Goal

Our project aimed to analyze artist data from Spotify, focusing on comparing min max track song popularity to gauge if an artist's fame is due to a single hit or overall consistent performance. We designed visualizations showcasing the range of an artist's popularity, represented through their songs' minimum and maximum popularity levels. Another key objective was to create visual charts displaying the follower counts of each artist, specifically within the American market, to provide a comprehensive view of their popularity and reach.

Achievement

We achieved our project goals by utilizing three Spotify APIs, one for artist IDs, one for artist information, and one for track specifically targeting the American market. We analyzed and categorized artists based on their track's popularity and follower counts. The data was then visually represented in graphs to highlight the correlation between a song's popularity and its impact, offering insights into the dynamics of music popularity within the Spotify platform.

Problem statement

In our project, we encountered a challenge with the Spotify API regarding token scopes. These scopes dictate the level of access to specific data or actions, and selecting the appropriate scopes for our project requirements proved to be a complex task. Balancing the need for adequate access against the limitations imposed by these scopes required careful consideration and planning.

Challenge

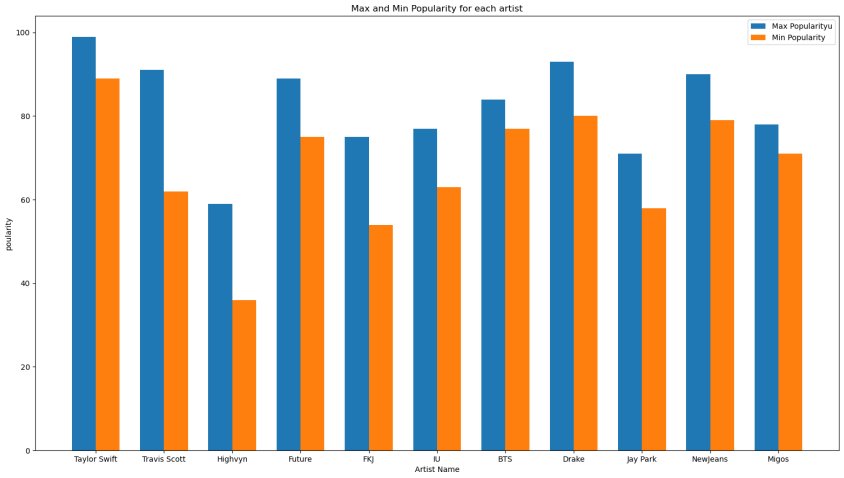
Scalability and Performance:

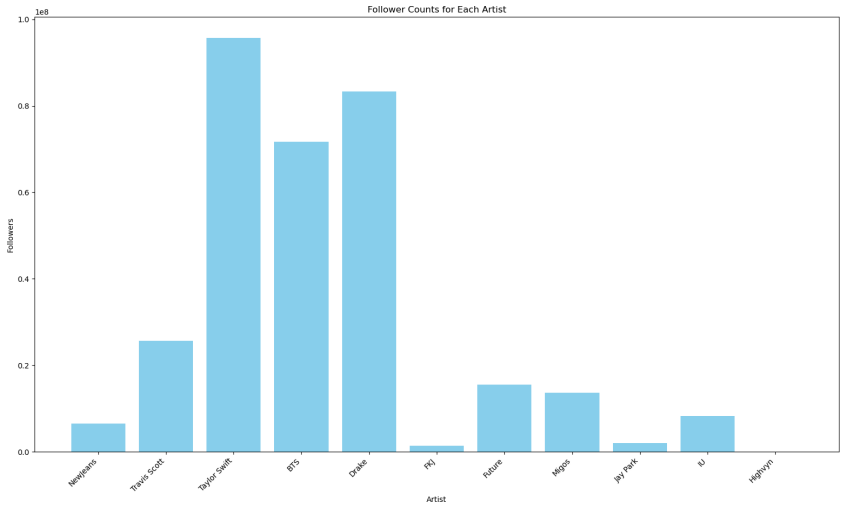
- Ensuring that database and script can handle large volumes of data efficiently. As dataset grows (more artists, more tracks), performance could become an issue.
- Optimizing SQL queries and script logic for better performance and scalability.

Authentication Management:

- Safely and efficiently handling authentication (like token generation in tokenizer function), especially considering tokens have an expiration and need to be refreshed or regenerated periodically.

Calculation



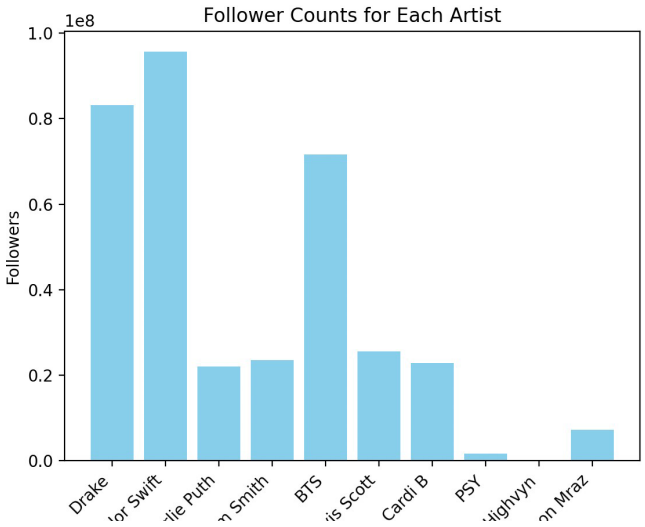


By calculating the difference between maximum and minimum popularity track, we can get an insight whether the artists are consistent. By looking at these graphs, we can calculate that such a popular artists like Taylor Swift, BTS, and Drake has 11 or less difference on maximum and minimum popularity.

Visualization

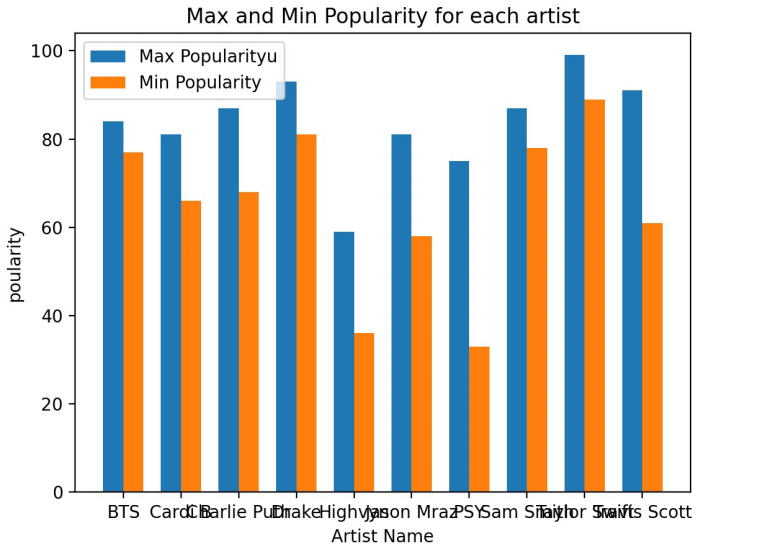
Follower Counts for Each Artist

The following graph illustrates Follower Counts for each artist on Spotify



Max and Min Popularity for each artist

The following graph illustrates the Maximum and Minimum Popularity for each artist on Spotify



Instruction for running code

- Open “spotify music” folder in VSCode
- Run the code from controller.py
- Select whether you want to get information on what’s automatically saved or new artists’ information by entering A as automatic or M as manual
- Graphs will show up in order.

Documentation

Controller.py

conn = sqlite3.connect("spotify.db")

Establish a connection to the 'spotify.db' SQLite database

cur = conn.cursor()

Create a cursor object to interact with the database

db.droptables(conn, cur)

Call a function 'droptables' from the 'db' module to drop existing tables in the database

db.createtables(conn, cur)

Call a function 'createtables' from the 'db' module to create new tables in the database

headers = accesstoken.tokenizer()

Generate headers by calling the 'tokenizer' method

def manual():

Define a function named 'manual' for manual data entry

name = input("Add the artist you want to: ")

Prompt the user to input an artist's name

while len(name) > 0:

Loop as long as the input name is not empty

id = spotifyApi.getArtistId(headers, name)

Get the artist's ID from Spotify API using the input name

info = spotifyApi.getArtist(headers, id)

Retrieve artist's information using the Spotify API

db.insert_artist(conn, cur, info)

Insert the artist's information into the database

info = spotifyApi.getTrack(headers, id)

Retrieve the tracks of the artist using the Spotify API

db.insert_track(conn, cur, info)

Insert the track information into the database

name = input("Add the artist you want to: ")

Prompt for the next artist's name

def automatic():

Define a function named 'automatic' for automatic data entry

names = ["New Janes", "Travis Scott", "Taylor Swift", "BTS", "Drake", "FKJ",

"Future", "Migos", "Jay Park", "IU", “Highvyn"]

