Tianny Lu
Barbara Ericson
SI 206—005
December 15, 2018

Final Project—Spotify API

Goals:

For my final project, my original plan was to use the Spotify API in order to collect data on popular Christmas albums and plot them on a scatter plot based on the age of the album and the average popularity score. However, due to some problems I encountered I had to change the goal of my project. For the new goal, I planned on using the endpoint https://api.spotify.com/v1/albums/ and the album id, which is the Spotify ID for the album, in order to conduct the search. I planned to search on each album and pull the the objects "name", "artist name", "track name", "track number", and "track duration". These objects would then be written into a SQLite database. After caching the results into the database, I planned on find the average length of a track for each album. I would then write the album name and the a bar graph that would plot each data based on the name of the album (x-axis) and the average track length (y-axis).

Goals Achieved:

Only some of my original goals were achieved because I changed my project proposal due to some problems I had encountered, which I will discuss in the next section. However, in the original plan, I was able to search on 8 popular Christmas albums using Spotify's API that searches the album ID. However, I used a Python library, Spotipy, to conduct the search, which uses a `.album( )` method. The method requires the album ID number, which is similar to how the original Spotify API conducts the album search. With the new plan, I was able to pull specific objects, such as album name, artist name, track name, track number, and length of track, from the results and place them into a new dictionary. From there, I would take the list of dictionaries named `spotifyList` and write each track and its objects to the database table `Albums`. In addition, I was able to write these objects into a database and only select what I needed from the database in order to calculate the average track length per album, which was then written into a CSV file with the album name and the average track length. From there, I

was able to create a new dictionary that contained the album name as the keys and the average track lengths as the values. Using this dictionary, I was able to create a visualization via Matplotlib and show a bar chart that plotted the average track length for each album.

Problems/Issues Encountered:

Initially, I planned on finding the average popularity score and age of each album, which would then be plotted in a scatter plot to show if the popularity of an album has withstood time. However, when I began to collect the data, I noticed that it would be difficult for me to find out the date the data was collected on as there it no timestamp object that is part of the results. In addition, even if I searched the same albums every day, there would not be enough data points collected, thus I would not be able to satisfy one of the requirements of the project which was to store at least 100 interactions in the database. As a result, I had to change the scope of the project in order to ensure that I would collect at least 100 interactions from Spotify.

Another issue I encountered was when I was trying to authorize my client key and secret client key. Initially I was using Spotify's own documentation on authorizing and the authorization flow. They even had a GitHub repository that included some examples on how to authorize. However, the code and examples that they offered were in Javascript or HTML, so while it was useful to gain an idea of what I was supposed to do, it did not help me actually authorize my keys. To solve the problem, I was able to find some documentation on a Python library called Spotipy which included instructions on how to authorize the client keys and conduct an album search.

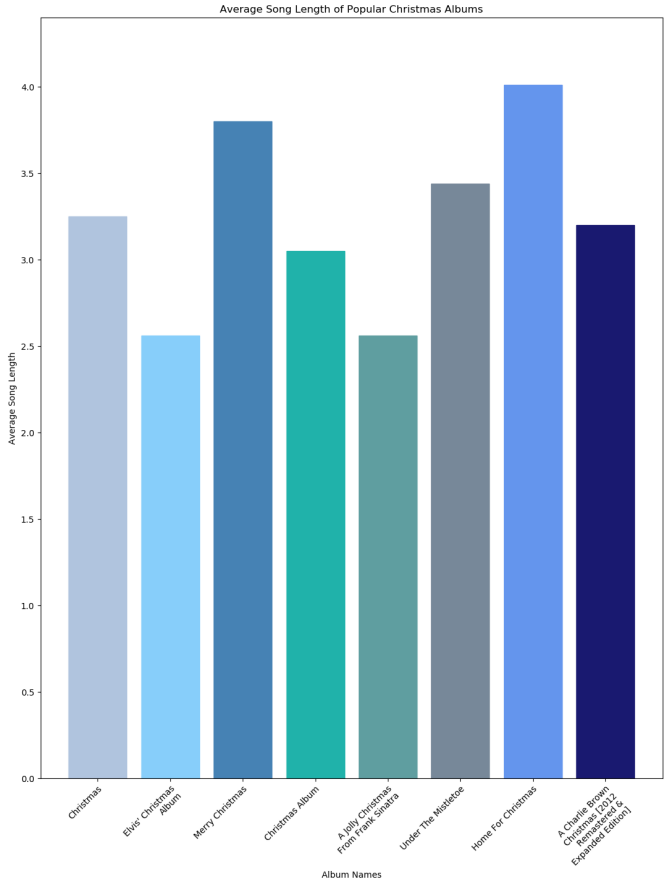
Report:

The report can be viewed in the file named avgLengths.csv and it includes two columns (one with the album names and the other with the average length of the tracks for each album).

	A	B
1	album names	average track length
2	Christmas	3.25
3	Elvis' Christmas Album	2.56
4	Merry Christmas	3.8
5	Christmas Album	3.05
6	A Jolly Christmas From Frank Sinatra	2.56
7	Under The Mistletoe	3.44
8	Home For Christmas	4.01
9	A Charlie Brown Christmas [2012 Rema	3.2

From the report, we can see that the album, “Home For Christmas”, by *NSYNC has the longest average track length of 4.01 minutes. The albums with the shortest average track length is “A Jolly Christmas From Frank Sinatra” by Frank Sinatra and “Elvis’ Christmas Album” by Elvis Presley—both albums have an average track length of 2.56 minutes.

