Final Project Report

Spotify Junkies: Brian Cho, Will Clancy, Matthew Yeh

SI 206

Prof. Barbara Ericson Section 101

December 12, 2022

GitHub Repository Link

https://github.com/machupichu412/SI-206-Final-Project

Goals for Project

Our goals for this project are to pull in data from Spotify, Tastedive, and Celebrity Ninjas to show a correlation between the popularity ranking of each artist based on genre, and the artists' net worth. Through this we plan on calculating figures such as the average net worth of artists based on genre, the correlation between artists' age and their net worth, and much more. With these calculations, we'll plot various types of graphs such as bar charts, scatter plots, and etc. to visualize our data and calculations. We hope to learn more about the overall process of scraping data from APIs, as well as learn about the different use cases that are possible with data scraping for any future projects of ours, whether that be in a personal or professional context.

Achieved Goals During Project

We achieved all of the goals we had for this project, and achieved goals we originally didn't have in mind as well. As for our original goals, we were able to successfully pull in data from Spotify, Tastedive, and Celebrity Ninjas APIs to gather relevant data points we could use for the calculations and visualizations. We also successfully used SQL Query statements to make calculations from the database figures and pull in selected data for our visualizations. By using matplotlib, we were able to create visualizations for our calculated figures, and learned when it makes most sense to use a certain type of chart versus another (depending on the data you're calculating and what the relationship is between the data you want to display). Finally, we most importantly learned more about how we could use the data scraping process and apply our learnings to any future projects we may work on that'll require the data gathering, calculating, and visualization skills we gained during this project.

As for goals we originally didn't have in mind, one skill we gained in this process was learning how to import and use other user-made modules. In our case, we used Spotipy which is a Python library for the Spotify Web API that allows full access to all music data provided by Spotify. Another skill that we never intended on improving heading into the project was just learning how to collaborate on GitHub, especially in an extremely collaborative environment such as this one where multiple people are working on separate files at the same time and

making commits at different times. This whole process helped us to appreciate the value of planning how to modularly separate the code by files and the order in which we write code for the files. Through this, we understood how much easier it is to interpret and process the code when splitting it up into smaller pieces/functions.

Problems Faced During Project

There were inherently a lot of challenges that arose during this project, specifically in the API selection process as well as during the data collection process. Before we began our project, our plan was to use the Spotify and Tastedive API in conjunction with the US Census Bureau API to examine if income level relates to the average time spent listening to the most popular music. After assessing the data points from the US Census Bureau API, we quickly realized that there were no common data points we could use between Spotify/Tastedive and the US Census Bureau that would allow us to make such calculations, so we had to pivot our goals for the project.

We then decided to use celebrity data along with data from Spotify and Tastedive, so we first used the Celebrity API from API Ninjas to pull in data from celebrities such as their age, net worth, and etc. However, the data points given to us by API Ninjas were limited in the sense that not all celebrities had the same amount of data points, and there simply wasn't enough useful data to make any meaningful calculations. Therefore, we ended up using the Celebrity Ninjas API as an alternative given there was much more information provided consistently throughout a majority of celebrities.

By far the biggest challenges we faced were during the data collection process. First, given that we were comparing celebrities' net worth and age with their spotify data, we had to filter out any artists that were considered a band as we wouldn't be able to get individual net worth and age data for these artists. We solved this issue by utilizing another API that contained artist information called Music Story. We used this API to check if each artist we were adding to the database was an individual instead of a band. The second issue was that even though we figured out a way to only search for individual artists, there were still some instances where a band (BTS in our case) was considered as a single artist. This was because Music Story returned a list of artists instead of only one, so the Music Story request returned the value for an individual who had BTS in their name. We fixed this by looping through the artists returned by

Music Story and only retrieving information if the returned artist name was exactly equal to the input artist name. Additionally, for artists with special characters or accent marks in their name (Beyoncė for example), the database wouldn't properly read in the artist's name so we had to figure out how to convert the artists' names into a standard format without any special characters by using the Unidecode Python package. You may need to install Unidecode in order for the code to run properly.

Our last challenge was limiting the data retrieved from each API to 25 at a time. This was especially difficult with the Celebrity Ninjas API, as we were running it in a separate file and it would update every single artist in the database, which meant that if there were more than 25 artists, it would run over 25 times. We solved this by integrating the Celebrity Ninjas API into the Spotify API and Tastedive API code. The updated code calls the Celebrity Ninjas API for each new artist added to the database. Since our requests to those APIs were already limited to 25 or fewer, the Celebrity Ninjas API would only run 25 or fewer times each time the code is run.

