Final Project Report: APIs, SQL, and Visualizations

Sing a Pysong: Sofia Gryzenia, Devon Vitale, and Lauren Pyfer

Dr. Barbara Ericson

SI 206 Data Oriented Programming

December 12, 2022

Link to GitHub repository: https://github.com/sofiagryzenia/Final-Project-SI-206

Project Goals

Our goals for this project was to take a topic we are passionate about and successfully apply what we have learned throughout the course to create a database, calculations, and visualizations of data about something we may have not known before. We wanted to utilize Spotify to allow us to compare our top songs in the past 6 months to the all time top songs based on Spotify streams. We wanted an API to gather a user's spotify data to reveal their top streamed songs because it is a platform we all use frequently. Our next goal was to then take the global artist top 100 rating from Wikipedia and use an API to gather data regarding the time, location, date, and venue of their next concert. Once we successfully gathered all of this intended information into a database using SQLite it was crucial that the data stored had a limit of 25 items for each time it was executed.

For our calculations we wanted to find different ways to discover something new from all tables to make comparisons from what was collected. First, we wanted to get the locations of the top 25 streamed artists' next venue which could then be visualized on a map plot. Next, we had the goal of getting the top 25 streamed artists based on the total streams of their songs in the top 100 Wikipedia page. Our final goal for the calculations was to further analyze the artist's top streaming by converting the streams into the billions format. The visualizations had to be based on the calculations made, so we decided to show the calculations for the longitude and latitude of the next location on a marked global map. In addition, for the Wikipedia top streamed artists based on songs we wanted to create a chart comparing the top 25.

The final goal for the overall project report was to be able to present our project completion in an organized understandable manner. Even though we expected to run into issues, and did, by documenting our process and blocks along the way we hoped to clearly document the finalized report including all steps.

Achieved Goals

We successfully accessed two APIs and one website using BeautifulSoup to gather specific data from each. Our first goal was to gather data from Spotipy, Spotify's Python library for Web API, to get a user's top 100 songs in the last six months, the popularity ranking, the title of the song, and artist. Then we used a Wikipedia page that contains a list of the top songs streamed on Spotify of all time. We utilized BeautifulSoup to scrape the page to gather the top 100 ranking of streamed songs, the title of the song, artist, and number of streams in billions. Then our third API, Bandsintown, then got the location, venue, date, and time of the top 100 music artists of all time based on all digital streaming services (Apple

Music, Spotify, iTunes, Youtube, TikTok, Shazam, and Deezer). After we gathered all this data to be returned in a list of tuples, the data group was inputted into a table created and sorted into the respective columns. Each time we run our code 25 items out of the 100 are stored without duplicating or changing the existing data using a for loop with a break statement once 25 is met. Therefore, after the code is run 4 times, all 100 items are inputted into the database in the correct columns and order.
When deciding on the calculations, we toggled between a couple combinations before finalizing our computations. For the first calculation we used the data from the Bandsintown API to get the artist name, upcoming event count, city location, and country location for the next upcoming event. For the second calculation we combined data from the Spotipy API and Wikipedia data to calculate the percentage of popularity between the top streamed songs of all time from the Wikipedia page and the top songs of the past six months from Spotify. We created a Spotify Developer App that sorted a persons most listened to tracks. We imported a medium_term which generates the users top listened tracks from the last six months. We were able to dive into the keys of what information was provided and grab a popularity statistic implemented in Spotify. From there we could sort by the popularity of the track. Our third calculation took the list of streams from the top 100 songs in the Wikipedia page and found all the artists who were listed numerous times for having a top song and combined their streams. Our fourth and extra calculation displayed the top 18 artists from the Bandsintown api, and the amount of upcoming events/concerts they have between the dates 2022-12-13,2023-12-9.

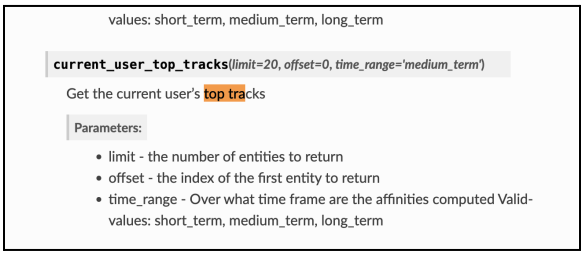
Finally, we completed the project report in a manner that was in depth, organized, and clearly written. Even though we face many problems we worked as a team to best support each other and solve the issues at hand.

Problems Faced

Throughout the project we learned a lot about the value of debugging, the importance of well-organized code, and effective communication. Our first problem that we faced was during our planning process. Originally, we struggled to grasp the type of APIs and websites that we could gather data from. This confusion resulted in a project plan that was not completely thought through and that we were not excited about due to the topics we selected. Therefore we had to change our project idea very late into the intended work timeline, setting us back progress wise. The next problem we faced was catching and fixing bugs in our code. A lot of time and frustration was unnecessarily occurring due to lack of continuous testing and debugging. Many of the flaws in our code came from typos, logic errors, and other small mistakes that would cause code to fail. However, once we started being more vigilant about being thorough with each step, these problems dissipated. A specific example of this occurring happened when trying to add the Wikipedia BeautifulSoup data into the database. What we thought to be a problem with the line of code that was supposed to be checking if an item was in the database actually ended up being a typo in one of the attribute names.

A problem faced when accessing the Spotify API was having to create a Spotify developer account in order to access a user's information. At first, we kept getting an error URI whenever we tried to run the code. We finally figured out that we needed to create a Spotify client ID and Spotify Client Secret and set those equal to the access codes in the Spotify Developer settings. But, even though the Client ID and Client Secret was set to the correct access codes, we still had to link the actual Spotify website by

using a client redirect ID of 'http://localhost:8888/callback' which had to be connected to the Spotify Developer settings in order to have a redirect link to notify the person to log onto Spotify. This definitely was the biggest challenge because there were many variables that needed to be correctly associated with arguments. As a result, we had to figure out which environmental characters we needed and assign it to the correct variable, or argument.

