Lilly Goebel, Alina Parr, & Saira Rathod
SI 206 - Data Oriented Programming
December 12, 2022

Final Report - The Music Mates

Our Goals:

Our main goal of the final project was to use the BillBoard's top 100 artists in order to find out whether or not each artist is going on tour and the details about their event as well as finding out each artists most popular song, its release date, how long the track is, and whether or not the song is explicit.

Our Achievements:

Our main achievement throughout this project was that we were able to generate a table of the different genres from the top 100 artists. We found that Pop is the most common genre from our top 100 artists data. This was done by gathering the top 100 BillBoard artists through BeautifulSoup, and passing that information into two different API's. In our fourth and final file, combined.py, we were able to import functions from our previous files and use the lists of tuples returned to store 25 pieces of data at a time. We then used Select statements to calculate data, including the percentage of explicit songs (73%), percentage of the top 100 artists that are currently on tour (52%), the average length of the songs (3 minutes and 32 seconds), and the top country our artists are going to be performing in (The United States). We were also able to create and write to a CSV file to document all of our calculations.

Our Problems:

The first problem we ran into was finding three APIs that were both free and related to one another. Once we found these APIs, we spent some time looking over the documentation of each, to avoid errors in our code. One problem that we struggled with was loading 25 items at a time into our database. We solved this issue by making a new file (combine.py), where we called each individual function that returned the tuples we created from the two API's and the beautiful soup file. After that, we made a for loop that inserted our information from our tuples into the database 25 rows at a time. This made loading our data more efficient, because we only had to run one file (combine.py) instead of each individual file and having multiple for loops. Another

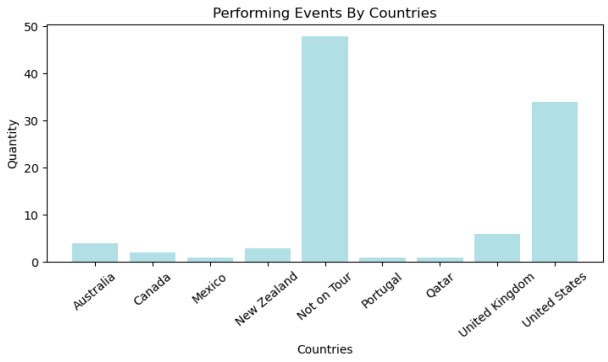
problem that we encountered during this project was having duplicate string data in our database. For example, we documented whether or not each specific song was explicit. In order to rid this duplicate data, we created a separate file, called explicit_information, to store each string ('explicit' or 'not explicit') into an id of 0 or 1. Another problem we encountered was figuring out how to best visualize the average length for the 100 songs. We decided that the best way to display the average song length was by using a normal distribution and calculating the standard deviation.

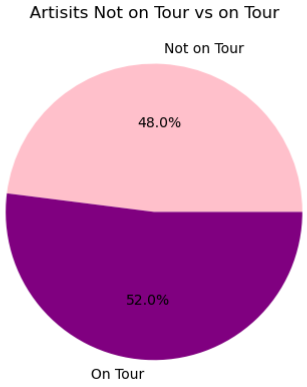
Calculation File:

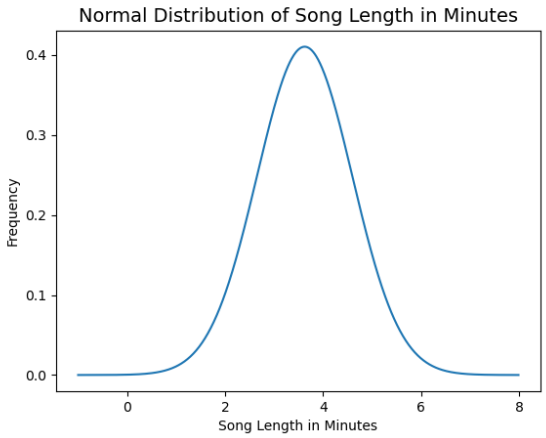
https://github.com/lillygoebel7/Final206Project/blob/main/combined.py

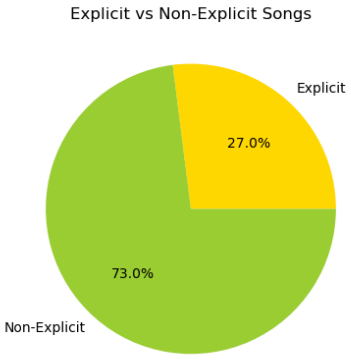
https://github.com/lillygoebel7/Final206Project/blob/main/music_data.csv