List of predefined artist names

for name in names:

Iterate through each artist in the list

id = spotifyApi.getArtistId(headers, name)

Get the artist's ID from Spotify API using the artist's name

time.sleep(0.1)

Pause execution for 0.1 seconds to avoid hitting API rate limits

info = spotifyApi.getArtist(headers, id)

Retrieve artist's information using the Spotify API

db.insert_artist(conn, cur, info)

Insert the artist's information into the database

info = spotifyApi.getTrack(headers, id)

Retrieve the tracks of the artist using the Spotify API

db.insert_track(conn, cur, info)

Insert the track information into the database

time.sleep(0.1)

Pause execution for 0.1 seconds to avoid hitting API rate limits

user_in = input("A for Automatic, M for Manual")

Prompt the user to choose between Automatic and Manual modes

if user_in == "A":

automatic()

else:

manual()

Execute the 'automatic' function if the user inputs 'A', otherwise execute the 'manual' function

analysis.max_min_popularity(conn, cur)

Call a function 'max_min_popularity' from the 'analysis' module to analyze maximum and minimum popularity

analysis.followers(conn, cur)

Call a function 'followers' from the 'analysis' module to analyze followers data

spotifyApi.py

def getArtistId(headers, name):

Define a function to get the Spotify artist ID given the artist's name

url = "https://api.spotify.com/v1/search"

Spotify API endpoint for searching

param = {

```
        "q" : name,
        "type" : "artist",
        "limit" : 1
    }
# Parameters for the API call: search query, type of search (artist), and limit to 1 result
res = requests.get(url, headers = headers, params = param)
# Perform the API request with the given URL, headers, and parameters
data = res.json()
# Parse the response as JSON
return data['artists']['items'][0]['id']
# Return the ID of the first artist found
def getArtist(headers, id):
# Define a function to get detailed information about an artist using their Spotify ID
url = "https://api.spotify.com/v1/artists/{}".format(id)
 # Spotify API endpoint for fetching artist details, formatted with the artist's ID
res = requests.get(url, headers= headers)
# Perform the API request with the given URL and headers
data = res.json()
# Parse the response as JSON
name = data['name']
followers = data['followers']['total']
popularity = data['popularity']
```

```
genres = data['genres'][0]
# Extract the artist's name, follower count, popularity, and first genre
info = (id, name, followers, popularity, genres)
# Combine these details into a tuple
return info
# Return the tuple containing the artist's details
def getTrack(headers, id):
# Define a function to get the top tracks of an artist using their Spotify ID
url = "https://api.spotify.com/v1/artists/{}/top-tracks".format(id)
# Spotify API endpoint for fetching top tracks of an artist, formatted with the artist's ID
    param = {
        "market" : "ES"
    }
# Parameters for the API call: specify the market
res = requests.get(url, headers= headers, params = param)
# Perform the API request with the given URL, headers, and parameters
data = res.json()
# Parse the response as JSON
 store = []
# Initialize a list to store track details
    for track in data['tracks']:
        store.append((id, track['name'], track['popularity']))
```

Append a tuple with the artist ID, track name, and track popularity to the list

return store

Return the list of tuples containing track details

analysis.py

def max_min_popularity(conn, cur):

Define a function to calculate and plot the maximum and minimum popularity of tracks for each artist

labels = []

Initialize a list to store artist names

max_popularity = []

Initialize a list to store maximum popularity values

min_popularity = []

Initialize a list to store minimum popularity values

cur.execute("SELECT track_name, name, min(track_popularity) FROM artist JOIN track USING(artist_id) group by artist_id")

Execute SQL query to find the minimum track popularity for each artist

for data in cur.fetchall():

labels.append(data[1])

Append the artist name to labels list

min_popularity.append(data[2])

Append the minimum popularity to the min_popularity list

cur.execute("SELECT track_name, name, max(track_popularity) FROM artist JOIN

track USING(artist_id) group by artist_id")

Execute SQL query to find the maximum track popularity for each artist

for data in cur.fetchall():

max_popularity.append(data[2])

Append the maximum popularity to the max_popularity list

bar_width = 0.35

Set the width of each bar in the bar chart

fig, ax = plt.subplots()

Create a matplotlib figure and axes for plotting

bar1 = ax.bar(labels, max_popularity, bar_width, label='Max Popularity')

Create a bar plot for the maximum popularity

bar2 = ax.bar(np.arange(len(labels)) + bar_width, min_popularity, bar_width, label='Min

Popularity')

Create a bar plot for the minimum popularity

ax.set_xlabel('Artist Name')

