Team: CovidSonicsAnalytics
Manvitha Mogalayapalli, Kayla Wehner
SI 206
12 December 2023

SI 206: Final Project Report

1. Goals and Scope of Project:

According to the World Health Organization, the COVID-19 pandemic led to a 25% increase in the prevalence of depression and anxiety worldwide (World Health Organization). The pandemic's expansive effects on mental health-related issues prompted our investigation into its potential effects on the average mood of the most popular songs. Using the Billboard Charts API, we extracted data about the top 100 songs –e.g. song rank, title, artist– during time periods before and during the COVID-19 pandemic. To represent the period right before the global pandemic, we chose December 1 2019 as an estimation since there is not a clear understanding of the exact date in which the first COVID-19 case occurred. To examine the lasting effects of the pandemic, we extracted data for December 1 2020 since there was an additional peak of COVID-19 cases and roughly a year had passed since the initial outbreak. Moreover, we used the Spotipy API to analyze audio features –e.g. valence, danceability– of the top songs returned from the Billboard API function. By gathering data about the positivity-related audio features of top songs before and during the pandemic, we aimed to analyze the possible effects of the COVID-19 pandemic on individuals' mental health in relation to their music taste.

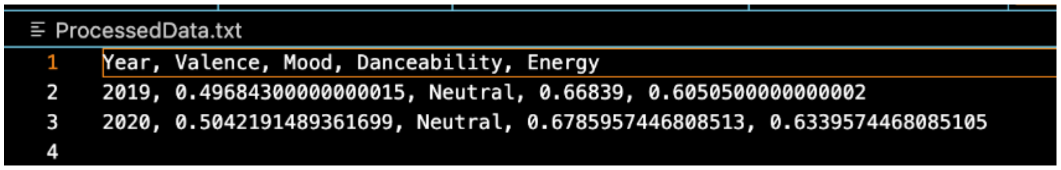
Source:
World Health Organization. (2022, March 2). *Covid-19 pandemic triggers 25% increase in prevalence of anxiety and depression worldwide*. World Health Organization. https://www.who.int/news/item/02-03-2022-covid-19-pandemic-triggers-25-increase-in-prevalence-of-anxiety-and-depression-worldwide

2. Achieved Goals and Gathered Data:

Fulfilling our initial plan, we extracted data from the Billboard Charts and Spotipy API to gather data about positivity-related audio features (e.g. valence) of the top 100 songs before and during the COVID-19 pandemic. As stated earlier, the Billboard Charts API returned information about the top 100 songs (e.g. song title, rank, artist) during the specified time period, while the Spotipy API returned data about the audio features of those top songs (e.g. valence, danceability). Although we initially hypothesized that the average valence –i.e. 'positivity'– level would decrease during COVID –due to the increased prevalence of mental-health related concerns– our gathered and processed data alluded that there was no significant difference in the average valence level of songs during the two time periods (i.e. before and during the COVID-19 pandemic).

3. Problems Encountered:
Initially, we faced a few problems while attempting to understand and implement the API-specific classes and attributes. To explain, the Billboard Charts API has its own specifically designated classes, arguments, and attributes, and such formatting-related requirements caused some initial confusion while we attempted to extract needed data from the API. Similarly, the Billboard Charts API information was taken from a public GitHub repository which required manually downloading the repository file to extract the data, and this process presented a few issues within our shared project repository since some necessary files would initially appear as empty. Another big problem we encountered was issues with implementing spotipy inorder to properly implement it we needed to create an account in spotify for developers and develop our own client id and client secret then using this we needed to create a spotipy object to access all the required functionalities. We also encountered problems with double string data with repeating songs in the two years of billboard data. To combat this we created a table that made song ideas for every unique song and the tables with the rankings included song ids instead of strings.

4. Data Calculations:



```
ProcessedData.txt
Year, Valence, Mood, Danceability, Energy
2019, 0.49684300000000015, Neutral, 0.66839, 0.6050500000000002
2020, 0.5042191489361699, Neutral, 0.6785957446808513, 0.6339574468085105
```

Figure 1: Text file depicting the average valence, mood, danceability, and energy level of top songs during 2019 and 2020. [These values were calculated from the average_song_analysis_features from the process_data.py file.]

