Platforms Vs. Stream - Alyssa Rodriguez & Madison Hall

Overall goal for the project & What we achieved

The goal of our project was to assess whether there was a correlation between an artist's top songs and the number of platforms their songs are available on. In conjunction, we specifically wanted to assess Spotify's power over the music industry. Since Spotify is the world's most popular audio streaming platform with more than 574 million users and catering to 180 markets, we wanted to assess whether an artist has the ability to hone in this specific platform instead of juggling multiple that do not garner the same attention. Thus, we utilized Songlink API, which gave us the number of platforms a song is available on. In addition, we used the Spotify API to gather the artist's top 10 songs and through songlink we were able to view the number of platforms the song is available on. Similarly, the Spotify API allowed us the option to view the popularity of the song which is scored on a scale of 1 to 100. Utilizing the popularity score to our advantage, we wanted to see if there was a correlation between high popularity scores and the probability of that song ending up on an artist's setlist on tour. Thus, we wanted to see the average score of a song on an artist's setlist to see whether popularity score influences an artist's decision to play the song. We did this using Setlistfm API, which gave us the setlist of an artist of our choice, that we accessed using the Spotify API, and it returned the setlist from each venue they have performed in.

Challenges we faced

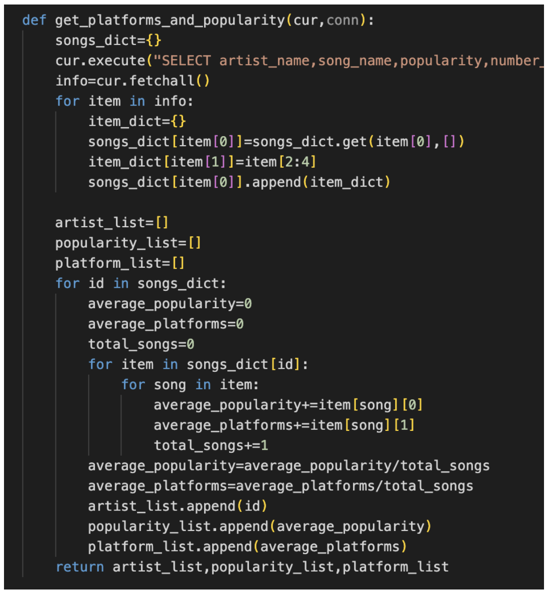
The problem we encountered was giving the tables an identical key to access them. To avoid duplicate string data, we assigned each song and artist a key to be used in the tables. It took time to make this system work as we intended, but was ultimately very useful. It was also a unique challenge learning to use each individual API and understanding how to best gather and store the data they provided.

In addition, since we decided to access every artist's top 10 songs, sometimes these songs came from the same album which inherently created duplicate text data. However, after the grading session and realizing our mistake, we decided to delete this column from the database table, spotify_data, for the better. Similarly, we encountered this same issue within our Setlistfm API, as there were duplicate venues. Yet, this column was important for our project so instead of deleting it we gave duplicate venues a venue ID.

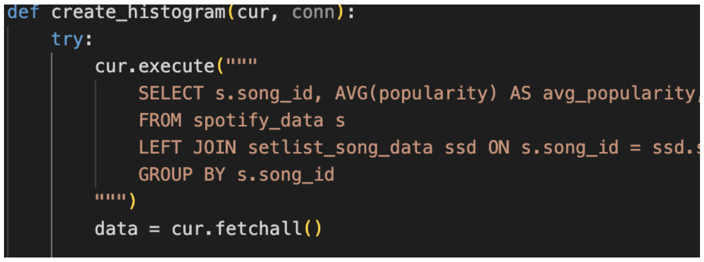
The last issue we faced was how to effectively display data within our visualization. Since we are working with so much data, the x and y axis quickly became clustered with too many output values. However, by creating ranges and using scatter plots we were able to effectively display our data aesthetically.

Calculation from the database

We calculated the averages for both song popularity per artist and how frequently songs appeared on different platforms per artist. This is also shown in a visualization.