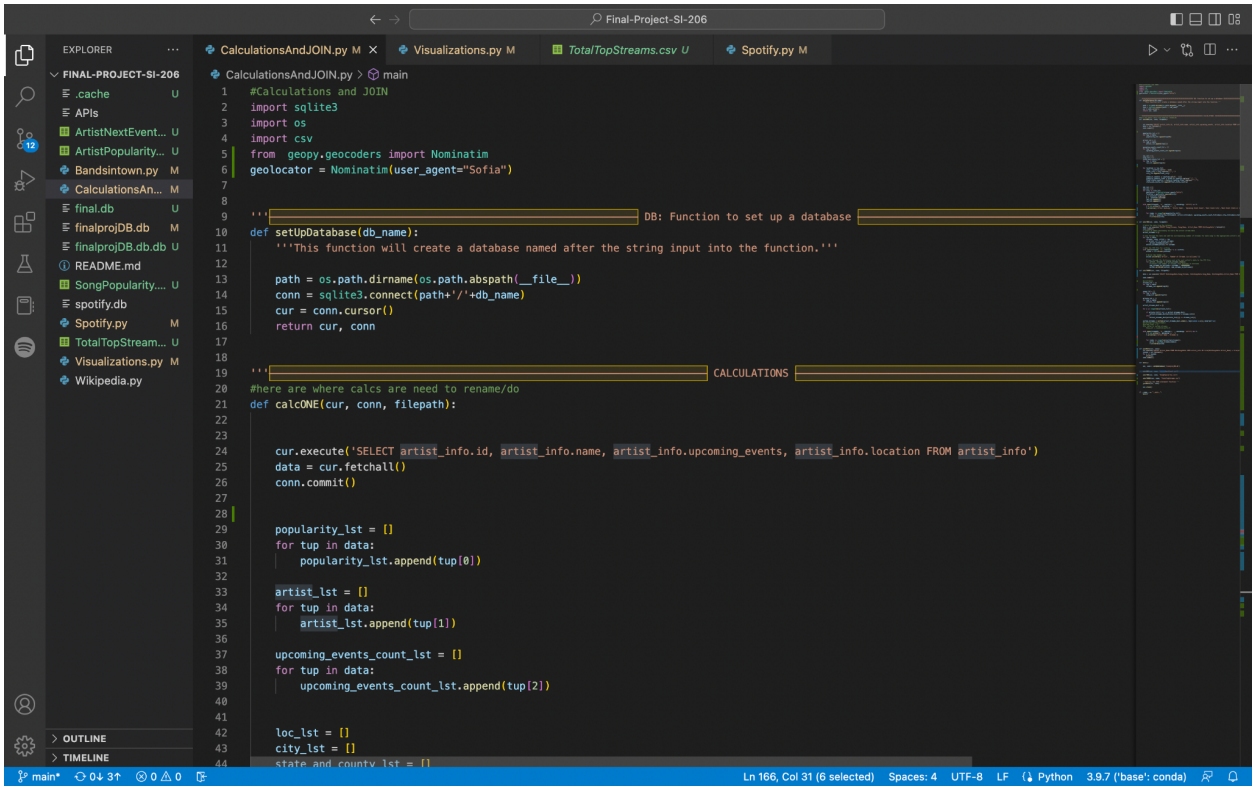


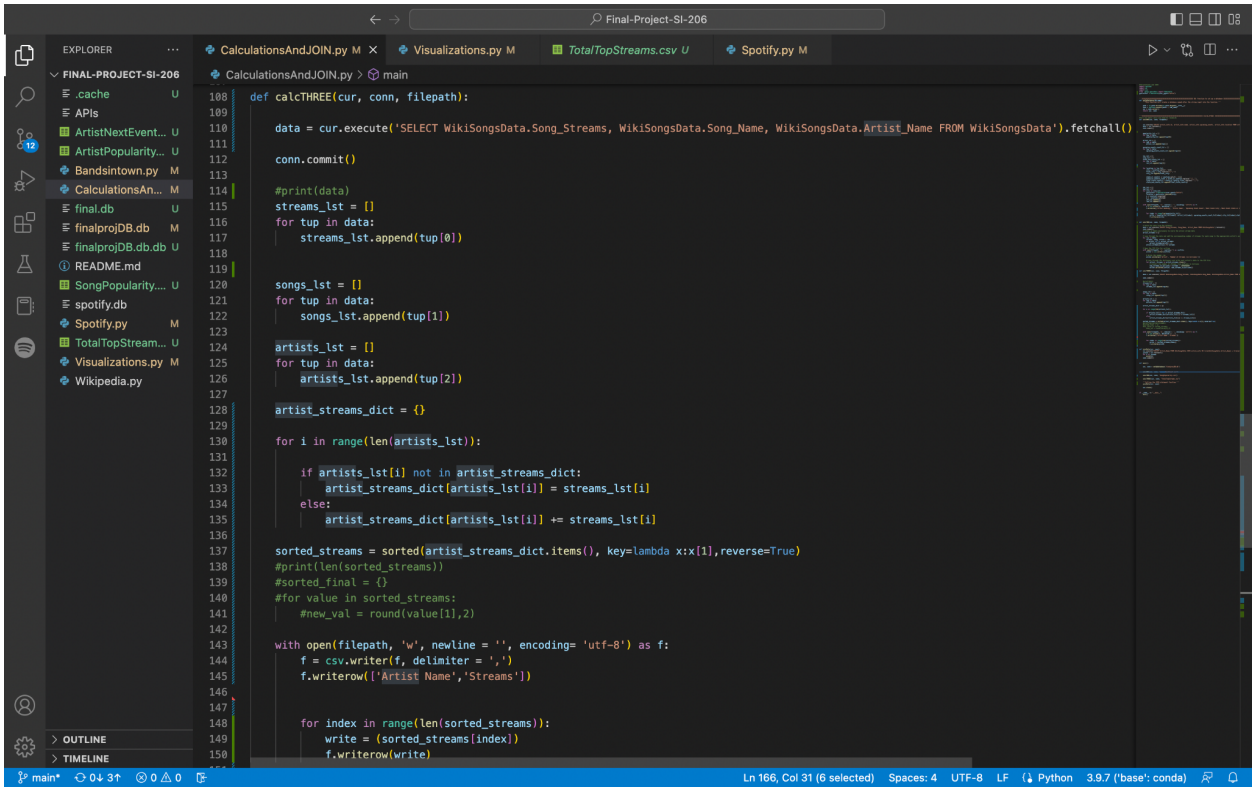
Unfortunately Spotipy has limitations on how much data you can access when using the Spotipy API. The maximum number of items we could receive from the API were 50. Which means, when we added the data to our final database. Instead of getting 25 new items four different times when the code ran, the user only gets it twice. We have looked for other ways but on the Spotipy tool website it claims certain functions have limitations and the function we used, 'user_top_tracks', has a limitation of how much information one can process from the API.

