Tracks and prints the number of new songs and artist top tracks inserted.
        iii. Input: None (invoked directly via 'python3 main.py')
        iv. Output: Updated 'music_data.sqlite' file and console confirmation of rows added.
    b. billboard.py
        i. top_hundred_songs()
            1. Input: None
            2. Output: dict - maps song name to a dictionary with Billboard ranking and artists list

3. Description: Scraped Billboard Hot 100 chart using BeautifulSoup and returns a structured dictionary of song data.

c. spotify_data.py

i. load_spotify_credentials(filepath)
   1. Input: filepath (str) - path to a text file with client_id and client_secret
   2. Output: dict - with client credentials
   3. Description: Loads Spotify API credentials from a local file

ii. get_spotify_client()
   1. Input: None
   2. Output: spotify.Spotify client object
   3. Description: Authenticates and returns a Spotipy client using Client Credentials flow

iii. fetch_spotify_data(billboard_data, limit)
   1. Input:
      a. billboard_data (dict) - Billboard song info from scraper
      b. Limit (int) - number of songs to process (default: 25)
   2. Output:
      a. Song_db (dict) - processed Spotify song metadata
      b. artist_db (dict) - top 5 tracks per artist
   3. Description: Searches each song on Spotify, collects metadata, and tracks per-artist top songs

d. database.py

i. create_music_db()
   1. Input: None
   2. Output: None
   3. Description: Creates the SQLite database schema with Songs, Albums, Artists, and ArtistTopTracks tables if they do not already exist

ii. song_rank_exists(rank)
   1. Input: rank (int) - Billboard ranking
   2. Output: bool - whether a song with that rank already exists
   3. Description: Prevents duplicate song entries across runs

iii. insert_album(name, release_date)
   1. Input: Album name (str), release_date (str)
   2. Output: album_id (int)
   3. Description: Inserts or retrieves album from Albums table

iv. insert_artist(name)
   1. Input: Artist name (str)
   2. Output: artist_id (int)
   3. Description: Inserts or retrieves artist from Artists table

v. insert_song(name, rank, popularity, album_id)
   1. Input: Song name (str), rank (int), popularity (int), album_id (int)

2. Output: None
3. Description: Inserts a new row in Songs table

vi. insert_artist_top_tracks(artist_id, track_list)
1. Input:
a. Artist_id (int)
b. Track_list (list of str)
2. Output: None
3. Description: Stores each artist's top 5 tracks into ArtistTopTracks table

e. analyze.py

i. get_rank_vs_popularity()
1. Input: None
2. Output: DataFrame with Billboard rank and Spotify popularity
3. Description: Supports scatter plot of rank vs popularity

ii. get_album_release_vs_rank()
1. Input: None
2. Output: DataFrame with album release date and Billboard rank
3. Description: Used for release timeline analysis

iii. get_top_artists_by_song_count()
1. Input: None
2. Output: DataFrame of artist name and number of top 100 songs
3. Description: Supports bar chart of top artists by volume

iv. get_artist_popularity_sum()
1. Input: None
2. Output: DataFrame of artist name and total popularity score
3. Description: Supports pie chart of artist popularity distribution

v. export_summary_text()
1. Input: None
2. Output: Writes analysis_summary.txt
3. Description: Exports top 10 artists by popularity and count as a readable text file (analysis_summary.txt)

f. visuals.py

i. Each graph function:
1. Input: SQLite cursor (cur)
2. Output: .png file & displayed plot
3. Description: Creates and saves a specific chart:
a. Graph_scattor_rank_vs_popularity() - scatter plot (rank_vs_popularity.png)
b. Graph_scatter_album_release_rank_num() - scatter plot by month/year (rank_by_album_release.png)
c. Graph_bar_top_artists_by_song_count() - horizontal bar chart (top_artists_by_song.png)

d. Graph_pie_artist_popularity_sum() - pie chart for top 10 artists (artists_by_popularity.png)

H. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
3/27/2025	Needed API	https://api.spotify.com	Yes
3/27/2025	Needed data source to narrow song selection	https://www.billboard.com/charts/hot-100/	Yes – Narrowed spotify requests to top 100 songs in rankings
3/27/2025	Trouble remembering web scraping syntax	Runestone Academy Python for Everybody - Interactive Textbook	Yes
3/29/2025	Spotify API usage and authentication	https://spotipy.readthedocs.io/en/2.22.1/	Yes – Used Spotipy to securely access data
4/1/2025	SQLite file locking and preventing simultaneous access	Stack Overflow & official SQLite FAQ	Yes – Helped avoid database access errors
4/2/2025	SQL JOIN syntax across multiple tables	https://www.sqlitetutorial.net/sqlite-join/	Yes – Successfully joined songs, artists, albums
4/3/2025	Customizing graph labels + other details	https://matplotlib.org/stable/plot_types/index.html	Yes – Allowed for greater understanding of code and customization
4/3/2025	Finding enough unique colors with high contrast for graphs	https://htmlcolorcodes.com/color-names/	Yes – Provided a wider variety of colors to choose from