Roger Barber & Alex Brand

SI 206 – Data Oriented Programming

Dr. Ericson

11/29/22

Spotify & Genius API – Relationships Between Word Usage & Song Metrics

**Goals**

**Problems / Limitations**

Some major issues we ran into with the Spotify API were the nature of the Billboard Hot 100 that we decided to work with. For one, the data set changes every week, so collecting 100 items proved to be difficult, as the set we were trying to examine changed regularly and revealed new issues. If I were to go back and change our project, it would have been collecting data every week on the Top 100 and storing that into a database – but that would get a bit large pretty quickly.

Secondly, we noticed that there is a lot of seasonal and timing aspects that change the composition of the Hot 100. Taylor Swift and Drake both released new albums before our project, so a large proportion of the top 100 songs were from their albums. This significantly skews the genre data we collected and most likely has a great effect on the music feature averages as well. And since this is the holiday season, there are a few Christmas songs that have crept back onto the charts, which is not representative of what the chart contains for the rest of the year. I think the goal of our project is a great idea, but this would require a long period of data collection and careful examination of external happenings such as album releases, seasonal music, and viral trends.

**Data Calculations & Visualizations**

**Instructions for Running Code**

First – run the Spotify API to collect data for usage by the Genius API and to create visualizations and tables for the Spotify API.

For the Spotify / Spotipy API:

* Open up both spotifyAPI\_1.py and spotifyAPI\_2.py.
* Both of these files are structured to have a main function call at the end. There is no need to enter in any external variables, and there are no user inputs required beyond running the code.
* Run spotifyAPI\_1.py first:
  + The Spotipy module creates an Oauth2 instance using my client secret and client ID that accesses the Spotify API using an object named ‘sp’ – all the call methods are ran on that object using methods native to the module.
  + If you want to check if API pull works, be sure the database file is empty. We will do so before submitting.
  + Run the program until the third table indicates it is full. The longest table pull will be 100 items which will take 4 runs of 25 maximum items. Approximately 8 runs should be the maximum you need to populate all three tables.
  + The program will indicate whether or not the table has been created from scratch.
  + It will also indicate whether the table has existing items, and it will also indicate that the function has ran out of items to parse from the API data.
  + Finally, it has a failsafe method that skips operation if you are attempting to add data to a table that is full.
* Run spotifyAPI\_2.py second:
  + Again, there should be no text files besides the reports and instructions after we submit, but double checking for old files ensures the data being used in the table matches the JSON files stored.
  + This program only needs to be ran once.
  + The program will populate three visualizations while running, for three different measurements: an average of the four musical scores we were testing for, a count of songs by genre, and a count of songs by artist.
  + You will need to exit out of the graph visualizations for the program to continue running, and for it to end.

**Function Documentation**

Roger // Spotify API

* spotifyAPI\_1:
  + artistIndex(track\_list)
    - This function creates an artist index file for use in a table.
    - Inputs: a “track\_list” item that is a dictionary object created from the Spotipy module by using a predetermined track ID for the Billboard Hot 100.
    - Outputs: a dictionary that contains an integer as a key and an artist name as a value.
  + genreIndex(track\_list, sp)
    - This function creates a genre index file for use in a table.
    - Inputs: a “track\_list” item that is a dictionary object created from the Spotipy module by using a predetermined track ID for the Billboard Hot 100. Also, a Spotipy object (“sp”), an instance of the Spotipy module.
    - Outputs: a dictionary that contains an integer as a key and a genre name as a value.
  + spotipyScouring(track\_list, artist\_index, genre\_index, sp)
    - This function creates a multi-level dictionary that contains the top 100 songs as keys, with their artist index, genre index, and song features as keys in a nested dictionary.
    - Inputs: The same above track\_list dictionary, the previously generated artist\_index dictionary, the previously generated genre\_index dictionary, and a Spotipy module object (‘sp’).
    - Outputs: the same as described in the first statement.
  + createSongTable25(song\_dict)
    - Creates a Top 100 song table.
    - Inputs: a dictionary, the output of spotipyScouring.
    - Outputs: creates a table or adds to a table, 25 items at a time.
  + createGenreTable25(genre\_dict)
    - Creates a Genre Index table.
    - Inputs: a dictionary, the output of genreIndex.
    - Outputs: creates a table or adds to a table, 25 items at a time. Returns True if no items are left to be added.
  + createArtistTable25(artist\_dict)
    - Creates an Artist Index table.
    - Inputs: a dictionary, the output of artistIndex.
    - Outputs: creates a table or adds to a table, 25 items at a time. Returns True if no items are left to be added.
  + tableWriter25(artist\_index, genre\_index, track\_features)
    - Consolidates three table creation functions to run in sequence and only 25 items per run overall.
    - Inputs: 3 dictionaries; one for artist index, one for genre index, and one for track features.
    - Outputs: Three tables, one for each dictionary. Nothing returned to program space.
  + main()
    - Runs all of the above functions sequentially to add items to all three tables at the same time, 25 items per run, per table.
    - Inputs: none.
    - Outputs: none.
* spotifyAPI\_2:
  + scoreAverage(output)
    - Writes a text file in JSON format that averages the four scores amongst all 100 songs.
    - Input: the filename you want for the stored data, including file extension.
    - Output: a text file with the average scores for all songs.
  + genreCount(output)
    - Writes a text file in JSON format that counts the number of songs in each genre amongst all 100 songs.
    - Input: the filename you want for the stored data, including file extension.
    - Output: a text file with the counts for all genres present in the data.
  + artistCount(output)
    - Writes a text file in JSON format that counts the number of songs for each artist amongst all 100 songs.
    - Input: the filename you want for the stored data, including file extension.
    - Output: a text file with the counts for all artists present in the data.
  + bar\_graph(filename)
    - Creates and displays a bar graph of the data for scoreAverage.
    - Inputs: a filename for where you want the picture stored.
    - Outputs: displays the graph within the program, and saves a picture in your file directory.

**Works Cited / Outside Resources Used**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Issue | Resource Used | Did it Work / Results |
| 11/20/22 | Learning to use Spotipy | https://towardsdatascience.com/extracting-song-data-from-the-spotify-api-using-python-b1e79388d50 | Yes – I successfully learned how to use the API and gave me examples on how to use it to pull information out. |
| 11/20/22 | SQL Help | https://www.w3schools.com/sql/ | Helped a little, referred to our notes for SQLite specific syntax. |
| 11/20/22 | Spotify Documentation / API Structure | https://developer.spotify.com/documentation/web-api/reference/#/ | Yes – helped me understand the structure of my API pulls so I could extract data. |
| 11/20/22 | Spotipy Documentation | https://spotipy.readthedocs.io/en/2.21.0/ | Yes – introduced me to the methods in the module. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |