

SI 206: Data-Oriented Programming
12 December 2022

SI 206 Final Project Report

Group name: BJG Group - Brogan Kaczkofsky (brogan@umich.edu), Jessica Levine (jesslevi@umich.edu), Grace Garmo (ggarmo@umich.edu)

Link to repository: https://github.com/bkaczkofsky/si206_final_project

Overarching Question: How does the ranking and genre of top songs compare across platforms?

Project Goal: Our project goal is to understand some of the various factors through which songs are ranked across multiple platforms since each platform has their own criteria that go into ranking. We want to discover how rankings of the same song on different platforms compare to each other, and we also want to analyze the different genres of songs on each platform that compose the infamous 'Top 100'.

Achieved Goals:

Billboard Beautiful Soup: Find top songs in top 100 for current week (<https://www.billboard.com/charts/hot-100/>)

iTunes API: Calculate percentage of each genre in both top 100 charts (Pie chart) (<https://performance-partners.apple.com/search-api>)

Shazam API: Find the most Shazamed songs for current week (Top 100) (<https://rapidapi.com/apidojo/api/shazam>)

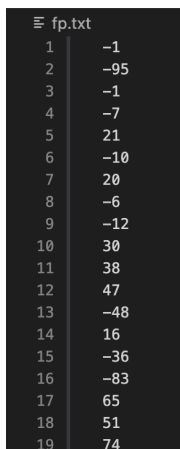
Overall: Analyze differences between Top 100 charts rankings of each song, and prevalence of genres of songs for both Billboard and Shazam via iTunes.

Problems Faced: A large source of our problems came from the ideation phase. Our group initially struggled to come to an agreement regarding the focus of the project, and it took some time, research, and back-and-forth ideation to eventually come to our project topic. Another problem we faced while working on this project occurred while working on the Beautiful Soup portion on the Billboard website. At first, when we created the class, we didn't realize that the stylization of the #1 song was different than the rest, and the #1 song wasn't being included in the dataset. We also faced an issue

where only the second song on the list was printing 99 times instead of printing the top 100 songs. Another common issue when analyzing data related to music has to do with repetitions - namely the same song on both websites and having artists appear multiple times for different songs if they have multiple songs on the top charts. We were careful to avoid duplicate data in our database by first creating dictionaries and lists of the songs, song IDs, artists, and genres for each platform and then utilized those to populate our database. We also faced issues adding the data into our database only 25 items at a time, so we decided to split up our Insert functions and prompt the user to continue running the code if they want to get all of the data into the databases, which measured out to running the code about 8 times. With this necessary repetition, we were careful to only run the code as much as necessary so as not to run out of query calls for our API (in the event that this happened, we created a new API key under a new account). Lastly, when creating the data visualizations, we encountered an issue where the text was overlapping each other, making the visualization difficult to read. We played around with changing the sizing of the shapes and text, but we eventually added a legend that we learned to make from watching a video tutorial, which helped to make the data easier to comprehend.

Database Calculations:

