Eric Andrews, Shanley Corvite
12/19/19
SI 206

Final Project Report

Link to GitHub Repository: https://github.com/ejandre/Final-Project

Goals for Project

For the purposes of this Final Project, our group, the "Music Machine," implemented the Spotify API and Genius Lyrics API in an effort to conduct an analysis of word trends within songs, and examine whether there are substantial differences in most-used words between the top 50 songs listened to by a user and the most-used words songs by performing artist Drake. The initial goals for the project were to compare the word-trends/frequencies between both members of the team and their respective top songs, and provide visualizations comparing the frequencies of each of their most-used words.

Goals that were Achieved

In the end, our group was able to succeed in many, but not all of these initial goals. For example, we were able to use the API's to retrieve the top 50 songs and the word frequencies for a user, however ended up comparing them to a more static artist, as opposed to another users information. Despite having to renavigate around some of our initial goals, the members of "Music Machine" are still proud of our final output, and the work that was put in to make it possible.

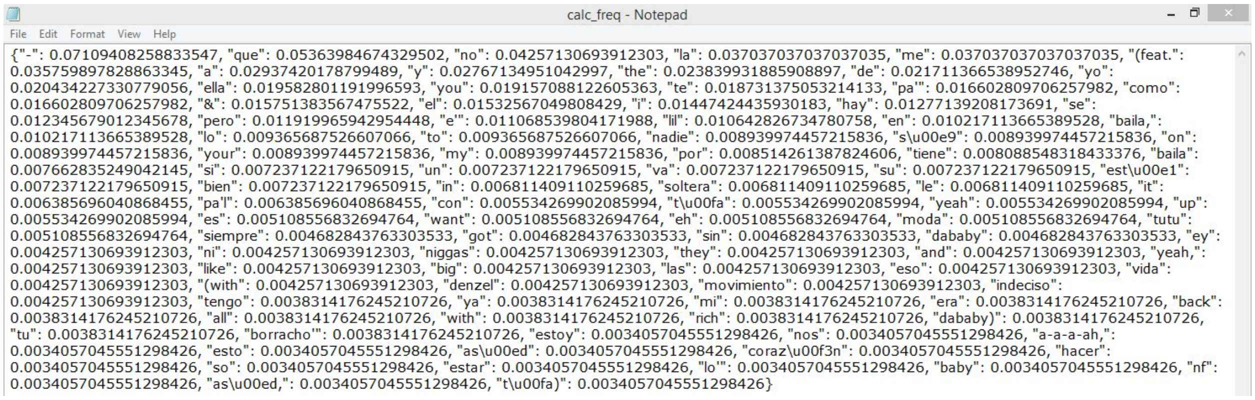
Problems that were Faced

Despite our contentment with our final product, this does not mean that our program did not experience more than its fair share of issues along the way. Our issues began with the retrieval of song lyrics via the Genius API. Although it was proving to be very successful originally, there were certain songs that the API was returning bizarre and inaccurate lyric transcriptions for. For example, in one member's Top 100 songs, the Genius API returned a section of a Kurt Vonnegut Novel as opposed to the lyrics of the song. As a result, we were forced to omit those songs to avoid misrepresenting the data.