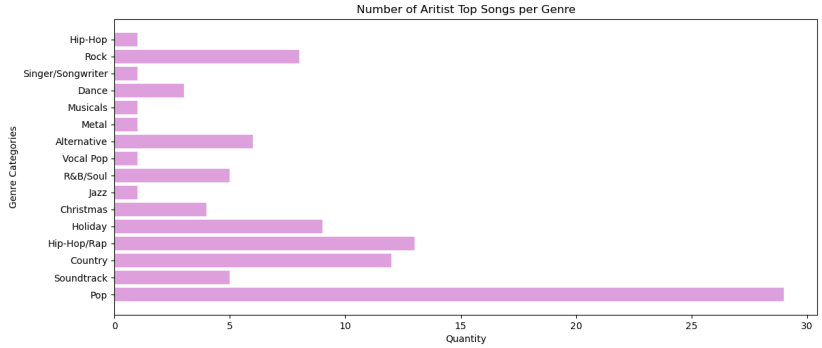
Visualizations:











Instructions for Running Our Code:

In order to run our code, run the combined.py file four times (each run through will add 25 rows to our database, totaling to 100). While running, each of our five visualizations will pop up, click the X in the top left corner to see the next visualization (they will also be saved in the images folder).

Function Documentation:

Our code consists of four python files and a database. Our first python file is called billboard_artists.py. This file begins by opening up our database, music_information.db. Then, the function read_billboard_data takes in the BillBoard Top 100 Artists URL. The URL is then used to create a Soup object, which is parsed through to return a list of tuples containing the artist_id and artist name. This function also creates a table in the database, called artists, which is added to in a later file.

Our second file, bands.py starts off by opening the database. We created our bandsintown function that takes in 3 parameters: the returned artists names from our billboard data file, conn, and cur. Next, we created and committed 3 separate tables: 1 for all of our data and then 2 other reference tables were created so that we don't have duplicate strings. We then inserted data into our Not_On_Tour table so that 0s are inputted into the table when the artist isn't on tour therefore having null data in those rows. After that we requested information from the bandsintown API for each of the Top 100 artists in our billboard data. If the artist didn't have any upcoming events then we would get an empty list and insert information into the database that correlated to an artist not on tour. If an artist did have upcoming events then we created multiple variables by

indexing the dictionary. We imported data into a separate reference table, country_table, that connected the country_id to the specific country that artists were performing in. Lastly, we created a tuple that included the artist_id which specified a number that correlated to each specific artist, the date_time which identifies the date and time of the event, and the country id which references the specific id in the country_table. We finished by adding the respective tuple for each artist into a list, called bands_list.

Our third file, iTunes.py, also begins by opening the database. Then, through the iTunesSearch function we created a table, itunes_songs, that included the specific artist_id, trackName that referenced the artist's top song, the release date of that specific song, and whether or not the track was explicit. Then, for each artist in the billboard_data we requested information from the iTunes API that allowed us to create the variables we are importing into the database table. We created a separate reference table, genre_table, that had each genre corresponding to a number that we used in the itunes_song table. We also created an explicit table to reference whether a song was explicit(1) or clean(0). We created a list of tuples where each tuple contains the artists_id, track_name, genre, release_date, track_time, and track explicitness for each artist and finally added each tuple to the itunes_lst.

Our fourth file, combine.py, starts off by importing the first three files (billboard.py, bands.py, and itunes.py). For all three files, we implemented a for loop to enter 25 pieces of data into our database using an INSERT OR IGNORE statement. Then, we started our calculations by pulling from our database using SELECT statements. Using COUNT statements, we gathered data on whether or not the song was explicit. We then sorted the songs by genre to determine which genre is the most common. Next, we calculated how many artists are on tour, and which country their tour will be in to determine which country is the most popular for our artists' future events. Finally, we calculated the average length of the songs and the standard deviation, which was used to create a normal distribution plot. Next, we wrote our calculations into our CSV file. We then created 5 visualizations based on our calculations using matplotlib. We used a pie chart to visualize the percentage of songs that are explicit, and the percentage of artists who are currently on tour. Then, we used a bar graph to show which genre is most popular, and which country is most popular for our artists' performances.

Resources:

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/5/22	We had a hard time comprehending the different parts of the dictionary	https://jsonformatter.curiousconcept.com/#	Yes. We were able to put our dictionary data into the formatter in order to organize the dictionaries allowing us to see the different parts of the outermost dictionary. This allowed us to create and save different variables from the dictionary.
11/30-12/5	We struggled at the beginning finding an API that was free and that supported our goal	https://github.com/public-apis/public-apis	Yes. We found 2 API's that are free and both take in an input of an artist's name
12/9	We had a hard time calculating the average and standard deviation of song times	https://www.w3schools.com/sql/sql_count_avg_sum.asp https://www.statology.org/plot-normal-distribution-python/	Yes. We figured out how to calculate standard deviation and what type of modules to import in order to make a normal distribution of song length for the 100 songs.
12/6-12/7	Needed help on how to use SQL	https://runestone.academy/ns/books/published/Fall22-SI206/database/toctree.html	Yes. We reviewed runestone academy for how to SELECT and COUNT data
12/10	We needed help learning how to use plotlib	Matplotlib Lecture Slides – Lecture 21	Yes. We were able to create five different visualizations after reviewing Professor Ericson's slide deck and w3 schools as listed above.