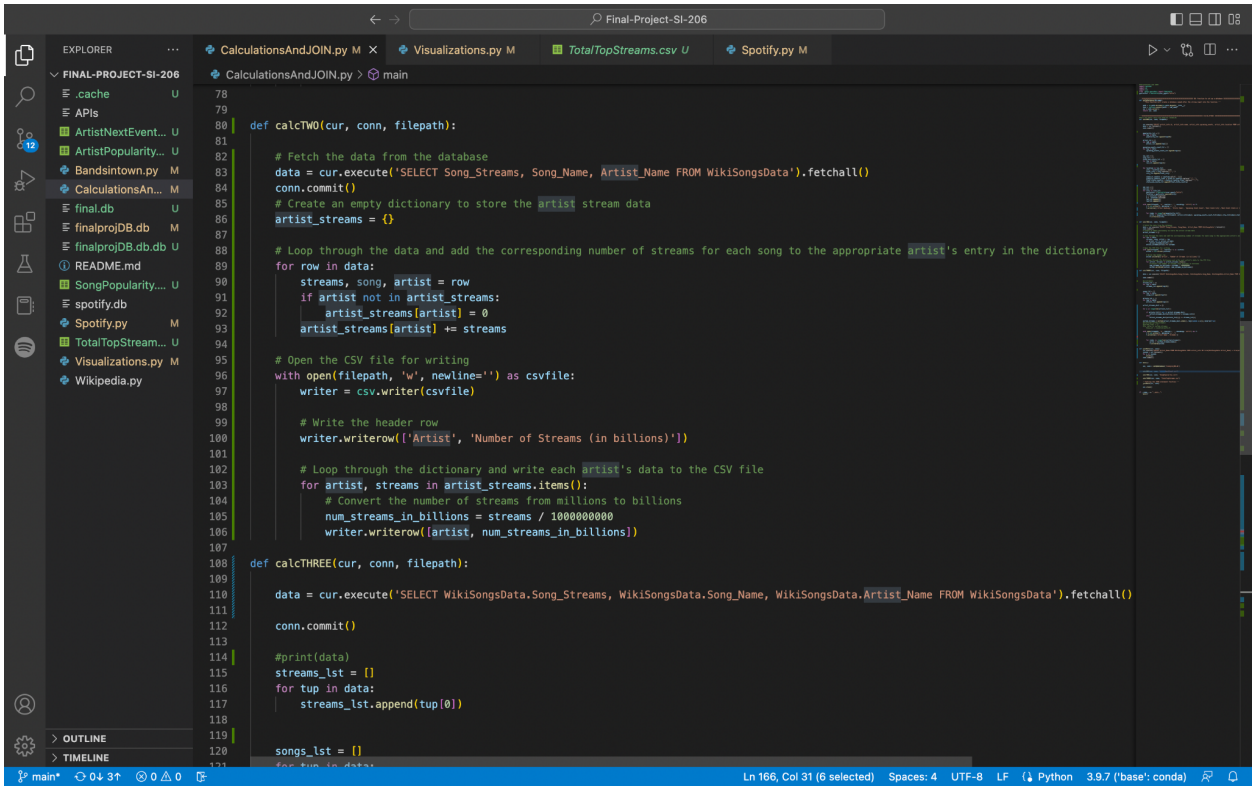
When using the Bandsintown api, we faced problems with how to format the data. After inputting an artist name to get the upcoming events in the api some artists would have no response data and others would have multiple responses. To solve this problem, all of the data besides the artist id, name, and upcoming event count was stored as a list. This provided more data to work with, otherwise the response would have just been the next upcoming event. However, because some of the data was stored as lists, when we accessed the data to combine it all to complete calculations the data had a very messy format. This resulted in a lot of time spent debugging and cleaning up the response data so it could be used for calculations and visualizations.

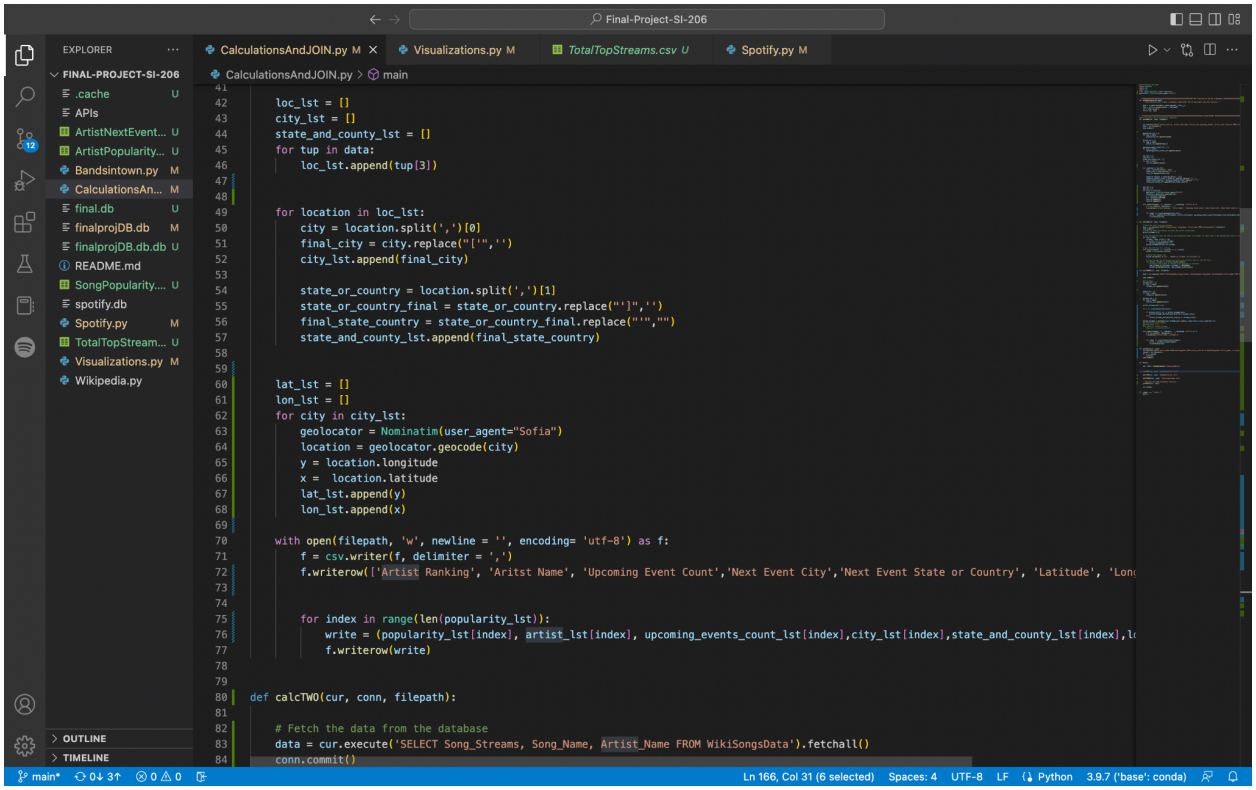
Lastly, we faced multiple problems with GitHub. When we were all working on the same file, we faced multiple errors when trying to upload the code to the repository. This led to errors when we tried to run or code in a file that had been modified and uploaded to the repository while one of us was working on it.