5. Data Visualization:

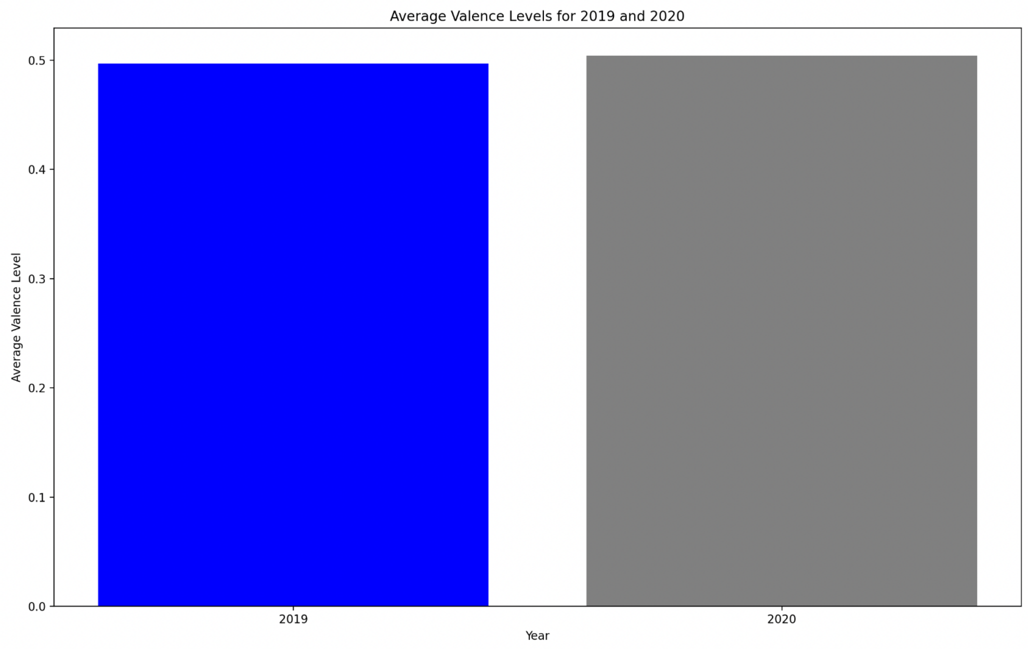


Figure 2: Histogram of Average Valence Levels of 2019 and 2020

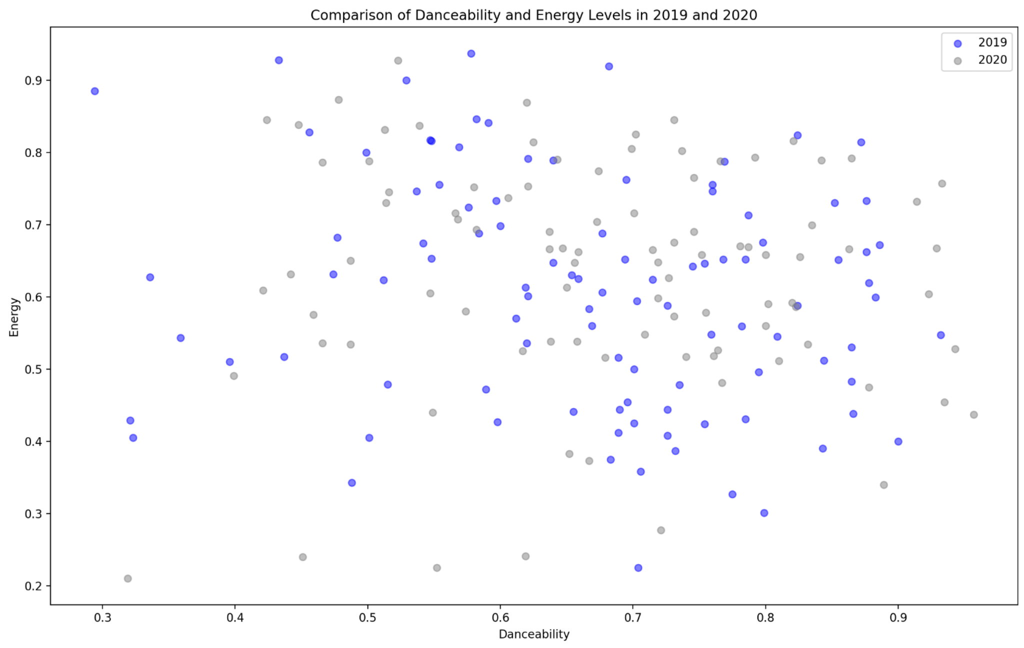


Figure 3: Scatter plot of danceability and energy for each year (i.e. 2019, 2020)