Calculations from Data in the Database

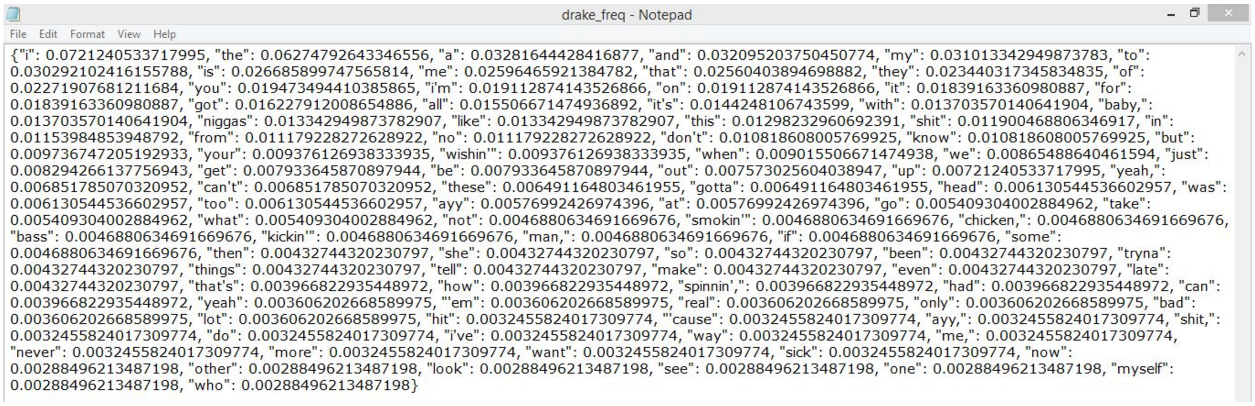
(Calculations explained in Documentation Pages)

Calc_freq.txt returns a dictionary with words as keys and their frequencies as values