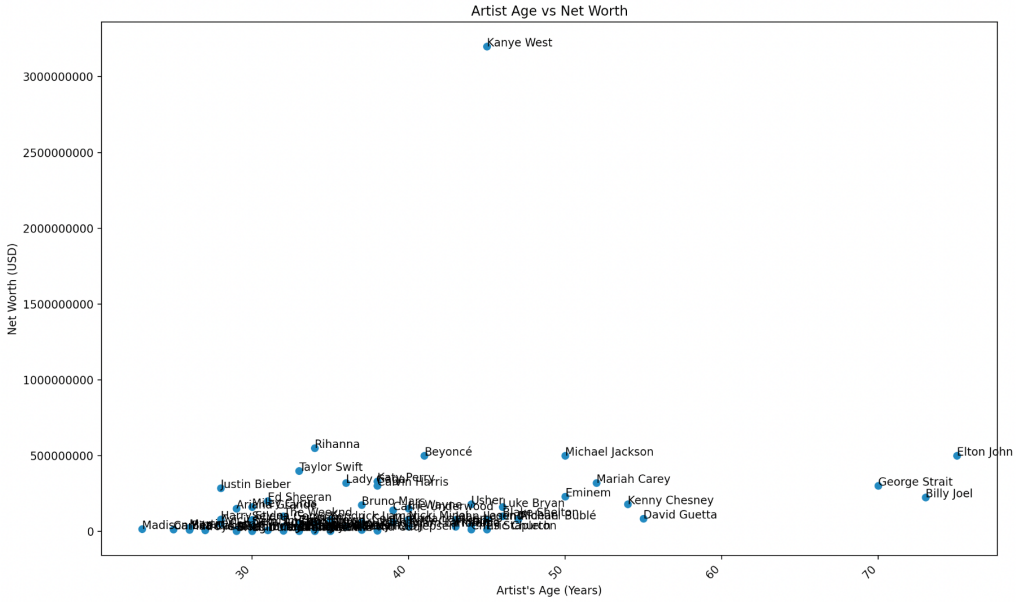
Calculations from Data in Database

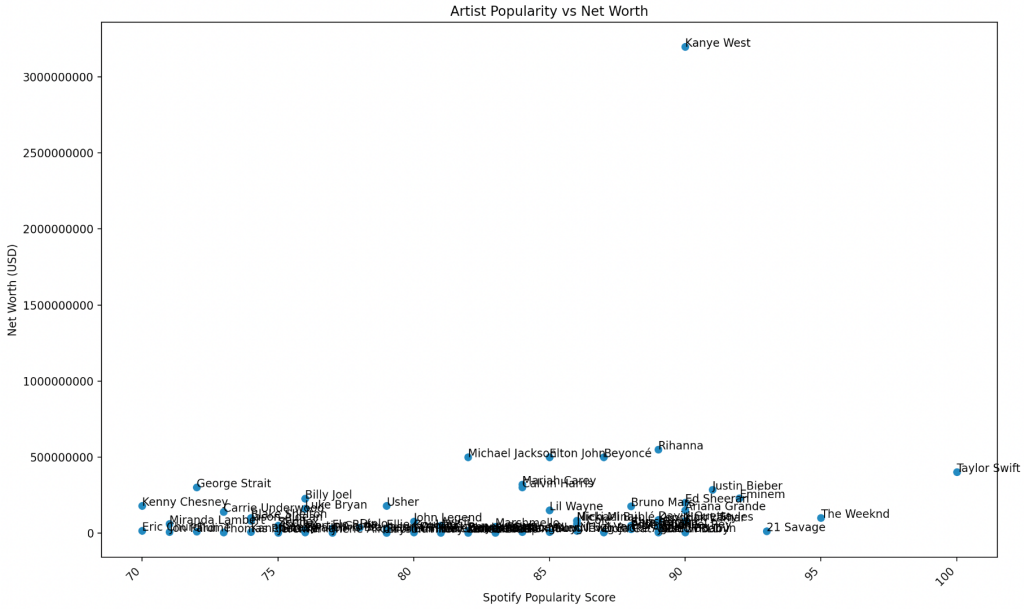
calculations.txt

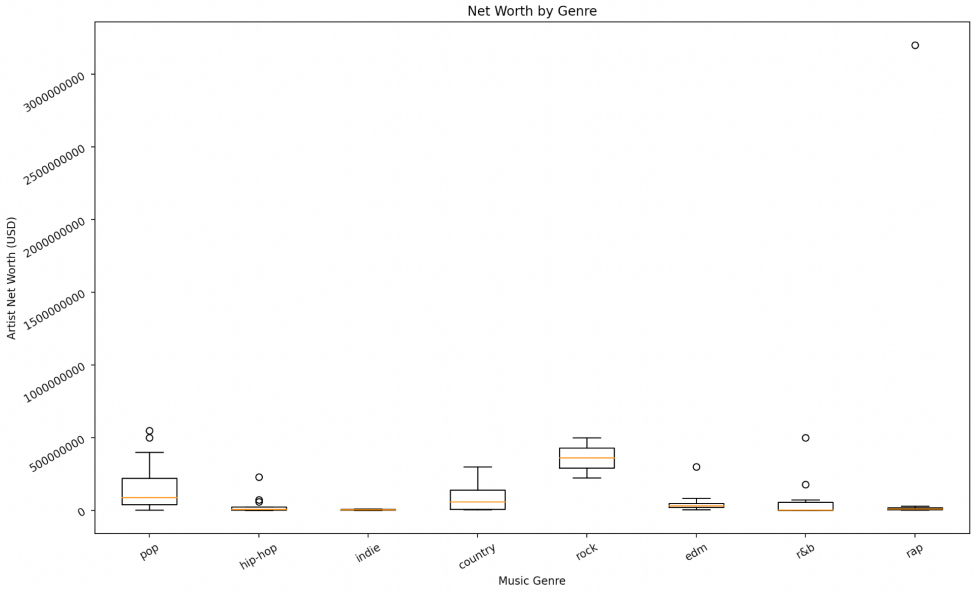
```
pop max net worth: Rihanna, $550000000
pop min net worth: SZA, $3000000
hip-hop max net worth: Eminem, $230000000
hip-hop min net worth: Lil Peep, $300000
indie max net worth: G-Eazy, $12000000
indie min net worth: Labrinth, $2000000
country max net worth: George Strait, $300000000
country min net worth: Luke Combs, $5000000
rock max net worth: Elton John, $500000000
rock min net worth: Billy Joel, $225000000
edm max net worth: Calvin Harris, $300000000
edm min net worth: Bebe Rexha, $5000000
r&b max net worth: Michael Jackson, $500000000
r&b min net worth: Jhené Aiko, $1000000
rap max net worth: Kanye West, $3200000000
rap min net worth: Polo G, $3000000

Correlation between artist age and net worth: 0.2353462605608429

Covariance between artist age and net worth:
[[1.05113481e+02 9.45063783e+08]
 [9.45063783e+08 1.53408427e+17]]

Correlation between artist's Spotify popularity and net worth: 0.21030966031643578

Covariance between artist's Spotify popularity and net worth:
[[4.26067901e+01 5.07563457e+08]
 [5.07563457e+08 1.36704696e+17]]

Correlation between artist's age and net worth without Kanye: 0.4484852979982345

Correlation between artist's Spotify popularity and net worth without Kanye: 0.2424598317683356
```

