SI206 Final Report

Manav Khanvilkar and Joseph Schaubroeck

GitHub Repository

https://github.com/kaavlu/SI206-LyricMatch

Original Goals

Our initial plan involved gathering information about U.S. treasuries, such as date ranges and average interest rates. Additionally, we were planning to collect data for popular ETFs and stocks, including their average prices during specific date ranges. Our original plan was to use the stock price data for different time periods to calculate average stock rates and compare them with the average interest rates on U.S. treasuries at those times. Although this was a cool idea, we ran into problems regarding actually accessing these APIs. Many of these APIs weren't free and required a premium developer subscription which we didn't want to invest in. This caused us to pivot to a new idea in which we analyze music. We are currently using BeautifulSoup to scrape the Billboard Hot 100 songs, and utilizing a lyrics API to fetch lyrics for each song on the list. The correlation we look for now is whether there is any relationship between the number of words in a song and its rank on the Billboard.

Goals Achieved

After our initial failure with our stock market APIs, we decided to pull data from the MusicXMatch developer API(https://developer.musixmatch.com/) and we used BeautifulSoup to scrape the Billboard Hot 100(https://www.billboard.com/charts/hot-100/). With this information we posed 2 specific questions. 1. Is there any correlation between the number of words in a song and its corresponding rank on the Billboard Hot 100 list, and 2. What are the distributions between the number of songs and the lyric word count. As seen in our tables, we were able to achieve both our goals. We used matplotlib and implemented a bar graph to answer question 1. Doing this we saw that there were no significant trends among the rank of a song and its word count. To answer question 2 we created a pie chart which showed that over 60 percent of the songs on the chart had word counts between 251-500 words. We also noticed that the smallest distribution of songs were ones that had a word count of 751+.

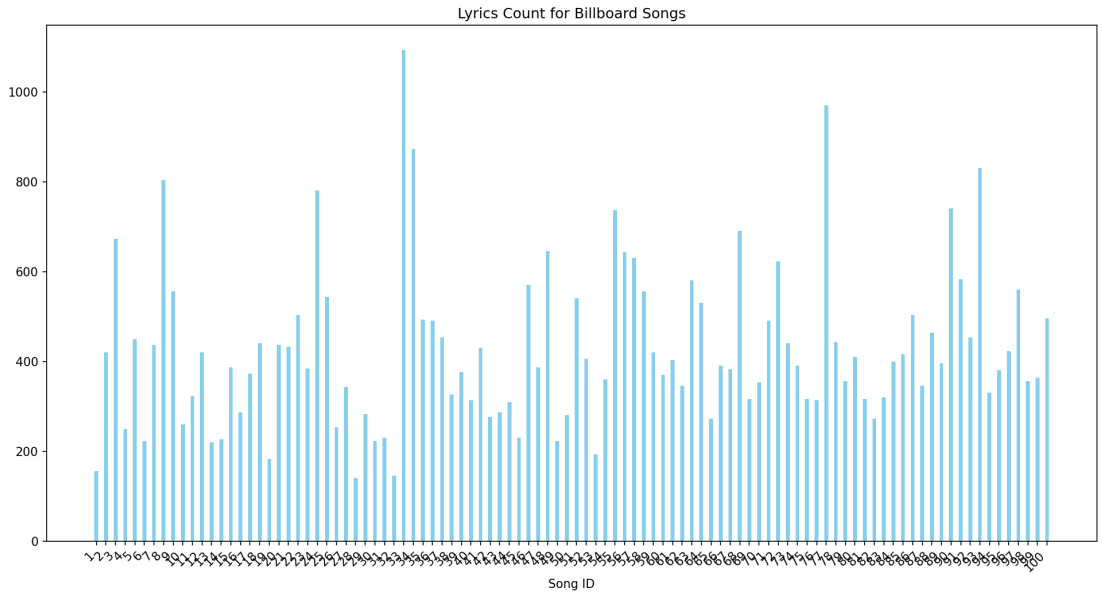
Problems

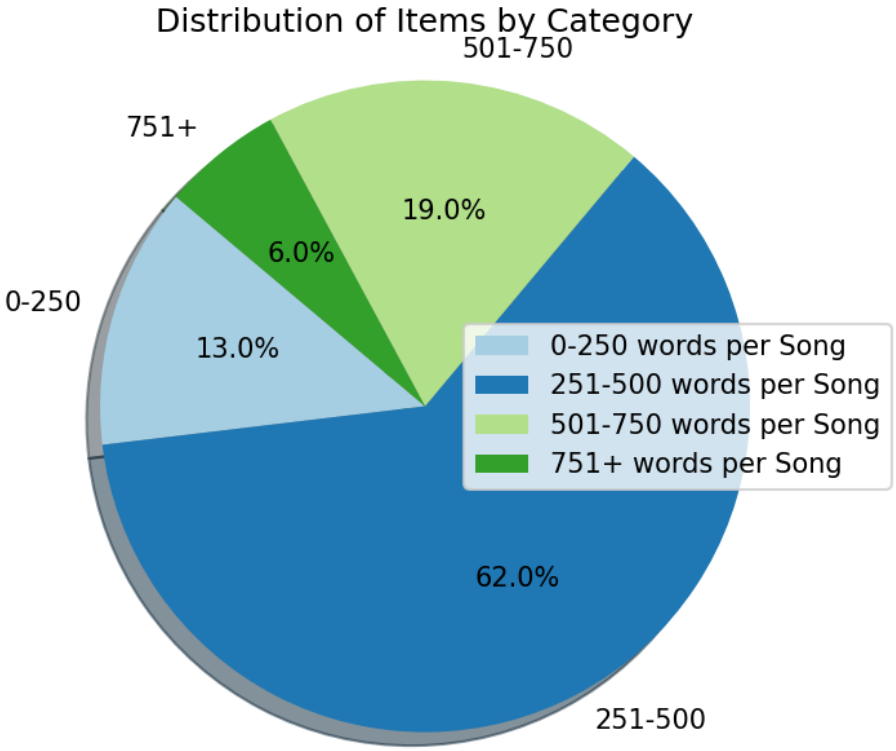
Our most prominent problem was our initial choice of APIs. We weren't able to access the information from our stock APIs for free which led us away from our first choice of a project. We then searched through the github API page and found many different APIs yet we had a hard time choosing one that might form meaningful correlations. After a lengthy search we stumbled upon a lyric API which led us to think of different ideas for a project. We then decided to use BeautifulSoup instead of an API to get song information from a website which solved our correlation problem. After that our only problems came from the code structure itself. At first we were having a hard time deciding how to design our interface but we came up with meaningful abstractions which allowed us to simplify the code.