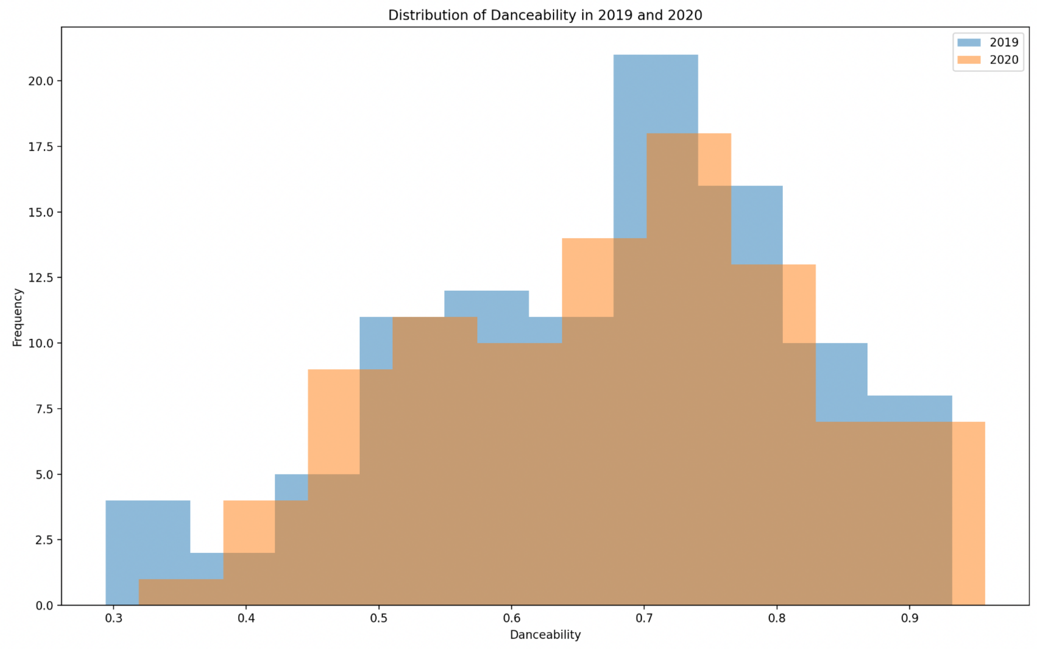


Figure 4: Distribution of danceability level of songs during 2019 and 2020

6. Code-Running Instruction:

Before running the code, make sure Python is installed on your system. The script requires several Python libraries: spotipy, billboard.py, requests, and matplotlib. If these are not already installed, you can install them using pip. Open your terminal or command prompt and execute the commands: pip install spotipy, pip install billboard.py, pip install requests, and pip install matplotlib. Accessing the Spotify API requires a Client ID and Client Secret. You can obtain these by registering your application on the Spotify Developer Dashboard. To make it easier, we have provided a client ID and secret in the script. Start by running the billboard.py file. To gather all necessary data, you need to run this file ten times. After completing this step, run the process_data.py file. This script will process the data stored in the database and output the results in the ProcessedData.txt file. Lastly, to view our visuals run visualization.py.

7. Function Documentations:

billboard.py:

1. **create_data_base(database_name):**
   Input: database_name: string representing the name of the database to be created.
   Output: Returns a tuple containing the cursor (cur) and connection (conn) to the SQLite database.
2. **song_table(chart, cur, conn)**

Input: billboard API chart, cur: cursor object, conn: connection object.
Output: returns None, updates the database with song information.

3. **billboard_hot_100(date, cur, conn)**
   Input: date: A string representing the date, cur: cursor object, conn: connection object.
   Output: Returns a list of songs with their details fetched from the Billboard Hot 100 chart for the given date.

spotify.py:

1. **create_spotify_oauth(client_id, client_secret, redirect_uri)**
   Input: client_id: the spotify client ID, client_secret: the spotify client secret, redirect_uri: The redirect URI set for Spotify API authentication.
   Output: Returns a SpotifyOAuth object configured with the credentials
2. **get_spotify_client(client_id, client_secret, redirect_uri)**
   Input: client_id: the spotify client id, client_secret: the spotify client secret, redirect_uri: The redirect URI for Spotify API authentication.
   Output: Returns a spotipy client instance
3. **get_track_features(id)**
   Input: id: The Spotify ID of the track.
   Output: Returns a dictionary with the features of the specified track.

5. **get_audio_features(sp, track_ids)**
   Input: sp: An authenticated spotipy client instance, track_ids: A list of track IDs for which audio features are requested.
   Output: Returns a list of dictionaries, containing the audio features of each track
6. **process_audio_features(audio_features)**
   Input: audio_features: A list of dictionaries containing audio features of tracks.
   Output: Returns a list of processed audio feature data.
7. **create_mood_table(cur, conn)**
   Input: cur: The database cursor object, conn: The database connection object.
   Output: returns None, Creates a Mood table in the database.
8. **assign_mood(track_data)**
   Input: track_data: A list of dictionaries containing track information features.
   Output: Returns the input track_data list with mood information added to each track's data.
9. **get_track_id(sp, title, artist)**
   Input: sp: An authenticated spotipy client instance, title: The title of the song,
artist: The name of the artist.
   Output: Returns the Spotify track ID as a string.
10. **clean_song_name(song_name)**
   Input: song_name: A string representing the name of a song.