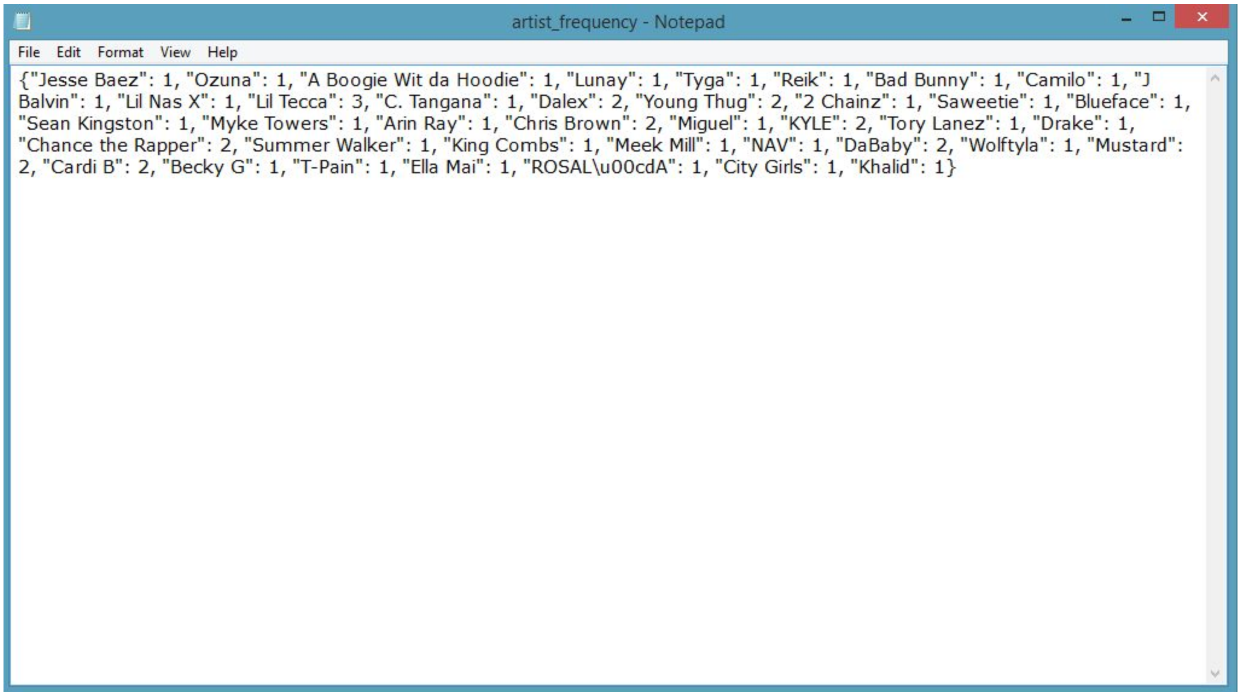
```
{"-": 0.07109408258833547, "que": 0.05363984674329502, "no": 0.04257130693912303, "la": 0.037037037037037035, "me": 0.037037037037037035, "(feat.":
0.035759897828863345, "a": 0.02937420178799489, "y": 0.02767134951042997, "the": 0.023839931885908897, "de": 0.021711366538952746, "yo":
0.020434227330779056, "ella": 0.019582801191996593, "you": 0.019157088122605363, "te": 0.018731375053214133, "pa'": 0.016602809706257982, "como":
0.016602809706257982, "&": 0.015751383567475522, "el": 0.01532567049808429, "i": 0.01447424435930183, "hay": 0.01277139208173691, "se":
0.012345679012345678, "pero": 0.011919965942954448, "e'": 0.011068539804171988, "lil": 0.010642826734780758, "en": 0.010217113665389528, "baila,":
0.010217113665389528, "lo": 0.009365687526607066, "to": 0.009365687526607066, "nadie": 0.008939974457215836, "s\u00e9": 0.008939974457215836, "on":
0.008939974457215836, "your": 0.008939974457215836, "my": 0.008939974457215836, "por": 0.008514261387824606, "tiene": 0.008088548318433376, "baila":
0.007662835249042145, "si": 0.007237122179650915, "un": 0.007237122179650915, "va": 0.007237122179650915, "su": 0.007237122179650915, "est\u00e1":
0.007237122179650915, "bien": 0.007237122179650915, "in": 0.006811409110259685, "soltera": 0.006811409110259685, "le": 0.006811409110259685, "it":
0.006385696040868455, "pa'l": 0.006385696040868455, "con": 0.005534269902085994, "t\u00fa": 0.005534269902085994, "yeah": 0.005534269902085994, "up":
0.005534269902085994, "es": 0.005108556832694764, "want": 0.005108556832694764, "eh": 0.005108556832694764, "moda": 0.005108556832694764, "tutu":
0.005108556832694764, "siempre": 0.004682843763303533, "got": 0.004682843763303533, "sin": 0.004682843763303533, "dababy": 0.004682843763303533, "ey":
0.004257130693912303, "ni": 0.004257130693912303, "niggas": 0.004257130693912303, "they": 0.004257130693912303, "and": 0.004257130693912303, "yeah,":
0.004257130693912303, "like": 0.004257130693912303, "big": 0.004257130693912303, "las": 0.004257130693912303, "eso": 0.004257130693912303, "vida":
0.004257130693912303, "(with": 0.004257130693912303, "denzel": 0.004257130693912303, "movimiento": 0.004257130693912303, "indeciso":
0.004257130693912303, "tengo": 0.0038314176245210726, "ya": 0.0038314176245210726, "mi": 0.0038314176245210726, "era": 0.0038314176245210726, "back":
0.0038314176245210726, "all": 0.0038314176245210726, "with": 0.0038314176245210726, "rich": 0.0038314176245210726, "dababy)": 0.0038314176245210726,
"tu": 0.0038314176245210726, "borracho'": 0.0038314176245210726, "estoy": 0.0034057045551298426, "nos": 0.0034057045551298426, "a-a-a-ah,":
0.0034057045551298426, "esto": 0.0034057045551298426, "as\u00ed": 0.0034057045551298426, "coraz\u00f3n": 0.0034057045551298426, "hacer":
0.0034057045551298426, "so": 0.0034057045551298426, "estar": 0.0034057045551298426, "lo'": 0.0034057045551298426, "baby": 0.0034057045551298426, "nf":
0.0034057045551298426, "as\u00ed,": 0.0034057045551298426, "t\u00fa)": 0.0034057045551298426}
```

drake_freq.txt returns a dictionary with words as keys and their frequencies as values