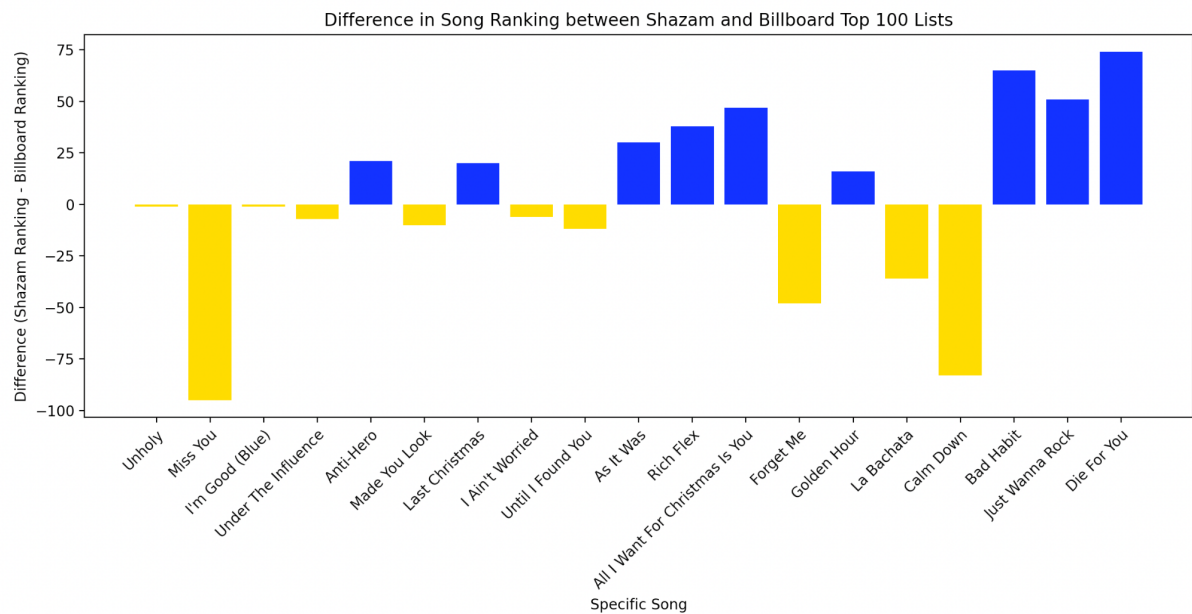
For songs on both charts, the difference between rankings over both platforms. We achieved these values by subtracting the Billboard ranking from the Shazam ranking: $\text{Shazam} - \text{Billboard} = \text{Difference}$, which meant that positive values were songs ranked higher on X. For example, the song Anti-Hero (song ID = 122) by Taylor Swift was ranked #22 on Shazam and #1 Billboard, making the difference = -21, whereas the song Die For You (Song ID = 189) by The Weeknd was ranked #89 on Shazam and #15 on Billboard, making the difference = 74



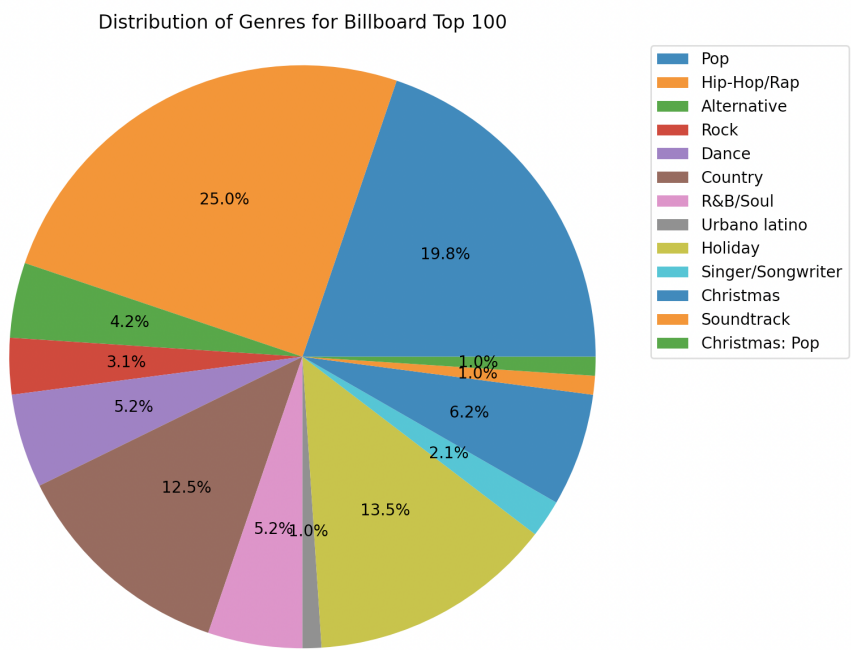
	fp.txt
1	-1
2	-95
3	-1
4	-7
5	21
6	-10
7	20
8	-6
9	-12
10	30
11	38
12	47
13	-48
14	16
15	-36
16	-83
17	65
18	51
19	74

Visualizations:

