WTF Presents:

SI 206 Final Project Report

By: Tamariah Davis and Nylah Omar
GitHub Link: https://github.com/tamarida/final206

The goals of our project

The goal of our project was to figure out if the average amount of profanity an artist uses in their music affects their popularity level. As a group, we believed that more profanity increased popularity and wanted to test this theory with R&B music using Spotify, Genius, and Genius Lyrics. With this goal we can also determine which R&B artist used the most profanity in their lyrics OR the correlation between profanity ("bad word") and popularity in R&B music.

Why did we choose this project topic?

We chose this project topic because we both enjoy R&B music and wanted to embrace something from our culture. Rhythm and Blues, frequently abbreviated as R&B or R'n'B, is a genre of popular music that originated in African-American communities in the 1940s.
(talk about how r&b music is in our culture)

The goals that were achieved

- Using the Spotify API and a list artist that our team created we found the top 10 tracks and the popularity level per song.
- Using the Genius API we searched for 10 songs per artist and retrieved the path URL. *(the path URL is the ending of a 'genius.com/songs/892719' that locates the song so we can retrieved the lyrics)*
- Using Beautiful Soup we were able to web scrape the lyrics for each song found in our Genius API search.
- Create database with information about artists, song tracks, and lyrics
- Use database to calculate averages with the data
- Create visualizations from calculated data
- And most of all we're more confident coders!

The problems that you faced

During this project there was many "hiccups" or "WTF's" we would like to call it:

- Finding an idea
  - We had so many ideas, we couldn't choose one
- Linking the API's
  - We had initial trouble with linking the API correctly. We had to get verification with the Genius APi and Spotify API.

    - Using the web we were able to find a resource that did something similar and use their code for reference.
- Retrieving the URL pathway for each song.
    - After retrieving the song from Spotify we went into Genius to search for our song and retrieve the **path url** “genius.com/**song/245633**”. This was a challenge because we needed the song title and artist name to match perfectly on both Spotify and Genius platforms. These are two different platforms which means that the titles did not always align.
    - Also the song we were looking for wasn’t always in the first few search results on the genius platform. The first research was not always the song we were looking for so creating a solution that said look through all the songs until you get to the top songs. (there was an issue with our indexing)
    - To get around this issue we looped through all the results and until we found the correct song with the correct song title. Some songs from Spotify couldn’t be found on the Genius platform, so we added a few more artists to make sure we had at least 100 items in the tables.
- Grabbing all the Lyrics from Genius Website.
    - All lyrics weren’t in one container
    - When we were getting the word count back for our song lyrics , the program would give us back the amount of lyrics being 0. We thought this was because maybe there were breaks when we went into inspect element and we created a separate for loop to try and grab those lyrics but this would still give an inaccurate word count.
- Counting the “Bad Words”
    - We had trouble when our function was not properly counting each curse word. We noticed when we picked a random song and went through and counted each curse word and our function told us there were two and we counted six.

Files and Functions Breakdown

functions.py

Our functions.py uses the Spotify API and a list artist that our team created we found the top 10 tracks and the popularity level per song. Our function also uses the Genius API to search for 10 songs per artist and retrieve the path URL. And finally uses Beautiful Soup to web scrape the lyrics for each song found in our Genius API search.

Functions
get_artistID - This function get_artistID takes in a list of artist names, searches the Spotify API for that artist, and returns a list if tuples of the (Artist Name, Artist ID)

get_topTracks - The function get_topTracks takes in a list of tuples that contain (Artist Name, Artist ID), uses the top tracks Spotify API and the Artist ID to return a list of tuples that have the artist name, artist ID, song title, song popularity with 10 songs (their top songs) per artist

Ex. (Artist ID, Artist Name, Song Title (1), Song Popularity)
(Artist ID, Artist Name, Song Title (2), Song Popularity)
etc...

get_songURLpath - The function get_songURLpath takes in a list of tuples (from get_topTracks) and searches the artist name and the song title using the Genius API finds the path url for each song (the pathurl is a string ex. '/songs/234513/' that is used at the end of the genius.com url to find the song lyrics) and returns a list of tuples that contains the Artist ID from Spotify, Song Title from Genius, Artist Name from Genius, and the Song URL Path

get_profanity - The function get_profanity takes in a list of tuples (from get_songURLpath) uses BeautifulSoup and the url path from the genius data to find the lyrics and return the lyrics to each song in a string.

calculations.py

The file that contains the data in the database is calculations.py. In this file, we calculate the average song popularity per artist, and average bad word usage per artist. We calculated these averages by using the database.