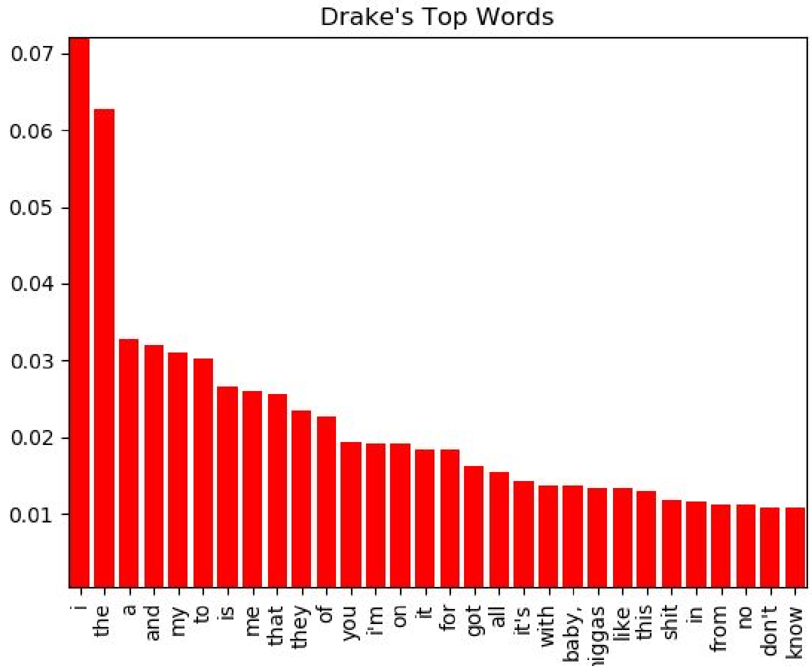
```
{"i": 0.0721240533717995, "the": 0.06274792643346556, "a": 0.03281644428416877, "and": 0.032095203750450774, "my": 0.031013342949873783, "to":
0.030292102416155788, "is": 0.026685899747565814, "me": 0.02596465921384782, "that": 0.02560403894698882, "they": 0.023440317345834835, "of":
0.02271907681211684, "you": 0.019473494410385865, "i'm": 0.019112874143526866, "on": 0.019112874143526866, "it": 0.01839163360980887, "for":
0.01839163360980887, "got": 0.016227912008654886, "all": 0.015506671474936892, "it's": 0.0144248106743599, "with": 0.013703570140641904, "baby,":
0.013703570140641904, "niggas": 0.013342949873782907, "like": 0.013342949873782907, "this": 0.01298232960692391, "shit": 0.011900468806346917, "in":
0.01153984853948792, "from": 0.011179228272628922, "no": 0.011179228272628922, "don't": 0.010818608005769925, "know": 0.010818608005769925, "but":
0.009736747205192933, "your": 0.009376126938333935, "wishin'": 0.009376126938333935, "when": 0.009015506671474938, "we": 0.00865488640461594, "just":
0.008294266137756943, "get": 0.007933645870897944, "be": 0.007933645870897944, "out": 0.007573025604038947, "up": 0.0072124053371799, "yeah,":
0.006851785070320952, "can't": 0.006851785070320952, "these": 0.006491164803461955, "gotta": 0.006491164803461955, "head": 0.006130544536602957, "was":
0.006130544536602957, "too": 0.006130544536602957, "ayy": 0.005769924269743960, "at": 0.00576992426974396, "go": 0.005409304002884962, "take":
0.005409304002884962, "what": 0.005409304002884962, "not": 0.004688063469166 9676, "smokin'": 0.0046880634691669676, "chicken,": 0.0046880634691669676,
"bass": 0.0046880634691669676, "kickin'": 0.0046880634691669676, "man,": 0.0046880634691669676, "if": 0.0046880634691669676, "some":
0.0046880634691669676, "then": 0.00432744320230797, "she": 0.00432744320230797, "so": 0.00432744320230797, "been": 0.00432744320230797, "tryna":
0.00432744320230797, "things": 0.00432744320230797, "tell": 0.00432744320230797, "make": 0.00432744320230797, "even": 0.00432744320230797, "late":
0.00432744320230797, "that's": 0.003966822935448972, "how": 0.003966822935448972, "spinnin',": 0.003966822935448972, "had": 0.003966822935448972, "can":
0.003966822935448972, "yeah": 0.003606202668589975, "'em": 0.003606202668589975, "real": 0.003606202668589975, "only": 0.003606202668589975, "bad":
0.003606202668589975, "lot": 0.003606202668589975, "hit": 0.0032455824017309774, "'cause": 0.0032455824017309774, "ayy,": 0.0032455824017309774, "shit,":
0.0032455824017309774, "do": 0.0032455824017309774, "i've": 0.0032455824017309774, "way": 0.0032455824017309774, "me,": 0.0032455824017309774,
"never": 0.0032455824017309774, "more": 0.0032455824017309774, "want": 0.0032455824017309774, "sick": 0.0032455824017309774, "now":
0.00288496213487198, "other": 0.00288496213487198, "look": 0.00288496213487198, "see": 0.00288496213487198, "one": 0.00288496213487198, "myself":
0.00288496213487198, "who": 0.00288496213487198}
```