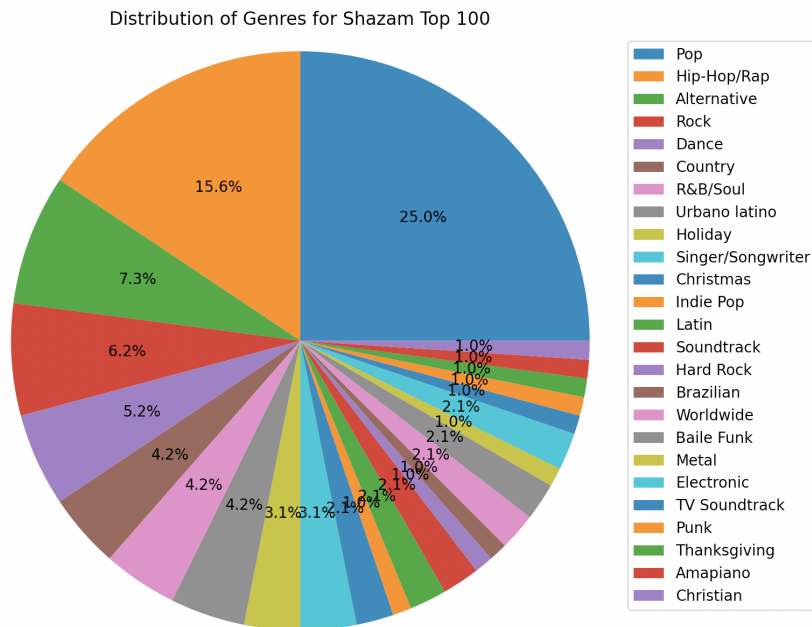
Bar chart to show the difference between rankings on Billboard.com and Shazam:



Pie chart of percentage of each genre for Billboard Top 100



Pie chart of percentage of each genre for Shazam Top 100



Instructions to Run the Code:

The code prompts the user with the option to Start Over, Continue, or Quit. When the user wants to initialize all of the tables, they should select Start Over, via the number 1. This command clears all previous tables and will create new ones to input the data. Subsequently, the user must then select Continue by entering the number 2 and repeating this process eight consecutive times in order to insert data into the tables 25 items at a time and get all of the data into the database. The user can Quit and exit the program if they do not want to Start Over or Continue by entering the number 3.

Documentation for Functions - include output and input for each:

def get_shazam_list(): this function does not take in any inputs, but rather utilizes the Shazam API to create a list (in increments of 20) of the top songs on Shazam. It returns a list of tuples containing the ranking, song title, and artist name for each song on the chart in a list called shazam_list.

def get_billboard_list(): this function does not take in any inputs, but rather scrapes the Billboard Top 100 Chart from the billboard.com website using BeautifulSoup and

returns a list of tuples that contain the ranking, song title, and artist name for each song on the chart in a list called `billboard_list`.

def get_itunes_list(songs_dict): this function takes in a dictionary of songs and iterates through each song, using the song title to find metadata on that song through iTunes. This metadata includes the genre and time (in milliseconds) of the song and this function returns a list of tuples with each tuple containing the song id, time, and genre.

def setUpDatabase(db_name): this function takes in the name of a database and returns `cur` and `conn` to set up the database we will use later, called `fp.db`.

def create_song_table(cur, conn): this function takes in the `cur` and `conn` from the database and creates a table called `songs` (it also drops it if it already exists so as to not repeat data), with the song id, song title, and artist name for every song from both Billboard and Shazam.

def count_songs_in_songs(cur): this function takes in the `cur` from the database and selects all elements in the `songs` table in order to get a count of how many songs exist in the table by returning the length of the table.

def add_songs(shazam_list, billboard_list, cur, conn): this function takes in lists `shazam_list` and `billboard_list` and the `cur` and `conn` from the database and generates and returns a dictionary of all existing songs, where the key is the song title and the value is a tuple of the song id and artist name. Simultaneously, this function inserts these songs into the `songs` table in the database in increments of 25.

def create_shazam_table(cur, conn): this function takes in the `cur` and `conn` from the database and creates a table called `shazam` (it also drops it if it already exists so as to not repeat data), with the song id and ranking for the top 100 songs from Shazam as the output in the database.

def count_songs_in_shazam(cur): this function takes in the `cur` from the database and selects all elements in the `shazam` table in order to get a count of how many songs exist in the table by returning the length of the table

def add_songs_shazam(songs_dict, shazam_list, cur, conn): this function takes in the dictionary `songs_dict`, list `shazam_list`, and the `cur` and the `conn` from the database and inserts the song id from the `songs_dict` and the ranking from the `shazam_list` into the `shazam` table in the database in increments of 25.