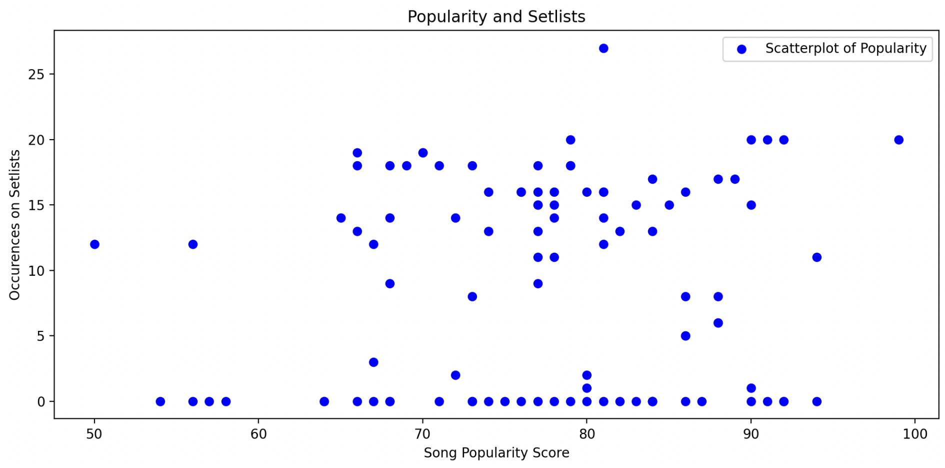


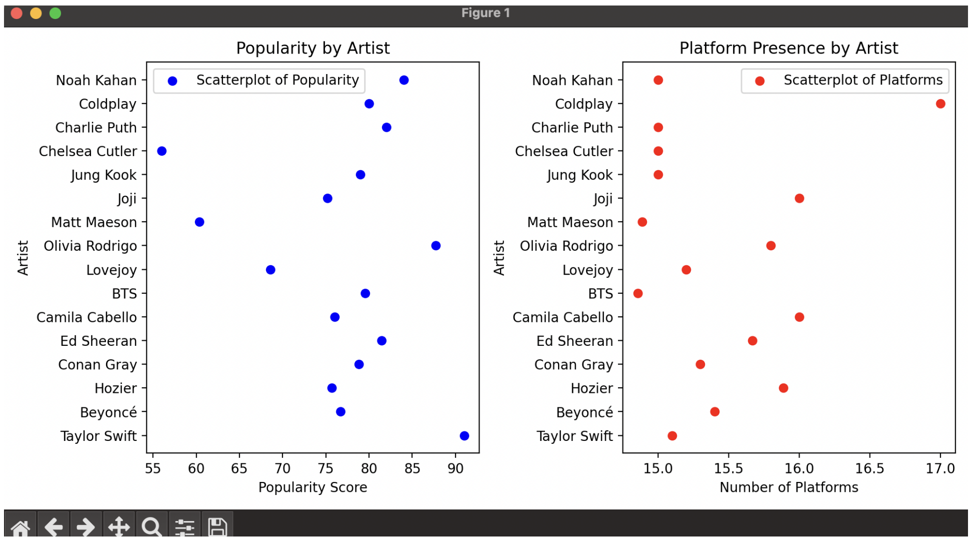
```python
def get_platforms_and_popularity(cur,conn):
    songs_dict={}
    cur.execute("SELECT artist_name,song_name,popularity,number_
    info=cur.fetchall()
    for item in info:
        item_dict={}
        songs_dict[item[0]]=songs_dict.get(item[0],[])
        item_dict[item[1]]=item[2:4]
        songs_dict[item[0]].append(item_dict)

    artist_list=[]
    popularity_list=[]
    platform_list=[]
    for id in songs_dict:
        average_popularity=0
        average_platforms=0
        total_songs=0
        for item in songs_dict[id]:
            for song in item:
                average_popularity+=item[song][0]
                average_platforms+=item[song][1]
                total_songs+=1
        average_popularity=average_popularity/total_songs
        average_platforms=average_platforms/total_songs
        artist_list.append(id)
        popularity_list.append(average_popularity)
        platform_list.append(average_platforms)
    return artist_list,popularity_list,platform_list
```