Set x-axis label to 'Artist Name'

ax.set_ylabel('Popularity')

Set y-axis label to 'Popularity'

ax.set_title('Max and Min Popularity for each artist')

Set title of the plot

ax.set_xticks(np.arange(len(labels)) + bar_width / 2)

Set the x-ticks to be in the middle of the grouped bars

ax.set_xticklabels(labels)

Set the labels for the x-ticks

ax.legend()

Add a legend to the plot

plt.show()

Display the plot

def followers(conn, cur):

Define a function to plot the number of followers for each artist

cur.execute("SELECT name, followers FROM artist")

Execute SQL query to retrieve the name and number of followers for each artist

artists = []

Initialize a list to store artist names

followers = []

Initialize a list to store follower counts

for data in cur.fetchall():

artists.append(data[0])

Append the artist name to the artists list

followers.append(data[1])

Append the follower count to the followers list

fig, ax = plt.subplots()

Create a matplotlib figure and axes for plotting

ax.bar(artists, followers, color='skyblue')

Create a bar plot for the number of followers

ax.set_xlabel('Artist')

Set x-axis label to 'Artist'

ax.set_ylabel('Followers')

Set y-axis label to 'Followers'

ax.set_title('Follower Counts for Each Artist')

Set title of the plot

plt.xticks(rotation=45, ha='right')

Rotate the x-axis labels for better readability

plt.show()

Display the plot

db.py

def createtables(conn, cur):

Define a function to create database tables

cur.execute("CREATE TABLE IF NOT EXISTS artist (artist_id TEXT PRIMARY KEY,

name TEXT, followers INT, popularity INT, genres TEXT)")

Create the 'artist' table with artist_id as PRIMARY KEY and other fields

cur.execute("CREATE TABLE IF NOT EXISTS track (artist_id TEXT, track_name

TEXT, track_popularity INT)")

Create the 'track' table with artist_id, track_name, and track_popularity fields

conn.commit()

Commit the changes to the database

def droptables(conn, cur):

Define a function to drop existing database tables

cur.execute("DROP TABLE IF EXISTS artist")

Drop the 'artist' table if it exists

cur.execute("DROP TABLE IF EXISTS track")

Drop the 'track' table if it exists

conn.commit()

Commit the changes to the database

def insert_artist(conn, cur, info):

Define a function to insert an artist's data into the artist table

cur.execute("INSERT OR IGNORE INTO artist VALUES(?,?,?,?,?)", info)

Insert the artist data into the artist table, ignore if the entry already exists

conn.commit()

Commit the changes to the database

def insert_track(conn, cur, info):

Define a function to insert track data into the track table

cur.executemany("INSERT OR IGNORE INTO track VALUES(?,?,?)", info)

Insert multiple track data entries into the track table, ignore if they already exist

conn.commit()

Commit the changes to the database

dbview.py

conn = sqlite3.connect("spotify.db")

Connect to the SQLite database 'spotify.db'

cur = conn.cursor()

Create a cursor object to interact with the database

cur.execute("SELECT * FROM artist")

Execute an SQL query to select all records from the 'artist' table

for data in cur.fetchall():

    print(data)

    # Fetch all the records from the 'artist' table and print each record

print()

print()

print()

Print blank lines for spacing

cur.execute("SELECT * FROM track")

Execute an SQL query to select all records from the 'track' table

for data in cur.fetchall():

print(data)

Fetch all the records from the 'track' table and print each record

accesstoken.py

def tokenizer():

Define a function to generate the authorization headers required for Spotify API requests

client_id = 'd9cc6771d4504cb0ae4acd085c1bc13e'

Spotify API client ID

client_secret = '1c6d19bbe9dd45c98a91f2d5389f518a'

Spotify API client secret

token = get_access_token(client_id, client_secret)

Obtain an access token using the client ID and client secret

headers = {'Authorization': 'Bearer {}'.format(token)}

Format the headers with the obtained access token

return headers

Return the headers for use in API requests

def get_access_token(client_id, client_secret):

Define a function to get an access token from Spotify API

token_url = 'https://accounts.spotify.com/api/token'

Spotify API token URL

auth = (client_id, client_secret)

Authentication tuple with client ID and client secret

data = {'grant_type': 'client_credentials'}

Data payload with the grant type set to 'client_credentials'

response = requests.post(token_url, auth=auth, data=data)

Make a POST request to the token URL with authentication and data

token = response.json().get('access_token')

Extract the 'access_token' from the response JSON

return token

Return the access token