Output: Returns the cleaned song name with certain characters replaced or removed.

11. clean_artist_name(artist_name)

Input: artist_name: A string representing the name of an artist.

Output: Returns the cleaned artist name with adjustment.

12. enhance_track_data(sp, track_list)

Input: sp: authenticated client instance. track_list: A list of tuples containing song details.

Output: Returns a list of enhanced track data with added audio features and mood information.

13. insert_spotify_data(cur, conn, spotify_data)

Input: cur: cursor object, conn: connection object, spotify_data: A list of tuples containing Spotify track data.

Output: returns None, Inserts Spotify data into a database table.

14. join_spotify_billboard(cur, conn, billboard_table_name)

Input: cur: The database cursor object , conn: The database connection object, billboard_table_name: The name of the Billboard table within the database.

Output: returns none, Creates a joined table in the database combining Billboard and Spotify data.

process_data.py:

1. **average_song_analysis_features(db_filename, table_2019, table_2020, audio_features_list)**:

   Input: **db_filename**: The name of the database that includes the joined data ('Billboard_Hot_100_Database.db'), **table_2019**: The table in the aforementioned database which includes data about top songs from 2019, **table_2020**: The table in the aforementioned database which includes data about top songs from 2020. **audio_features_list**: a list containing the names of the numeric audio features (i.e. valence, danceability, energy) from the database.

   Output: **tuple_average_values_2019**: A tuple containing the average values of the valence, danceability, energy, and mood levels of the top song for 2019 and 2020.

visualization.py:

1. **valence_histogram_visualization(db_filename, table_2019, table_2020)**:

   Input: **db_filename**: The name of the database that includes the joined data ('Billboard_Hot_100_Database.db'), **table_2019**: The table in the

aforementioned database which includes data about top songs from 2019,
table_2020: The table in the aforementioned database which includes data about top songs from 2020.
Output: No return value, but a histogram is generated to depict the average valence values of songs from 2019 and 2020.

2. **Danceability_energy_scatterplot(filename, table_2019, table_2020):**
   Input: filename: string name of the file, table_2019: name of the table with billboard songs from 2019 and their audio features, table_2020: name of the table with billboard songs from 2020 and their audio features
   Output: produces a scatterplot with energy and danceability from both dates
3. **danceability_distribution_histogram(db_filename, table_2019, table_2020):**
   Input: filename: string name of the file, table_2019: name of the table with billboard songs from 2019 and their audio features, table_2020: name of the table with billboard songs from 2020 and their audio features
   Output: produces a histogram with the frequencies of danceability values for each year

8. Resources:

12/11/2023: Joining the audio features table and the billboards table, A big issue we encountered was attempting to join the two tables they would not join together efficiently and for some reason was not giving us an output of 100 for each table after running 10 times, So we asked Chat Gpt to help us figure out the issue, it gave us a possible solution, which was too get the last valid id from the table and set a condition. We implemented this in the Join_spotify_billboard() function and it allowed us to solve the issue and produce the right tables.

12/8/23: While working with the Billboard Charts API, we encountered a few issues with structuring and implementing functions that called and stored song data into the database in the necessary increments (e.g. 25 or fewer items in each run). However, this issue was resolved after we attended SI 206 office hours since we were provided with advice about possible ways our code could factor in such requirements (e.g. using index).

12/7/23: While attempting to extract information about the top 100 songs from the Billboard Charts API, we encountered a few issues with actually extracting the data from the API since our data was taken from a public GitHub repository (https://github.com/guoguo12/billboard-charts.git). Attending SI 206 office hours resolved this issue since we gained insight into the most effective ways to extract and organize data taken from the API, which was crucial for our later work in the project.

11/30/2023: Figuring out how to use spotify API was very difficult. In order to combat this issue we used many resources including the website below which helped us figure out how spotify api works and how we can access it.
https://medium.com/@maxtingle/getting-started-with-spotifys-api-spotipy-197c3dc6353b

11/30/2023: another issue we encountered was trying to navigate the different functions that can be used to extract data from spotify. So we used https://spotipy.readthedocs.io/en/2.22.1/ to help understand the functions that can be used to get all the right information we needed such as audio features.

Github link: https://github.com/Manvitham123/SI206-FInalProject