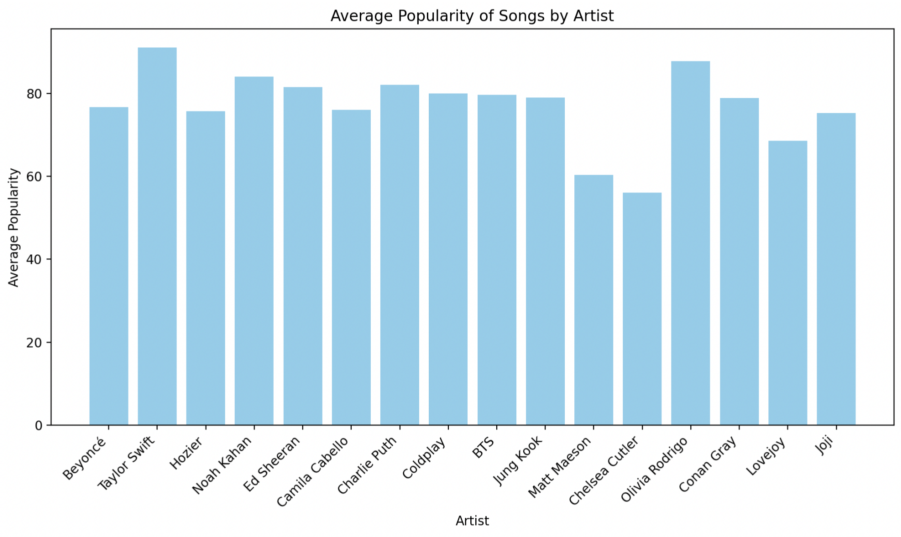


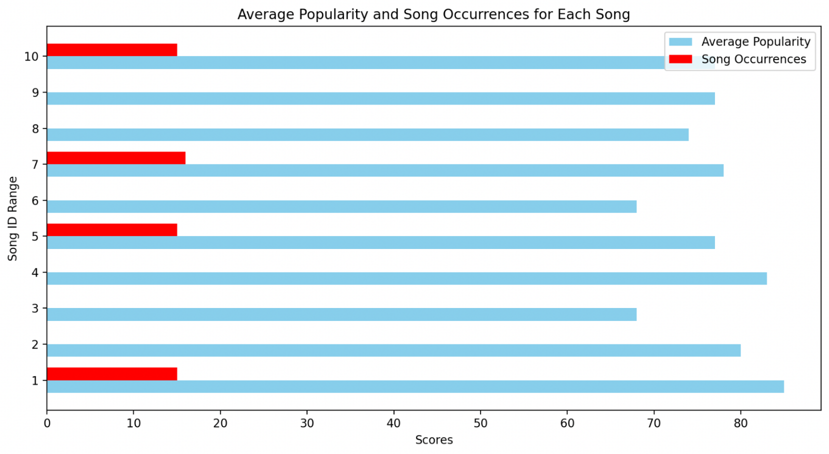
```python
def create_histogram(cur, conn):
    try:
        cur.execute("""
            SELECT s.song_id, AVG(popularity) AS avg_popularity,
            FROM spotify_data s
            LEFT JOIN setlist_song_data ssd ON s.song_id = ssd.s
            GROUP BY s.song_id
        """)
        data = cur.fetchall()
```

Screenshots of visualizations









Instructions for running our code

Must import spotipy, import request, and from spotipy.oauth2 import SpotifyClientCredentials. To utilize the Spotify API you have to create an account and list a project in order to receive your clientID and clientSecret which gives you access to the data. Add artists to a list in the main.py file (there are currently 10 already in the list, which adds their 10 top songs at a time: 100 values). For each artist, run the main file once to add all information to the database. To view visualizations, run each visualization file 1-4.