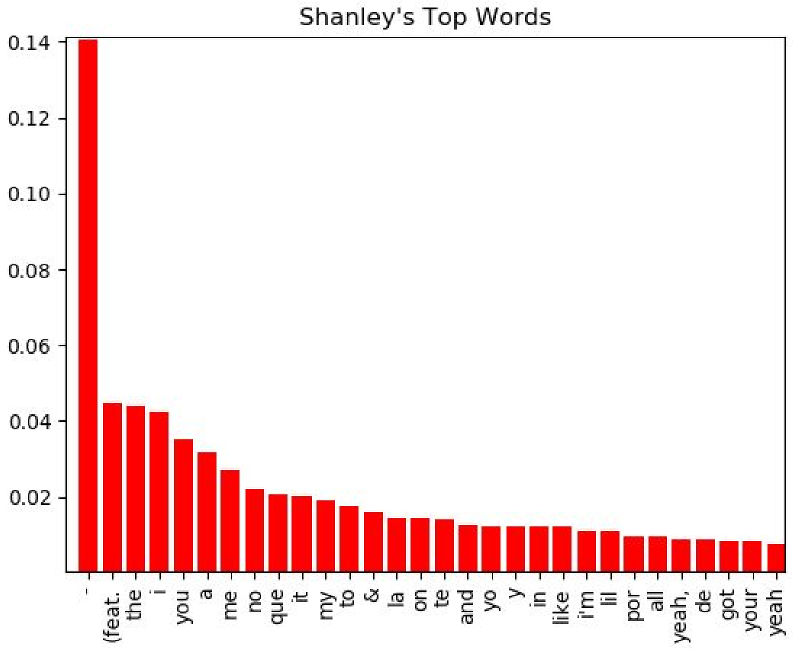
artist_frequency.txt returns a dictionary with artist as key and their number of occurrences as values