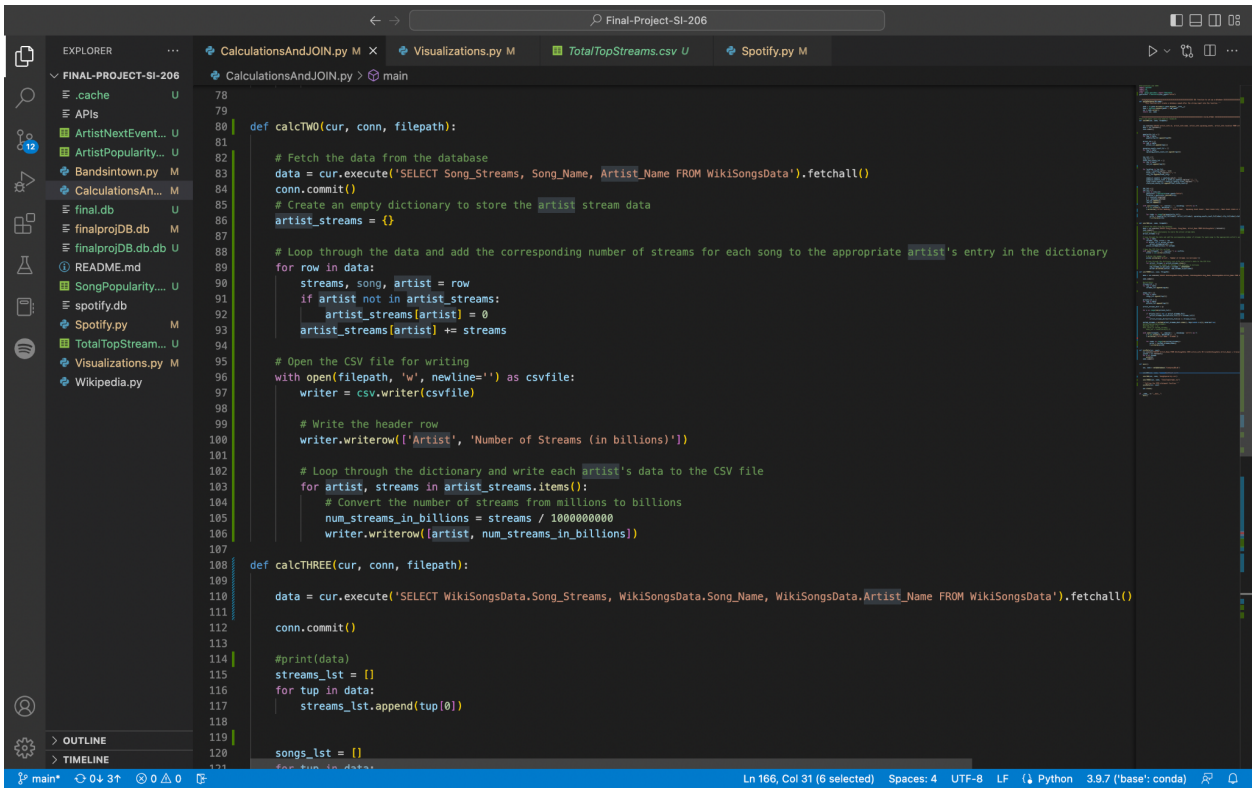
File that contains the calculations from the data in the database


Final-Project-SI-206
EXPLORER
CalculationsAndJOIN.py M
Visualizations.py M
TotalTopStreams.csv U
Spotify.py M
FINAL-PROJECT-SI-206
.cache
APIs
ArtistNextEvent... U
ArtistPopularity... U
Bandsintown.py M
CalculationsAn... M
final.db U
finalprojDB.db M
finalprojDB.db.db U
README.md
SongPopularity.... U
spotify.db
Spotify.py M
TotalTopStream... U
Visualizations.py M
Wikipedia.py
OUTLINE
TIMELINE
CalculationsAndJOIN.py > main
#Calculations and JOIN
import sqlite3
import os
import csv
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="Sofia")
''' DB: Function to set up a database
def setUpDatabase(db_name):
'''This function will create a database named after the string input into the function.'''
path = os.path.dirname(os.path.abspath(__file__))
conn = sqlite3.connect(path+'/'+db_name)
cur = conn.cursor()
return cur, conn
''' CALCULATIONS
#here are where calcs are need to rename/do
def calcONE(cur, conn, filepath):
cur.execute('SELECT artist_info.id, artist_info.name, artist_info.upcoming_events, artist_info.location FROM artist_info')
data = cur.fetchall()
conn.commit()
popularity_lst = []
for tup in data:
popularity_lst.append(tup[0])
artist_lst = []
for tup in data:
artist_lst.append(tup[1])
upcoming_events_count_lst = []
for tup in data:
upcoming_events_count_lst.append(tup[2])
loc_lst = []
city_lst = []
state_and_county_lst = []
main*
0↓ 3↑
0 0
Ln 166, Col 31 (6 selected)
Spaces: 4
UTF-8
LF
Python
3.9.7 ('base': conda)