def create_billboard_table(cur, conn): this function takes in the cur and conn from the database and creates a table called billboard (it also drops it if it already exists so as to not repeat data), with the with the song id and ranking for the top 100 songs from Billboard.com as the output in the database.

def count_songs_in_billboard(cur): this function takes in the cur from the database and selects all elements in the billboard table in order to get a count of how many songs exist in the table by returning the length of the table.

def add_songs_billboard(songs_dict, billboard_list, cur, conn): this function takes in the dictionary songs_dict, list billboard_list, and the cur and the conn from the database and inserts the song id from the songs_dict and the ranking from the billboard_list into the billboard table in the database in increments of 25.

def create_itunes_table(cur, conn): this function takes in the cur and conn from the database and creates a table called itunes (it also drops it if it already exists so as to not repeat data), with the song id, time (in ms), and genre of the top songs from iTunes as the output in the database.

def count_songs_in_itunes(cur): this function takes in the cur from the database and selects all elements in the itunes table in order to get a count of how many songs exist in the table by returning the length of the table.

def add_songs_itunes(itunes_list, cur, conn): this function takes in the list itunes_list and the cur and the conn from the database and inserts the song id, time, and genre of each song into itunes table in the database in increments of 25.

def get_difference(cur, conn): this function takes in the cur and the conn from the database and selects the ids and ranking from the shazam and billboard tables to find where the song ids are the same (and thus the songs are the same) using a join. This function returns the difference in ranking (shazam ranking - billboard ranking) in a list called difference_list while also utilizing this list to create a visual representation of the differences as a bar chart.

def write_txt(matches, filename): this function takes in the list difference_list and the output file name and opens the file, writing each value in the difference_list on a new line in the output file.

def get_shazam_genres(cur, conn): this function takes in cur and conn from the database to create a pie chart displaying the distribution of genres of musical tracks on the Shazam Top 100 chart which shows genres and the percentages of each genre that appear on the chart

def get_billboard_genres(cur, conn): this function takes in cur and conn from the database to create a pie chart displaying the distribution of genres of musical tracks on the Billboard Hot 100 chart which shows genres and the percentages of each genre that appear on the chart

Resources Used:

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/11/2022	iTunes API	https://performanc e-partners.apple.com/search-api	Yes, allowed us to find data about specific songs
12/11/2022	Shazam API	https://rapidapi.com/apidojo/api/shaz am	Yes, allowed us to find data about most Shazamed songs by week
12/11/2022	Billboard Hot 100	https://www.billboa rd.com/charts/hot-100/	Yes, allowed us to scrape data and obtain Top 100 songs on the US charts
12/11/2022	JSON Formatter	https://jsonformatt er.org/	Yes, made it much simpler to view formatted data
12/11/2022	SI 206 Project 2	GitHub/Canvas class resources	Yes, referenced past project to implement Beautiful Soup and help with web scraping
12/11/2022	SI 206 Discussion 11	GitHub/Canvas class resources	Yes, helped with working with databases and JOIN statements
12/11/2022	SI 206 Discussion 8	GitHub/Canvas class resources	Yes, helped us to format Beautiful Soup
12/11/2022	Stack Overflow	https://stackoverflow.com/	Yes, helped to solve general coding questions

12/11/2022	SQL Tutorial	https://www.sqltutorial.org/sql-window-functions/sql-row_number/	Yes, used to know how to use the ROW_NUMBER() to assign a sequential number to each row in a query result set
12/11/2022	GeeksforGeeks	https://www.geeksforgeeks.org/sql-tutorial/	Yes, read SQL tutorial and got help on queries
12/11/2022	W3 Schools	https://www.w3schools.com/sql/sql_join.asp	Yes, assisted with SQL JOIN statements
12/12/2022	Tutorials Point	https://www.tutorialspoint.com/how-to-add-a-legend-to-a-matplotlib-pie-chart#:~:text=Matplotlib%20with%20Python&text=Make%20a%20list%20of%20labels,by%20changing%20the%20axis%20limits.	Yes, helped to add a legend to a Matplotlib pie chart
12/12/2022	GeeksforGeeks	https://www.geeksforgeeks.org/diverging-bar-chart-using-python/	Yes, helped to create diverging bar chart with positive and negative values