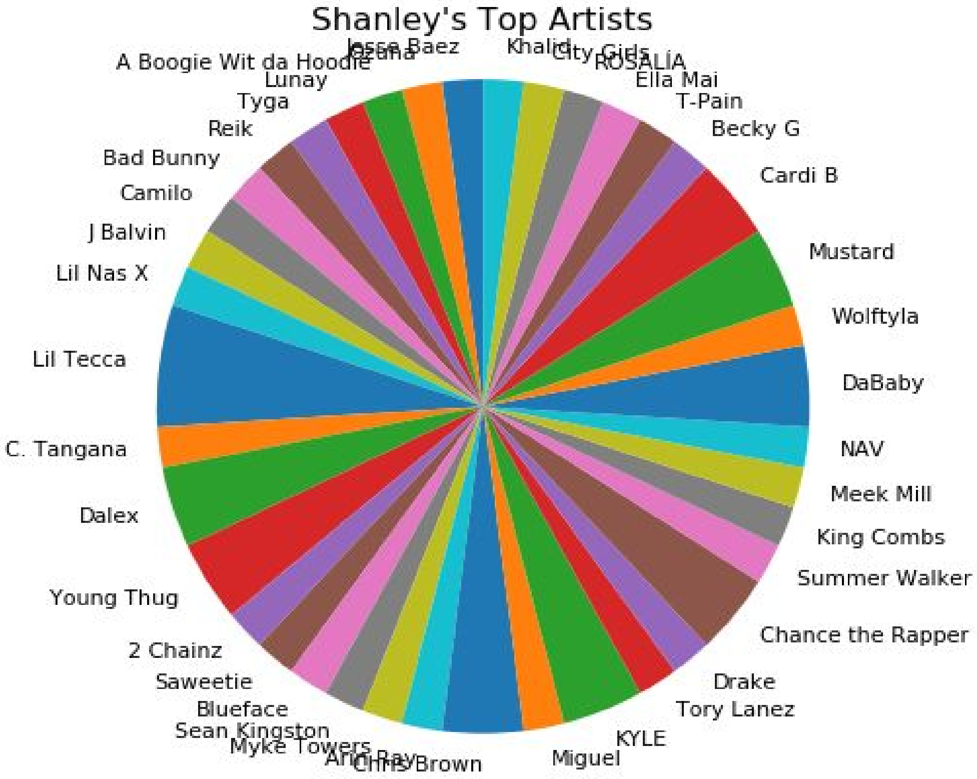
```
{"Jesse Baez": 1, "Ozuna": 1, "A Boogie Wit da Hoodie": 1, "Lunay": 1, "Tyga": 1, "Reik": 1, "Bad Bunny": 1, "Camilo": 1, "J
Balvin": 1, "Lil Nas X": 1, "Lil Tecca": 3, "C. Tangana": 1, "Dalex": 2, "Young Thug": 2, "2 Chainz": 1, "Saweetie": 1, "Blueface": 1,
"Sean Kingston": 1, "Myke Towers": 1, "Arin Ray": 1, "Chris Brown": 2, "Miguel": 1, "KYLE": 2, "Tory Lanez": 1, "Drake": 1,
"Chance the Rapper": 2, "Summer Walker": 1, "King Combs": 1, "Meek Mill": 1, "NAV": 1, "DaBaby": 2, "Wolftyla": 1, "Mustard":
2, "Cardi B": 2, "Becky G": 1, "T-Pain": 1, "Ella Mai": 1, "ROSAL\u00cdA": 1, "City Girls": 1, "Khalid": 1}
```

Visualizations





Shanley's Top Words



Shanley's Top Artists

Instructions for Running the Code

In order to successfully run the code, there are certain procedures that need to be executed first. Firstly, one must first ensure that they have the DB Browser for SQLite installed. After that, in order to use the program to retrieve their own Top 100 Spotify songs, one should apply for a Spotify Developer account, which will grant them a Client ID and Client Secret, two elements necessary for authorizing the Spotify API to access your account. In addition, add https://accounts.spotify.com/authorize to the Redirect URIs in your Spotify Developer account under Edit Settings. The same steps must be done for the Genius API without the authroization link, so one must apply for a developer account for that service as well. After all is said and done on that end, users of the program need to install a couple of new modules via their terminal window (pip install ____), which include spotipy, lyricsgenius, and matplotlib. After this is complete, the user will fill in their Client ID and Client Secret in the appropriate sections of the code (spaces where client_id and client_secret are requested), at which point the code is ready to run. At this point, the user will be redirected to a webpage where they are prompted to accept authorization for the program to access their Spotify account. If they accept and follow the prompts given in the terminal, the code will run as expected.

Documentation for Functions

Functions in Retrieval Script

- **grab_spotify_top_tracks(username,scope,client_id, client_secret,redirect_uri)**
  - This function takes the Spotify user's username, anticipated user data, Client ID and secret, as well as a redirect URI as input. Using the inserted data, the function will prompt for a user token to allow for the users data to be extracted. If the

token is accepted, a spotipy object is created using the Spotify API, and the users top tracks over a 6-month length of time are retrieved in JSON format. From there, the function iterates through the data and grabs each song name, artist name, and album that the song came from. These items are stored in tuples, which are then appended to a list. This list is returned by the function.