Final-Project-SI-206
EXPLORER
CalculationsAndJOIN.py M
Visualizations.py M
TotalTopStreams.csv U
Spotify.py M
FINAL-PROJECT-SI-206
.cache
APIs
ArtistNextEvent... U
ArtistPopularity... U
Bandsintown.py M
CalculationsAn... M
final.db U
finalprojDB.db M
finalprojDB.db.db U
README.md
SongPopularity.... U
spotify.db
Spotify.py M
TotalTopStream... U
Visualizations.py M
Wikipedia.py
OUTLINE
TIMELINE
CalculationsAndJOIN.py > main
def calcTHREE(cur, conn, filepath):
data = cur.execute('SELECT WikiSongsData.Song_Streams, WikiSongsData.Song_Name, WikiSongsData.Artist_Name FROM WikiSongsData').fetchall()
conn.commit()
#print(data)
streams_lst = []
for tup in data:
streams_lst.append(tup[0])
songs_lst = []
for tup in data:
songs_lst.append(tup[1])
artists_lst = []
for tup in data:
artists_lst.append(tup[2])
artist_streams_dict = {}
for i in range(len(artists_lst)):
if artists_lst[i] not in artist_streams_dict:
artist_streams_dict[artists_lst[i]] = streams_lst[i]
else:
artist_streams_dict[artists_lst[i]] += streams_lst[i]
sorted_streams = sorted(artist_streams_dict.items(), key=lambda x:x[1],reverse=True)
#print(len(sorted_streams))
#sorted_final = {}
#for value in sorted_streams:
#new_val = round(value[1],2)
with open(filepath, 'w', newline = '', encoding= 'utf-8') as f:
f = csv.writer(f, delimiter = ',')
f.writerow(['Artist Name','Streams'])
for index in range(len(sorted_streams)):
write = (sorted_streams[index])
f.writerow(write)
main*
0↓ 3↑
0 0
Ln 166, Col 31 (6 selected)
Spaces: 4
UTF-8
LF
Python
3.9.7 ('base': conda)

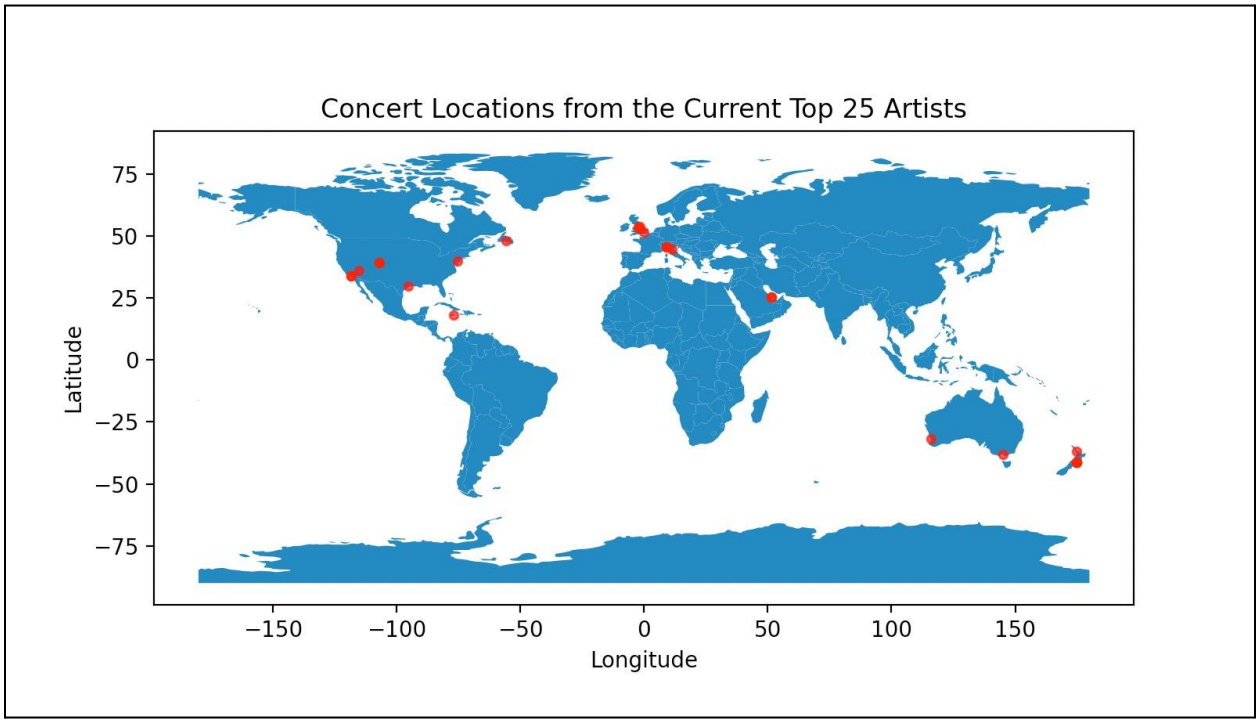

Final-Project-SI-206
EXPLORER
FINAL-PROJECT-SI-206
.cache U
APIs
ArtistNextEvent... U
ArtistPopularity... U
Bandsintown.py M
CalculationsAn... M
final.db U
finalprojDB.db M
finalprojDB.db.db U
README.md
SongPopularity.... U
spotify.db
Spotify.py M
TotalTopStream... U
Visualizations.py M
Wikipedia.py
OUTLINE
TIMELINE
CalculationsAndJOIN.py M
Visualizations.py M
TotalTopStreams.csv U
Spotify.py M
CalculationsAndJOIN.py > main
def calcTWO(cur, conn, filepath):
# Fetch the data from the database
data = cur.execute('SELECT Song_Streams, Song_Name, Artist_Name FROM WikiSongsData').fetchall()
conn.commit()
# Create an empty dictionary to store the artist stream data
artist_streams = {}
# Loop through the data and add the corresponding number of streams for each song to the appropriate artist's entry in the dictionary
for row in data:
streams, song, artist = row
if artist not in artist_streams:
artist_streams[artist] = 0
artist_streams[artist] += streams
Open the CSV file for writing
with open(filepath, 'w', newline='') as csvfile:
writer = csv.writer(csvfile)
# Write the header row
writer.writerow(['Artist', 'Number of Streams (in billions)'])
# Loop through the dictionary and write each artist's data to the CSV file
for artist, streams in artist_streams.items():
# Convert the number of streams from millions to billions
num_streams_in_billions = streams / 1000000000
writer.writerow([artist, num_streams_in_billions])
def calcTHREE(cur, conn, filepath):
data = cur.execute('SELECT WikiSongsData.Song_Streams, WikiSongsData.Song_Name, WikiSongsData.Artist_Name FROM WikiSongsData').fetchall()
conn.commit()
#print(data)
streams_lst = []
for tup in data:
streams_lst.append(tup[0])
songs_lst = []
main* 0↓ 3↑ 0 0
Ln 166, Col 31 (6 selected) Spaces: 4 UTF-8 LF Python 3.9.7 ('base': conda)