From the visualization I created using Matplotlib, the viewer is able to easily see the albums’ average track lengths compared to each other. Again, the album, “Home For Christmas”, by *NSYNC has the longest average track length, while “A Jolly Christmas From Frank Sinatra” and “Elvis’ Christmas Album” are tied for the shortest average track length. Each album has a different color in order to differentiate them from each other which allows for an easier understood visualization.



Running the Code:

1. Download and save the file folder tianny_lu_final to your preferred directory and then open the folder.

2. Obtain your own Spotify Client Key and Client Secret Key. Open the python file spotify_info.py and fill in the the variables `SPOTIFY_CLIENT_ID` and `SPOTIFY_CLIENT_SECRET` with the appropriate strings.

3. On lines 129 and 150, please edit and change the directory path to match where the folder tianny_lu_final is on your device.

   - For example, this is the code on the both lines 129 and 150: `conn = sqlite3.connect('/Users/tiannyylu/Desktop/206_programs/final-project-tiannyylu/albums.sqlite')`

   - Please edit change where it says `"/Users/tiannyylu/Desktop/206_programs/final-project-tiannyylu/"` to the correct directory on your device.

4. In Terminal, navigate to the directory where the folder tianny_lu_final is located and run the Python program final_project.py by using the command `python final_project.py`.

Code Documentation:

In the file final_project.py, documentation can be found in the form of docstrings for each function. Overall, the code takes in a list of album IDs and calls the function `getSpotifyList` in order to iterate through the list of album IDs and return a list of dictionaries. Within the function `getSpotifyList`, the function `get_album` is also called. The function `get_album` takes in the parameter, `id`, and calls a search for album using the album ID number. After receiving the results from the API search, it creates a new dictionary that includes the album name and a list of dictionaries for each track on the album.

The function `addtoTable` takes in the parameters `spotifyList, conn, and cur`. The function checks to see if there is a table in the SQLite database called `Albums`, and if not, then it creates a table with the columns `album, artists_name, track, track_number, and length_of_track`. It will then iterate through the list of dictionaries, `spotifyList`, and adds each element to the appropriate column.

The function `getTrackLengths` takes in the parameters `cur` and `spotifyList`. Using `cur`, the function selects the `length_of_track` column for each album in `spotifyList`. Then, it converts the track length for every track in the album from milliseconds to minutes and adds the tracks length to a list, `minLst`. Using `minLst`, the function then finds

the average length of the tracks for each album and adds to the dictionary `avgDict` with the album name as the key and the average track length as the value. From there, the function uses the `avgDict` to write to a CSV file that includes the album name in one column and the average track length for each album in another column.

Lastly, the function drawBarChart takes in the parameter `avgDict` and uses the keys and values of the dictionary to create a bar chart. The bar chart compares the average track length of every album and each album is represented by a different bar color.

Resources:

Date	Issue Description	Location of Resource	Result
12/4/18 - 12/6/18	Knowing what is included in the objects of the various search options	https://developer.spotify.com/documentation/web-api/reference/object-model/	From the page, I was able to see what information was available from each search method and decide what I wanted to include in my program.
12/4/18 - 12/10/18	Understanding what information is included in the album object and how to pull the exact data I wanted	https://developer.spotify.com/documentation/web-api/reference/albums/get-album/	I found what objects I wanted to use and highlight in the program. In addition, I was able to see how the information was embedded and how I can pull the exact information I wanted.
12/4/18	Understanding how to authorize the client key and search for the album	https://spotipy.readthedocs.io/en/latest/	I was able to find an easy way to authorize my client information and an easy way to search for the albums using the .album( ) method.
12/3/18	Understanding how to receive a client key and how to authorize it	https://developer.spotify.com/documentation/general/guides/authorization-guide/	I was able to find how to create my own client key and tried to use their authorization flows in order to authorize my keys. However, the examples were in JavaScript and I decided to authorize the keys using Spotipy.

Date	Issue Description	Location of Resource	Result
12/8/18	How to select only a specific column given a conditional	https://www.w3schools.com/sql/sql_where.asp	I was able to only select the length_of_track column for every track in the list of albums I had searched.
12/9/18-/12/12/18	How to wrap the axis labels because they were running into each other	https://stackoverflow.com/questions/15740682/wrapping-long-y-labels-in-matplotlib-tight-layout-using-setp	This allowed me to understand how I can manipulate the x-axis labels by wrapping the words and make sure that the whole label is seen.
12/9/18-/12/12/18	How to wrap the axis labels because they were running into each other	https://matplotlib.org/gallery/text_labels_and_annotations/autowrap.html	Along with the link above, I was able to not only wrap the x-axis labels in order to ensure that they would not be cut off, but I was also able to angle the labels to ensure that they fit the figure size
12/10/18	The labels were being cut off because the size of the figure was not correct	https://stackoverflow.com/questions/17109608/change-figure-size-and-figure-format-in-matplotlib/17109830	I was able to increase the figure size to the correct portions so every label is easily seen and read.