- **grab_drake_top_100(username,client_id, client_secret)**
  - This function is very similar to the aforementioned one, with a couple slight deviations. Taking the username, Client ID and Client Secret as input, the function creates a SpotifyClientCredentials object, which takes in the Client ID and Secret to approve authorization for the program. From there, a spotipy object is created using the Spotify API, and this object is used to retrieve 100 Drake songs from a playlist guaranteed to have all of his songs. These songs are returned in JSON format, at which point the function iterates over the data and once again grabs the song, artist, and album, storing it in a tuple and appending the tuple to a list.
- **get_song_lyrics(song,artist)**
  - This function begins by creating a lyricsgenius object using the Genius API and the Client ID provided upon making a Developer account. It then uses this object to specify a couple of parameters, including one to remove section headers (i.e. [Chorus], [Outro], [Interlude], etc.) and one to omit non-songs from the lyric retrieval. The function then uses the lyricsgenius object and the inputs (song and artist) to grab the lyrics to the passed song, then returning the lyrics.

- **write_lyrics_to_text(lyrics, filename)**
  - Using the lyrics to a song and the appropriate filename as input, this function begins by creating a file to append to. That way, it will write to it various times as the program iterates over different songs. The function then writes the lyrics to the songs to the text file, proceeding to close the file afterwards.
- **check_frequency_of_words(text)**
  - This function takes a text file as input, then proceeds to initialize an empty dictionary that will keep track of the amount of times a word appears in the text file. The function then opens the aforementioned text file, this time for reading purposes. Using the readlines() method to iterate over each line of the text file, the function turns every line to lowercase, then proceeds to iterate over each word in the line, and accumulating the number of times a word appears over the entirety of the text file. The function then sorts the data by items, creating tuples from the dictionary and using the values as a key, while also being sure to sort in reverse order to get the most-occurred words first. First 100 words are sliced and returned.
- **setUpDatabase(db_file)**
  - This function takes a string as input, and establishes a database connection and cursor, returning the cursor and connection.
- **add_user_tracks(tracks, cur, conn)**
  - This functions takes the tracks returned by grab_spotify_top_tracks, cursor, and connection as input. This function creates and connects to a database called

MainDatabase. In addition, a table is created, if it not already existing within the database, called UserTracks. It inserts the song title, artist name, and album name into the table. Using an iterator and a for loop, it adds 20 new songs to the table each time the code runs.

- **add_user_lyrics(x, cur, conn)**
  - This function takes the lyrics returned by check_frequency_of_words, cursor, and connection as input. This function connects to the database, MainDatabase, and creates a new table called UserLyrics if it does not already exist. It inserts the word and frequency for each word in the text file, lyrics.txt.
- **add_drake_tracks(tracks, cur, conn)**
  - This function takes the tracks returned by grab_drake_top_100, cursor, and connection as input. This function connects to the database, MainDatabase, and creates a new table called DrakeTracks if it does not already exist. It inserts Drake's song name and name into the table. Using an iterator and a for loop, it adds 20 new songs to the table each time the code runs.
- **add_drake_lyrics(x, cur, conn)**
  - This function takes the lyrics returned by check_frequency_of_words, cursor, and connection as input. This function connects to the database, MainDatabase, and creates a new table called DrakeLyrics if it does not already exist. It inserts the word and frequency for each word in the text file, lyrics2.txt.
- **main()**

- This function is where the actual execution of the project occurs. It begins by first storing the return of spotify top tracks and drake top tracks in their own respective variables, person 1 and drake_compare. The function then iterates over both of the tracks, fetching the lyrics to the songs and continuing if they cannot be found. The function then writes these fetched lyrics to a text file, and proceeds to execute the check_frequency_of_words function on these. The main function then proceeds to establish a database, 'MainDatabase.db', grabbing the user top tracks, adding them to a table of song and artist, then making another table with words in the lyrics, and the amount of times they appear. It then does the same with the retrieved Drake tracks