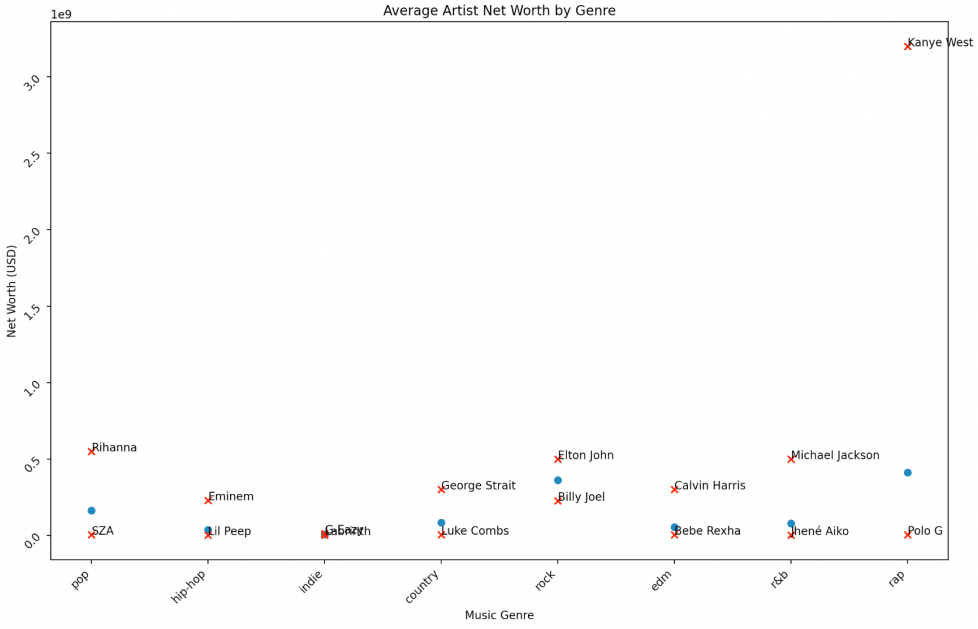
Visualizations Created

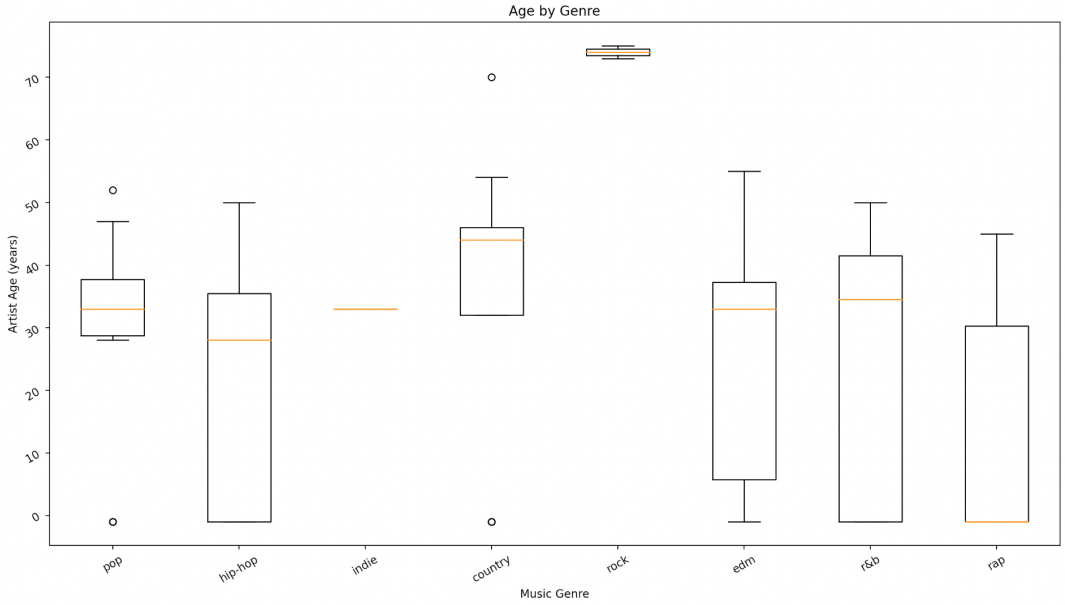
Note: some visualizations, such as the scatterplots, are better viewed after running the code. Matplotlib allows you to zoom into the plots so that you can read the labels.






Net Worth by Genre
Artist Net Worth (USD)
3000000000
2500000000
2000000000
1500000000
1000000000
500000000
0
pop
hip-hop
indie
country
rock
edm
r&b
rap
Music Genre


Average Artist Net Worth by Genre
1e9
Net Worth (USD)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Kanye West
Rihanna
Elton John
Michael Jackson
George Strait
Calvin Harris
Eminem
Billy Joel
SZA
Lil Peep
Luke Combs
Bebe Rexha
Jhené Aiko
Polo G
pop
hip-hop
indie
country
rock
edm
r&b
rap
Music Genre