Calculations

1. Our first calculation was done in our get_lyrics.py file where we had to overcome some data issues. Our lyrics API only returns 30 percent of the lyrics on a free plan so we had to do some calculations to smooth out our data. We accounted for this by dividing our lyric length by 0.3. If the lyrics don’t exist (artist name is spelled wrong or track title is spelled wrong), we input the average lyric count for that sound to prevent our data from being skewed.
2. Our second calculation was done in our main.py file where we had to calculate the distributions for our pie chart. We split the data up into sections of 250 (except the last section which accounts for 750+ words). We then incremented a list full of counters whenever a song fell into a certain category to be able to calculate percentages. Once we finished that we were able to graph our results in a pie chart with accurate data splits.

Visualization





Instructions

1. Run get_billboard.py 4 times in order to scrape 25 instances of song data each time, totalling 100 instances of song data in Billboard table in music.db
2. Run get_lyrics.py 4 times in order to use API to get lyrics for 25 songs each time, totalling 100 instances of lyric count data in Lyrics table in music.db
3. Run main.py once in order to join data, create visualizations, and write output to a file

Function Documentation

get_billboard.py:

set_up_database():

description: Establishes connection with database

input: None

output: cur and conn for database

create_table(cur, conn):

description: Creates Billboard table in database if it does not yet exist

input: cur, conn

output: None

insert_songs(cur, conn):

description: Gets 25 songs from Billboard Hot 100 website, and inserts the song data into the database.

input: cur, conn

output: None

get_billboard_info(num_rows):

description: Gets next 25 songs from Billboard Hot 100 website using BeautifulSoup, and uses the number of rows currently in the database to know which songs to grab.

input: number of rows currently in Billboard table

output: list of tuples including (song name, artist)

get_lyrics.py:

set_up_database():

description: Establishes connection with database

input: None

output: cur and conn for database

create_table(cur, conn):

description: Creates Lyrics table in database if it does not yet exist

input: cur, conn

output: None

generate_lyrics(artist, track):

description: Uses API to get lyrics of song as a string based on artist name and song name

input: artist name and song name

output: lyrics as a string

insert_lyrics(cur, conn):

description: Gets the next 25 songs, generates their lyrics, inserts lyric data into Lyrics table in database

input: cur, conn

output: None

get_lyrics(songs):

description: Counts the number of words per song based on lyrics

input: list of songs

output: list of word counts corresponding to each song

main.py:

set_up_database():

description: Establishes connection with database

input: None

output: cur and conn for database

plot_lyrics_count(cur, conn):

description: Performs database join to get number of words in each song in the lyrics table combined with the rank of each song from the Billboard table. Then, gets rank and lyric count data as lists and plots info as a bar graph with matplotlib.

input: cur, conn

output: None

plot_pie_chart(cur, conn):

description: Performs database join to get number of words in each song in the lyrics table combined with the rank of each song from the Billboard table. Categorizes data into different word count ranges and turns those calculations into a pie chart using matplotlib.

input: cur, conn

output: None

data_to_text(cur, conn):

description: Outputs findings to text file.

input: cur, conn

output: None

Resource Documentation

Date	Issue Description	Location of Resource	Result
12/1/23	Our old APIs weren't working well for us and we needed new ideas and new resources.	Github page of open APIs	We were able to locate a lyrics API that would work well with our new project idea.
12/3/23	The lyric API we found seemed to be a good resource from what we could tell but when implemented we saw that the API had been taken down.	Google Searches	We were able to locate our new API (musicxmatch) and this actually pulled the lyric data.
12/5/23	We needed to find a music API that provided us with artist names, track titles, and 100 queries but we couldn't locate a resource like this that was free.	Canvas	We switched over to BeautifulSoup and used this to scrape the Billboard Hot 100 for 100 queries, artist names, and track titles. This also provided us with ids.
12/7/23	All our code was in one file and we needed to figure out a better way to abstract our code.	Brainstor ming	By discussing how to structure our project, we were able to come up with our current structure. We have a py file for our lyrics subtable, a separate file for our billboards table, and a driver file that does all our plotting and calculations.
12/10/23	Matplotlib was graphing our bar chart strangely with all the bars squished together due to our 100 values and our graph was looking inaccurate.	Matplotlib website	By reading some documentation, we were able to figure out bar spacing, horizontal width, and other aspects that allowed us to graph our data properly.