Functions in Calculations Script

- **set_connection(db_file)**
  - This function takes a string as input, establishing a connection with a database and returning the connection to the requested database.
- **get_freq(conn)**
  - This function takes the connection object as input. It selects the sum of all the words in the database and stores it in a variable 'sum_word', and also selects the individual words and their respective number of occurrences, storing it in the variable 'data_words'. The data from data_words is then fetched, iterated over, and stored in the list freq_list. Using this list, each number of occurrences is iterated over and the frequency of each word is calculated by dividing the number

of occurrences by the total sum. A dictionary is then created, using the word as a key and it's percentage frequency as a value. This dictionary is the return value.

- **get_drake_freq(conn)**
  - This function does the exact same as the above function, however accesses a different table, returning a dictionary specified for Drake lyrics.
- **write_drake_to_text(data)**
  - This functions takes a dictionary as input. It opens a text file, "drake_freq.txt", set for writing, and uses json to dump the dictionary in the text file.
- **write_calc_to_text(data)**
  - This functions takes a dictionary as input. It opens a text file, "calc_freq.txt", set for writing, and uses json to dump the dictionary in the text file.
- **artist_frequency(conn)**
  - This function takes the connection as input. It creates a new dictionary called artist_dict. It selects all the artists from the UserTracks table from the MainDatabase. It goes through each artist and adds it to the dictionary, keeping track of how many times that artist is repeated from the table.
- **artist_text_file(artists)**
  - This function takes a dictionary as input. It opens a text file titled, "artist_frequency.txt", set for writing, and uses json to dump the dictionary in the text file.
- **pie_chart_top_artists(data)**

  - This function takes a text file as input. Json is used to access the text file and grab the dictionary that was stored within, which has the top artists and the amount of times they occur on the top tracks. From there two variables are created, one which stores all the keys, and one that stores the values. Matplotlib is then used to create a pie chart titled “Shanley’s Top Artists”, which uses the two variables made before as the categories and their respective quantities. This is then saved as a .png file.

- **bar_chart_word(data,title)**
  - This function takes a string and text file as input, using json.loads to read the data and return the dictionary inside, which has the words and their respective frequencies. From there two variables are created, one which stores all the keys, and one that stores the values. Matplotlib is then used to create a bar chart, using the input string as a title. The bar chart uses the keys as labels, and values as the sizes of the bars. After rotating labels by 90 degrees and changing color of bars, the chart is then saved as a .png file.

Documentation of Resources

Date	**Issue Description**	**Location of Resource**	**Result (did it solve the issue?)**
12/03/19	Explains how to integrate the Spotipy API with Genuis Lyrics API	https://dev.to/willamesoares/how-to-integrate-spotify-and-genius-api-to-easily-crawl-song-lyrics-with-python-4o62	Yes. We were able to use both the Spotipy API and Genius Lyrics API

12/03/19	Explains how to get authorization from Spotify so user can use the Spotipy API	https://developer.spotify.com/documentation/general/guides/authorization-guide/	Yes. We were able to write code that would give the user access to Spotify with the Spotipy API.
12/03-12/18/19	Explains errors that were unfamiliar to us	https://stackoverflow.com	Yes. It helped us fix errors that we were running into.
12/05/19	Documentation for the Spotipy API	https://spotipy.readthedocs.io/en/latest/	Helped us in figuring out how to use the different methods for the Spotipy module under the Spotify API
12/5/19	Documentation for the lyricsgenius API	https://github.com/johnwmillr/LyricsGenius	Helped in figuring out different methods and parameters for the lyricsgenius module under the Genius API
12/10/19	Explains how to put contexts of a text file into a database	https://stackoverflow.com/questions/57903952/sqlite-python-read-records-into-table-from-txt-file	No. The Databases are still not being created.
12/10/19	Explains how to add a list of tuples into a database	https://stackoverflow.com/questions/50519558/insert-list-of-tuples-sqlite-python-error	Yes. The database was created with multiple tables made from lists of tuples.