Instructions for Running Code

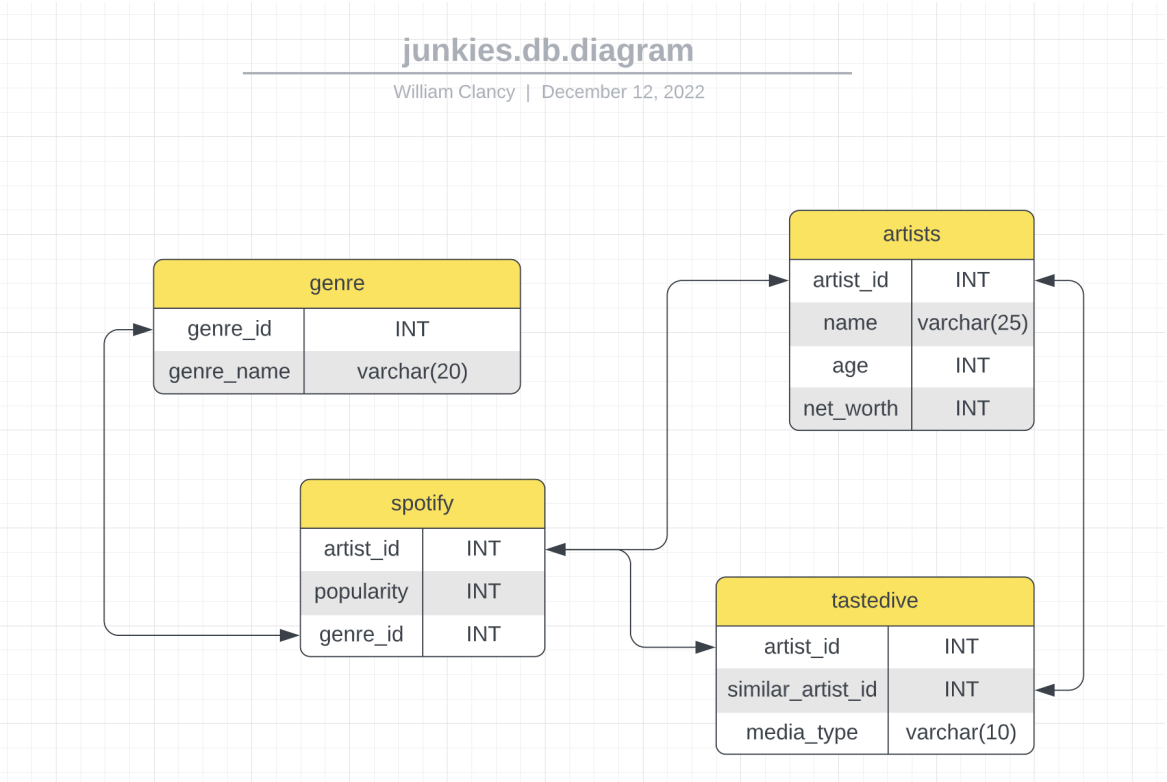
It may be required for you to pip install Unidecode before running our project. As for running the code, the biggest importance is the sequence in which to run the individual files. First, we must run the storingapi.py file to create the individual database tables for the APIs that we're working with. The database should already be prepopulated, however if you desire to start the database from scratch, uncomment the # table_reset(cur, conn) located in the main function. This will completely wipe the database. Second, we would run the spotify.py file to get a list of artists based on genre and update both the artists and spotify table with the corresponding artists' values. For step 2, you must run this file multiple times in order to input different genres for the genre table in the database. Third, run the tastedive.py file to input an artist's name and get a list of similar artists recommended by the Tastedive API, which is then inserted into the tastedive table. Finally, run the calculations.py file to calculate various figures and create visualizations using matplotlib.

Documentations for Functions Written

Within the storingapi.py file:

- There are five functions that all are regarding the database and table creation and setup to store the information that we are retrieving from the different API's.
- The functions take the database 'junkies.db' and set it up then go through the next four functions taking a cur and conn argument to create the four different tables that we have, one for artists, one for spotify data, one for tastedive data, and one for the genre information. They all create the table and column names for the information that we will then retrieve from the API's.

Name	Type	Schema
Tables (4)		
artists		CREATE TABLE artists (artist_id INTEGER PRIMARY KEY,name TEXT UNIQUE,age INTEGER,net_worth INTEGER)
arti...	INTEGER	"artist_id" INTEGER
name	TEXT	"name" TEXT UNIQUE
age	INTEGER	"age" INTEGER
net...	INTEGER	"net_worth" INTEGER
genre		CREATE TABLE genre (genre_id INTEGER PRIMARY KEY,genre_name TEXT UNIQUE)
gen...	INTEGER	"genre_id" INTEGER
gen...	TEXT	"genre_name" TEXT UNIQUE
spotify		CREATE TABLE spotify (artist_id INTEGER PRIMARY KEY,popularity INTEGER,genre_id INTEGER)
arti...	INTEGER	"artist_id" INTEGER
pop...	INTEGER	"popularity" INTEGER
gen...	INTEGER	"genre_id" INTEGER
tastedive		CREATE TABLE tastedive (artist_id INTEGER,similar_artist_id INTEGER)
arti...	INTEGER	"artist_id" INTEGER
simi...	INTEGER	"similar_artist_id" INTEGER
Indices (0)		
Views (0)		
Triggers (0)		