Final-Project-SI-206
EXPLORER
CalculationsAndJOIN.py M
Visualizations.py M
TotalTopStreams.csv U
Spotify.py M
FINAL-PROJECT-SI-206
.cache
APIs
ArtistNextEvent... U
ArtistPopularity... U
Bandsintown.py M
CalculationsAn... M
final.db U
finalprojDB.db M
finalprojDB.db.db U
README.md
SongPopularity.... U
spotify.db
Spotify.py M
TotalTopStream... U
Visualizations.py M
Wikipedia.py
OUTLINE
TIMELINE
CalculationsAndJOIN.py > main
loc_lst = []
city_lst = []
state_and_county_lst = []
for tup in data:
loc_lst.append(tup[3])
for location in loc_lst:
city = location.split(',')[0]
final_city = city.replace("['",'')
city_lst.append(final_city)
state_or_country = location.split(',')[1]
state_or_country_final = state_or_country.replace("']",'')
final_state_country = state_or_country_final.replace("'","")
state_and_county_lst.append(final_state_country)
lat_lst = []
lon_lst = []
for city in city_lst:
geolocator = Nominatim(user_agent="Sofia")
location = geolocator.geocode(city)
y = location.longitude
x = location.latitude
lat_lst.append(y)
lon_lst.append(x)
with open(filepath, 'w', newline = '', encoding= 'utf-8') as f:
f = csv.writer(f, delimiter = ',')
f.writerow(['Artist Ranking', 'Aritst Name', 'Upcoming Event Count','Next Event City','Next Event State or Country', 'Latitude', 'Lon
for index in range(len(popularity_lst)):
write = (popularity_lst[index], artist_lst[index], upcoming_events_count_lst[index],city_lst[index],state_and_county_lst[index],lo
f.writerow(write)
def calcTWO(cur, conn, filepath):
# Fetch the data from the database
data = cur.execute('SELECT Song_Streams, Song_Name, Artist_Name FROM WikiSongsData').fetchall()
conn.commit()
main*
Ln 166, Col 31 (6 selected)
Spaces: 4
UTF-8
LF
Python
3.9.7 ('base': conda)

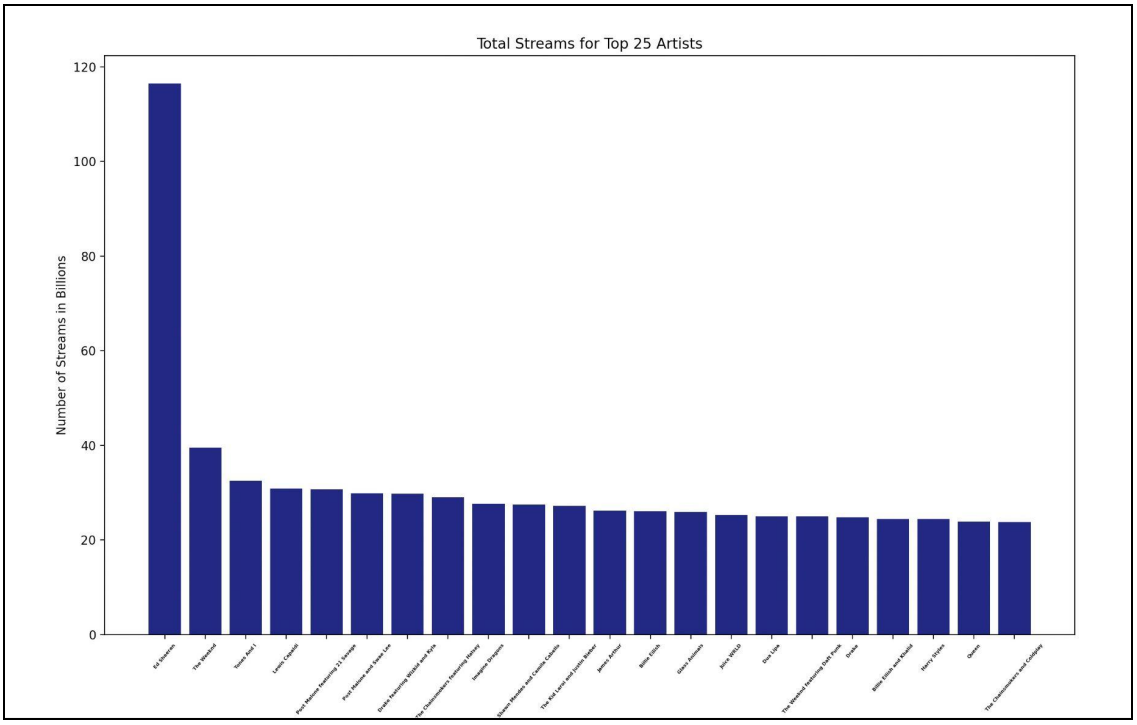

Final-Project-SI-206
EXPLORER
CalculationsAndJOIN.py M
Visualizations.py M
TotalTopStreams.csv U
Spotify.py M
FINAL-PROJECT-SI-206
.cache
APIs
ArtistNextEvent... U
ArtistPopularity... U
Bandsintown.py M
CalculationsAn... M
final.db U
finalprojDB.db M
finalprojDB.db.db U
README.md
SongPopularity.... U
spotify.db
Spotify.py M
TotalTopStream... U
Visualizations.py M
Wikipedia.py
OUTLINE
TIMELINE
CalculationsAndJOIN.py > main
def calcTWO(cur, conn, filepath):
# Fetch the data from the database
data = cur.execute('SELECT Song_Streams, Song_Name, Artist_Name FROM WikiSongsData').fetchall()
conn.commit()
# Create an empty dictionary to store the artist stream data
artist_streams = {}
# Loop through the data and add the corresponding number of streams for each song to the appropriate artist's entry in the dictionary
for row in data:
streams, song, artist = row
if artist not in artist_streams:
artist_streams[artist] = 0
artist_streams[artist] += streams
Open the CSV file for writing
with open(filepath, 'w', newline='') as csvfile:
writer = csv.writer(csvfile)
# Write the header row
writer.writerow(['Artist', 'Number of Streams (in billions)'])
# Loop through the dictionary and write each artist's data to the CSV file
for artist, streams in artist_streams.items():
# Convert the number of streams from millions to billions
num_streams_in_billions = streams / 1000000000
writer.writerow([artist, num_streams_in_billions])
def calcTHREE(cur, conn, filepath):
data = cur.execute('SELECT WikiSongsData.Song_Streams, WikiSongsData.Song_Name, WikiSongsData.Artist_Name FROM WikiSongsData').fetchall()
conn.commit()
#print(data)
streams_lst = []
for tup in data:
streams_lst.append(tup[0])
songs_lst = []
main*
Ln 166, Col 31 (6 selected)
Spaces: 4
UTF-8
LF
Python
3.9.7 ('base': conda)