Functions

get_pop - takes in a query that selects the popularity and artist id columns from the track table in the final project database and returns a list of tuples (artist id, song popularity)

get_artist - takes in a query that selects the name of the artist and the artist id columns from the artist table in the final project database and returns a list of tuples (artist id, artist name)

get_lyrics - takes in a query that selects the artist id and lyrics columns from the lyrics table in the final project database and returns a list of tuples (artist id, lyrics)

get_avgpop - takes in a two list of tuples one that contains (artist id, song popularity) and the other that contains (artist id, artist name) and returns a dictionary of average popularity of songs for each artist

get_badWords - takes in a list of tuples that contain (artist if, lyrics) , a list of tuples that contain (artist id, artist name), and a list of profanity words. This function returns a dictionary of artist name and a list of bad word total per song in a key value pair.

get_avgbad - takes in a dictionary that contains artist name and list of bad word count per song in a key value pair and returns the average of bad word usage per artist as a tuple.

databases.py

The databases.py file creates, reads, and stores the data we collected. In this file, we made sure not to repeat data from table to table.

Functions
open_database - creates and opens database

createArtistTable - takes in a list of artists and creates a table called lyrics with artist name as a column and creates an artist id for each artist in another column.

createTrackTable - takes in a list of tuples containing song title, artist name, and song popularity and creates a table

createLyricTable - takes in a list of tuples containing song title, artist name, lyrics and creates a table

visualization.py

This file contains and creates the visualizations we created in this final project.

Functions
plot_avg - takes in a list of tuples that contains average song popularity per artist and creates a bar graph
plot_avgbad - takes in a list of tuples that contains average bad words usage per artist and creates a bar graph
plot_correlation - takes in a list of tuples that contains average bad words usage per artist and a list of tuples that contains average song popularity per artist and creates a scatter plot.