Within the spotify.py file:

- The first function getArtists asks the user to input a genre and then will take that genre and add it to the genre table in the database and assign it a genre_id. After that, the function uses the Spotify API to retrieve the top 25 most popular artists within that genre. It is worth noting that because the search will return some artists that are not individuals, it is likely that less than 25 artists will be added to the database every time. The artists are then put through the checkStatus function where the Music Story API will check to see if the artist that is an individual or a band, as we are only interested in artists who are individuals. If the checkStatus function returns true, the function update_artist_table is called and and obtains corresponding data (net worth and age) from API Ninjas for the artist. If age or net worth can not be obtained by the API or do not exist in the API database, it is defaulted to -1. This allows us to filter out null data for our calculations and visualizations by using SQL WHERE clauses. It then updates the artists database table with the corresponding net worth and age for each artist. The artist, net worth, and age is added to the artists table and the Spotify popularity data is added to the spotify table

along with the corresponding artist_id so that artists name is not repeated in multiple tables.

Within the tastedive.py file:

- The function get_artist_from_tastedive asks the user to input the name of a specific artist and then will take that artist's name and returns a JSON dictionary of lists that contain the names of similar artists recommended by Tastedive as given the media type of the artist (in this case always being music). Besides the artist name (given by the user) being one of the url parameters, some other parameters include the media type, the access key, as well as the limit of how many entries to return (in this case being 25). For each recommended artist returned by the Tastedive API, the checkStatus function is called in order to determine if the artist is a band or individual. If it is an individual, the recommended artist's net worth and age are obtained via update_artist_table function. Each recommended artist is inserted into the artists table along with their net worth and age. The function then inserts the artist_id of the given artist, and the artist_id of recommended artists, into the tastedive table in the database.

Within the calculations.py file:

- The function get_genre_list retrieves the list of input genres from the database using an SQL query, then reformats the list of tuples into a list of strings representing genre names using a list comprehension. It then returns the list of genres.
- The function avg_networth_genre finds the average net worth of all artists in a given genre.
- The function max_networth_genre finds the maximum net worth of all artists in a given genre and returns a tuple with the name of the artist with the highest net worth and their net worth.
- The function min_networth_genre finds the minimum net worth of all artists in a given genre and returns a tuple with the name of the artist with the lowest net worth and their net worth.
- The function avg_net_worth_by_genre_vis creates a plot of the maximum, minimum, and average net worths per genre by looping through each genre in the database and calling max_networth_genre, min_networth_genre, and avg_networth_genre

- The function net_worth_by_genre_vis plots a box plot of every artist in a given genre for every genre in the database.
- The function popularity_vs_net_worth_vis plots each artist in the database on a scatter plot, with the x variable being their Spotify popularity value, and the y variable being their net worth in USD.
- The function age_vs_net_worth_vis plots each artist in the database on a scatter plot, with the x variable being their age in years, and the y variable being their net worth in USD.
- The function correlation_age_networth calculates the correlation between age and networth for artists in the database.
- The correlation_age_networth and correlation_popularity_networth functions calculate the correlations between age vs networth and Spotify popularity vs networth, respectively.
- The covariance_age_networth and covariance_popularity_networth functions calculate the covariance for age vs networth and Spotify popularity vs networth, respectively.
- The correlation_age_networth_exclude_outlier and correlation_popularity_networth_exclude_outlier functions exist due to Kanye West being a very high outlier in terms of net worth (though this data may be outdated given recent events). The functions calculate the correlations between age vs networth and Spotify popularity vs networth without including Kanye West.
- The function age_by_genre_vis creates box plots of artists' age for each genre in the database.
- The function correlation_popularity_networth calculates the correlation between age and networth for artists in the database.
- The main function allows the user to select which visualization to appear. It also writes the calculations to a file called "calculations.txt".

Documentations for Resources Used

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
Dec 4	Creating tables for the database	lucidcharts.com	Was able to visualize the database through

			a DBMS diagram
Dec 6	Spotify does not provide information on whether or not an artist is an individual or band	https://developers.music-story.com/developers	Using Music Story along with XML Element Tree, we are able to determine the status of an artist and only add them to the database if they are an individual
Dec 10	Music Story API does not recognize accent marks on characters	https://pypi.org/project/Unidecode/	Using Unidecode, we were able to convert accented characters into regular characters in order to properly use the Music Story API
Dec 10	Creating visualizations with data	w3schools.com	Was able to create scatter plots + box & whisker plots to visualize calculations