Visualizations

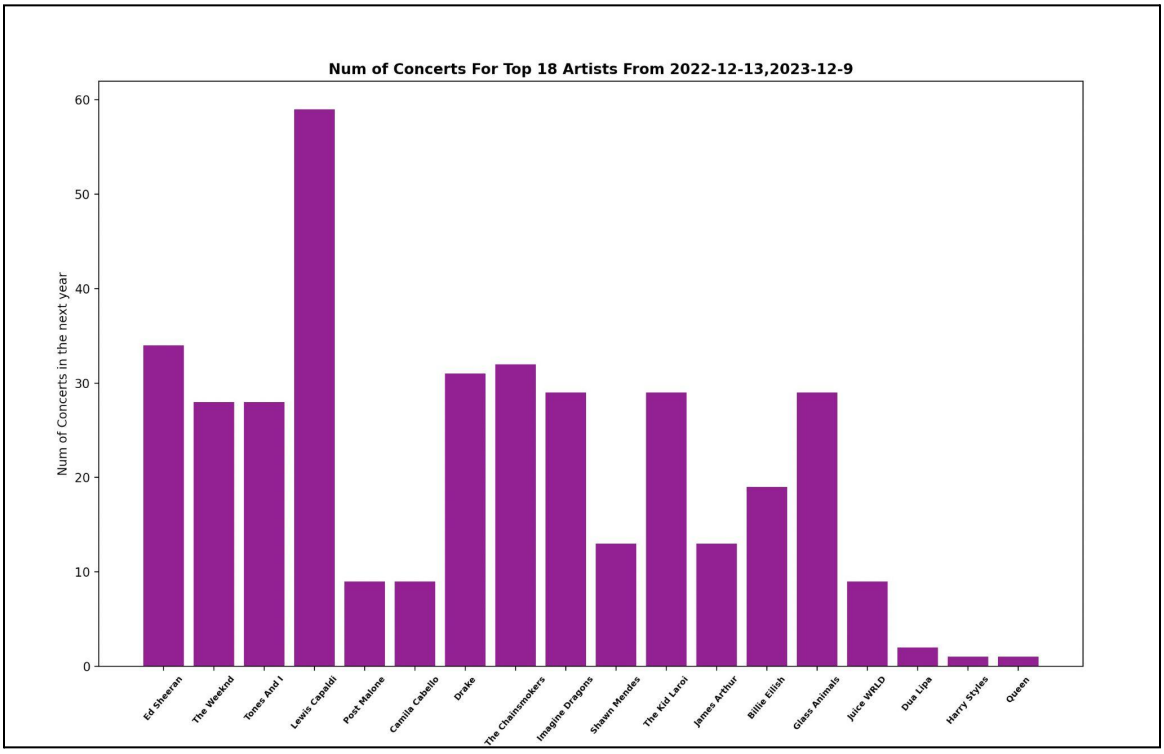
Events_visualization:



Streams_visualization:



Num_concerts_visulization:



Instructions for running our code

To run our Spotipy implementation the user must install the Spotipy tool in order to run the code. After installing you must import Spotipy, from spotipy.oauth2 import SpotifyClientCredentials, and from spotipy.oauth2 import SpotifyOAuth. All three of those must be installed and imported for the Spotipy to run successfully.

Before running our calculations and .join implementation, it is important the user has run each of our apis in order to store the data in the database.

To run our calculations and .join implementation, the user must install Nominatim and geopy because it is used in our first function in order to calculate the latitude and longitude values for an inputted city.

To run our visualization implementation, the user must install geopandas in order to view our visuals. Specifically for the first function (the world map visualization) the user must also install matplotlib.pyplot.

Documentation for each function

The **def setUpDatabase(db_name)** function is used to set up a SQLite database with the given 'db_name' using the 'os.path' module to get the current dictionary where the script is being run. Then it concatenates the dictionary path with the given 'db_name' to form the full path to the database file. Next, the function uses the 'sqlite3' module to connect to the database file if it does not already exist. Finally, the function returns the cursor and connection objects.

The **def main()** function is the main entry point of the program. It calls the function to set up the database and all tables. In addition it calls the visual functions, calculation functions, and JOIN statement. Once all called steps have been completed, the function closes the database connection and exits.

Wikipedia.py:

The **def getSongs()** function is used to retrieve data from the specified Wikipedia page using BeautifulSoup to parse and return a list of tuples containing the ranking of the top 100 most-streamed songs of all time on Spotify, along with their rank, number of streams, and artist. Once the HTML is parsed the function uses the 'find_all' method to search for the table containing the song data. Then it iterates over each row in the table, using the 'find' and 'find_all' methods to extract the data. The extracted data is then added to a list of tuples, which is returned by the function. The song ranking is returned as an integer while the streams are returned as a float.

The **def setUpSongsTable()** function creates the WikiSongsData table if it doesn't exist with the song ranks, song streams, song name, and artist name pulled from the list of tuples and sorts the data into columns.

Spotify.py:

The **def Spotify()** is a Spotify client implementation. It sets the necessary environment variables for connecting to the Spotify API using the 'os.environ' object in Python. It then initializes a Spotify client using the 'spotipy' library and authenticates it using the 'client_credentials_manager' object. The function then retrieves the user's top tracks from Spotify, extracts the popularity, name, and artist information, and returns them as three separate lists. The range 'medium_term' is used in Spotify so the data has a timeline from 6 months to the present.

The **def SetUpSongsTable()** creates a table that inserts the lists from the previous Spotify function into three columns sorted by artist name, song name, and popularity.

Bandsintown.py:

The **def get_artist_lst()** function uses BeautifulSoup to parse the HTML content of the global artist ranking webpage to retrieve and return a list of the top 100 digital platform artists by only including the 14th to 114th items.

The **def get_events_info(artist_lst) f**unction is used to retrieve information about events for the list of artists given by "artist_lst" by iterating over the list of artists. The function then uses the 'json' module to parse the JSON response from the Bandsintown API and extracts relevant information about the events for each artist. This information includes the number of upcoming events, the location and venue of the events, the dates and times of the events, and the URL to purchase tickets for the events. This information is then added to a global 'data' list, which is returned once all artists have been processed.