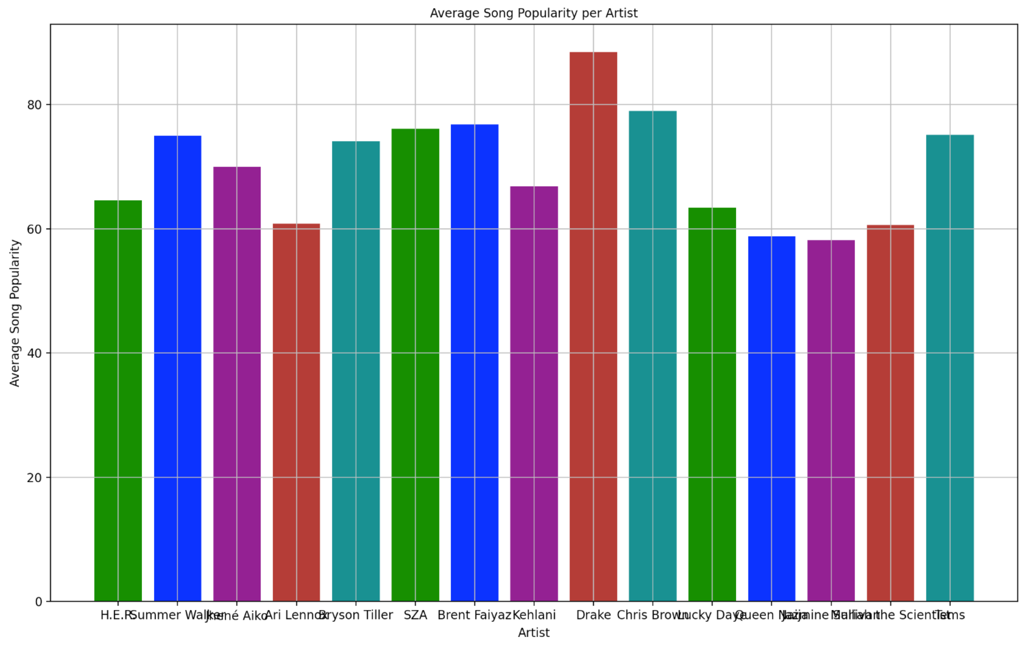
secrets.py

This file contains access keys and access tokens from the Spotify and Genius API, so it's easy for us to change access codes when working in a group.

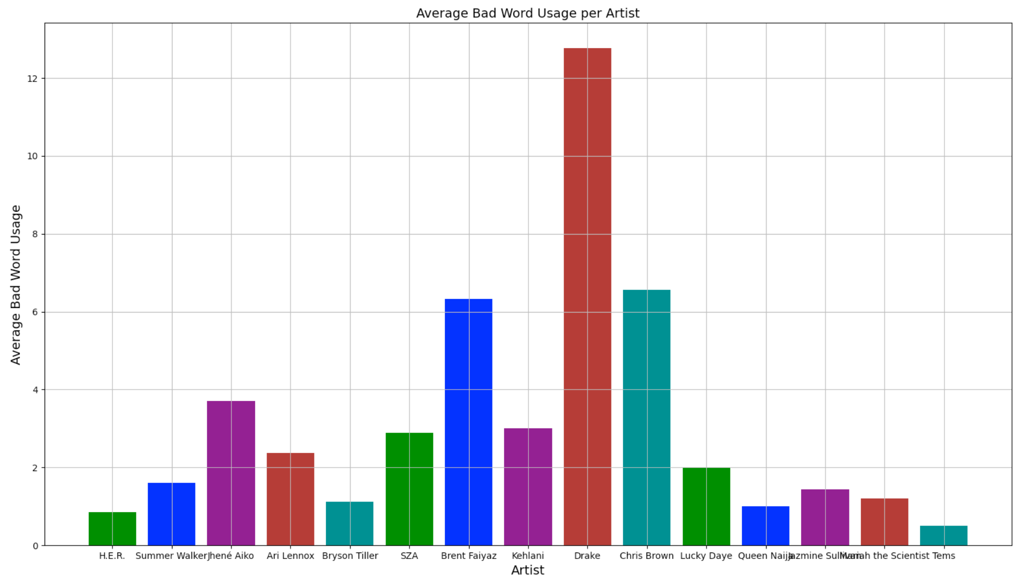
FinalProject.db

This is our database that stores our three tables: artist, tracks, and lyrics. We use these tables essentially to calculate our findings and create visualizations.

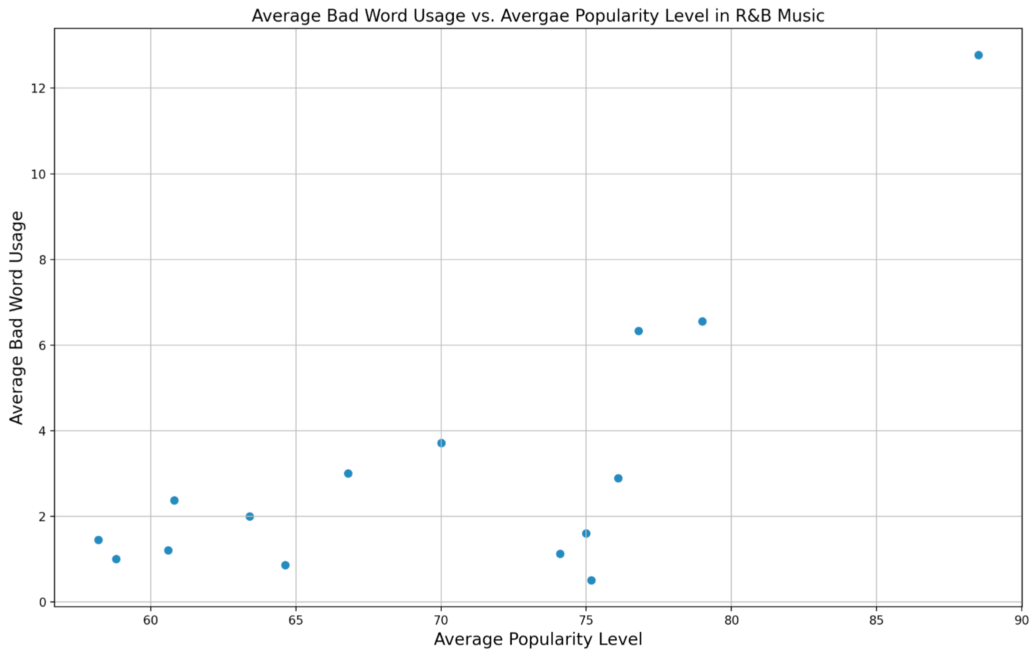
Visualizations



This visualization shows the average song popularity per artist. Popularity levels are similar across the board. With Jazmin Sullivan having the lowest average song popularity and Drake having the highest song popularity level.



This visualization shows the average number of bad words an artist uses in a song. As you can see the artist with the most bad word usage is Drake .



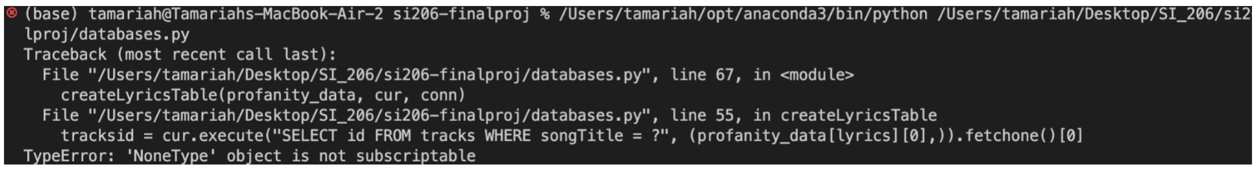
This visualization shows the correlation between bad word usage and popularity level using R&B music.

Instructions for running our code

Step 1 - Run secrets.py file
Step 2 - Run the functions.py file

➔ This will run through both Spotify and Genius API then use Beautiful Soup to pull from Genius.com. Waring: This may take like 2-3 minutes

Step 3 - Run the databases.py file (5 times)



```
⊗ (base) tamariah@Tamariahs-MacBook-Air-2 si206-finalproj % /Users/tamariah/opt/anaconda3/bin/python /Users/tamariah/Desktop/SI_206/si2
  lproj/databases.py
  Traceback (most recent call last):
    File "/Users/tamariah/Desktop/SI_206/si206-finalproj/databases.py", line 67, in <module>
      createLyricsTable(profanity_data, cur, conn)
    File "/Users/tamariah/Desktop/SI_206/si206-finalproj/databases.py", line 55, in createLyricsTable
      tracksid = cur.execute("SELECT id FROM tracks WHERE songTitle = ?", (profanity_data[lyrics][0],)).fetchone()[0]
  TypeError: 'NoneType' object is not subscriptable
```

➔ The first run you will receive an error. **Please ignore this**, the error will pop up because one database needs to run at least once before the other database can be created fully since we are limiting it to 25 items per run.
➔ Remember to run this file 5 times!
➔ Comment out line 12 (this will help the calculations.py and visualizations.py run faster)

Step 4 - Run the calculations.py file
Step 5 - Run the visualizations.py file

Resources

- https://prettystatic.com/automate-the-spotify-api-with-python/
- https://developer.spotify.com/dashboard/applications

- https://developer.spotify.com/documentation/web-api/quick-start/
- https://developer.spotify.com/console/get-artist-top-tracks/
- https://developer.spotify.com/console/get-search-item/
- https://developer.spotify.com/documentation/general/guides/authorization/client-credentials/