The **def create_artists_table(cur,conn,data, artist_lst)** function is used to create a table in a SQLite database to store the information about the given lists of artists. First the provided cursor object is used to execute a 'DROP TABLE' query to delete the table if it already exists then a 'CREATE TABLE' with the specified columns to store the information about the events. The function then iterates over the list of artists and event information for each artist and inserts a new row into the 'artist info' table. Once all artists are processed, 'conn.commit()' is used to save the changes to the database.

CalculationsAndJOIN.py

The **def calcONE(cur, conn, filepath)** function is used to extract and process data from the SQL database. The function takes in three arguments: 'cur' and 'conn' which are the cursor and connect to the database, and 'filepath' which is the file path to the CSV file. The function uses cur to execute a SQL query that selects artist information from the 'artist_info' table in the database. Then the resulting data of

the artist's ID, anime, upcoming event count, location, and latitude, longitude of the city is processed and written to the CSV file from the specified file path.

The **def calcTWO(cur, conn, filepath)** function takes the total number of streams for an artist if they come up multiple times in the top 100 and plots their overall number of streams in billions. It fetches data from the WikiSongs table. I then created a dictionary that connects artists to the number of streams of their song. After I completed that I used a csv reader to create rows and labeled one of them songs in billions. From there I had to use a calculation which was using the amount of streams divided by 1000000 so the song would come in the value of billions.

The **def calcTHREE(cur, conn, filepath)** function is used to extract and process data from the SQL database and takes in the three arguments 'cur', 'conn', and 'filepath' which when called connect to the database and file path to the "TotalTopStreams.csv" CSV file. The function then retrieves all rows of data from the WikiSongsData table in the database, which are stored in the variable 'data'. Next, the function creates three lists: streams_lst, songs_lst, and artists_lst, which contain the values of the Songs_Streams, Song_Name, and Artist_Name columns, respectively, from each row in data. An empty dictionary called 'artist_streams_dict' is then created and uses a for loop to iterate through the values in 'artists_lst', adding the artist name as a key and setting the value to the corresponding number of streams from 'streams_lst'. The final result writes the 'sorted_streams' data to the file in CSV format with the first rows containing column headers "Artist_Name" and "Streams". Each subsequent row contains the name of the top 25 streamed artists and the total number of streams.

Visualizations.py

The def **events_visualizations(cur, csvfile**) function uses the 'pandas' and 'geopandas' libraries to read the CSV file 'ArtistNextEvent' and create a scatter plot of the data using the columns from the CSV containing the longitude and latitude of the top artists' next events. The function sets the size and color of the points to red, labels the axes 'Longitude' and 'Latitude', and titles the plot 'Concert Locations from the Current Top 25 Artists' '.

The **def streams_visualizations(cur, csvfile)** function uses the pandas library to read in the data from the CSV file "TotalTopStreams.csv ". It then creates a DataFrame object from the data and assigns it to the df variable. Next, the function extracts the data for number of streams for rows and the artists name for columns and stores in the X and Y variables. The chart is given a title, x-axis label, and y-axis. The x-axis tick labels are rotated and resized for readability. Finally, the show() method is called to display the chart.

The **def num_concerts_visualzation(cur, csvfile)** function uses the pandas library to read data from the CSV file 'ArtistNextEvent'. It then creates a DataFrame object from the data and assigns it to the df variable. Next, the function extracts the data for the number of upcoming events for rows and the artists name for columns and stores in the X and Y variables. The chart is given a title, x-axis label, and y-axis. The x-axis tick labels are rotated and resized for readability. Finally, the show() method is called to display the chart.

Resources Used

Date	Issue Description	Location of Resource	Result
12/07/2022	We needed to find a page that had data we wanted to scrape that would be useful for our goal.	https://en.wikipedia.org/wiki/List_of_most-streamed_songs_on_Spotify	Wikipedia page used for web scraping top streamed Spotify songs of all time.
12/07/2022	Need to utilize Spotipy, a wrapper for Spotify API.	https://spotipy.readthedocs.io/en/2.21.0/	How we got access to Spotify music data that can be ran in Python.
12/07/2022	We needed a second API that showed us information regarding concerts.	https://app.swaggerhub.com/apis-docs/Bandsintown/PublicAPI/3.0.1#/artist%20events/artistEvents	Website for the Bandsintown API to gather relevant data from.
12/08/2022	We wanted to try a new way of formatting Beautiful Soup so that each tag was on its own separate line so better visualize the parse tree.	https://www.skytowner.com/explore/beautiful_soup_prettify_method	This ended up working on the Wikipedia webscrape. However, it was found to be unnecessary and was removed in later commits.
12/08/2022	We kept getting the error "sqlite3.OperationalError: incomplete input" and could not figure out what to fix.	https://stackoverflow.com/questions/54116641/how-to-fix-sqlite3-operationalerror-incomplete-input-when-the-error-is-rega	The resource had an example of another user having the same error notification. The issue was solved once they realized that they were missing a closing brace. When we looked back at the code we found that the error was caused by a missing closing parenthesis and the error was solved.
12/11/2022	Bandsintown has a required parameter so we needed a list of the	https://kworb.net/itunes/	Used this website to gather the top 100 artists currently to

	top 100 artists across all platforms.		create a list that was the parameter for Bandsintown API data.
12/11/2022	Needed clarification for question 6 and 7 of the final report.	206 Piazza Page	Another student asked the same question and we got the clarification needed to properly document our code.
12/11/2022	Needed to calculate the exact location from a given city.	https://medium.com/analytics-vidhya/how-to-generate-lat-and-long-coordinates-of-city-without-using-apis-25ebabcaf1d5	Used this website to find out how to calculate latitude, longitude from inputting a city for our visualization.
12/11/2022	We were having issues with our Spotify API because it was only working with one account.	Piazza Post https://piazza.com/class/l754aa3ib8z43s/post/492	We found that Spotify was limiting our data.
12/12/22	We were having issues using geopandas to create data from a world map	https://coderzcolumn.com/tutorials/data-science/plotting-static-maps-with-geopandas-working-with-geospatial-data	Visualized the location of the top 25 artists next concert on a world map
12/12/22		https://365datascience.com/tutorials/python-tutorials/bar-chart-python-matplotlib/	Visualized the total streams of the top 25 artists