Documentation for each function we wrote

- spotifydata.py
  - get_spotify_token():
    - Retrieved an access token from the Spotify API using client credentials
    - Parameters: None
    - Returns: token(str): the access token obtained from the Spotify API
  - get_spotify_auth_header(token):
    - Returns: the authorization header for Spotify API
    - Parameter: token (str)- the access token obtained from Spotify
    - Returns: dict: the authorization header in the format {"Authorization": "Bearer <token>"}.
  - get_spotify_artist_search(token, artist_name):
    - Searches for an artist on an artist on Spotify using the provided access token and artist name
    - Parameters:
      - token(str): the access token for Spotify API authentication
      - artist_name(str): the name of the artist to search for
    - Returns: artist_info(dict): a dictionary containing information about the artist, such as their name, ID, and external URL's
    - Raises: None
  - spotify_get_songs_by_artist(token, artist_id):
    - Retrieves the top tracks of a given artist from Spotify API
    - Parameters:
      - token(str): the access token for Spotify API authentication
      - artist_id(dict): the artist ID obtained from Spotify API
    - Returns:
      - List: a list of top tracks of the artist, each represented as a dictionary
  - setUpDatabase(db_name):
    - Sets up a connection to the SQLite database
    - Parameters:
      - db_name (str): the name of the database file
    - Returns:
      - cur (sqlite3.cursor): the cursor object for executing the SQL statements
      - conn (sqlite3.connection): the connection object for interacting with the database

- store_spotify_data(track, cur, conn):
  - Stores spotify data for a track in the database
  - Parameters:
    - track (dict): a dictionary containing information about the track
    - cur (cursor): the database cursor object
    - conn (connection): the database connection object
  - Returns:
    - None
  - Raises:
    - Exception: if there is an error storing the Spotify data

- setlistdata.py
  - get_setlists_for_artists(api_key, artist):
    - Retrieved setlists for a given artist using the Setlist.fm API
    - Parameters:
      - api_key(str): the API key for accessing the Setlist.fm API
      - artist(str): the name of the artist to retrieve setlists for
    - Returns:
      - List: a list of setlists for the specified artist
  - store_setlists_for_artists(data, cur, conn):
    - Store setlist data for artists in the database
    - Args:
      - data(list): a list of setlist data
      - cur (cursor): the database cursor
      - conn (connection): the database connection
    - Returns: None
  - store_tours_for_artists(data,cur,conn):
    - Stores tour data for artists in a database table
    - Parameters:
      - data(list): a list of tour data for artists
      - cur(cursor): the database cursor object
      - conn(connection): the database connection object
    - Returns: none
- songlinkdata.py
  - get_songlink_data(song_url, title):
    - Retrieves song link data from given song URL
    - Parameters:
      - song_url(str): the URL of the song
      - title(str): the title of the song
    - Returns:

            - Song_dict (dict): a dictionary containing the song title as the key and a tuple of platforms and the number of platforms as the value
  - store_songlink_data(data,cur,conn):
    - Stores songlink data in the database
    - Parameters:
      - Data (list): a list of dictionaries containing songlink data
      - Cur (cursor): the database cursor object
      - conn(connection): the database connection object
    - Returns
      - None
- main.py
  - get_setlists_for_artists(api_key, artist):
    - Retrieves setlists for a given artist using the Setlist.fm API
    - Parameters:
      - api_key(str): the api key for accessing the Setlist.fm API
      - artist(str): the name of the artist to retrieve setlists for
    - Returns:
      - List: a list of setlists for the given artist
  - store_setlists_for_aritsts(data,track_list,cur,conn):
    - Stores setlist song data for artists in a database
    - Parameters:
      - data(list): a list of dictionaries containing setlist data for artists
      - track_list(list):a list of track titles to be considered
      - cur(cursor): the database cursor object
      - conn(connection): the database connection object
    - Returns
      - None
  - store_tours_for_artists(data,cur,conn):
    - Store tour data for artists in a database table
    - Parameters:
      - data(list): a list of tour data for artists
      - cur (cursor): the database cursor object
      - conn(connection): the database connection object
    - Returns:
      - None
- visualization1.py
  - get_platforms_and_popularity(cur,conn):
    - Retrieves the average popularity and number of platforms for each artist's songs from the database
    - Parameters:

            - cur(cursor object): the cursor object to execute SQL queries
            - conn(connection object): the connection object to the database
        - Returns:
            - artist_list(list): a list of artist names
            - popularity_list(list): a list of average popularity values for each artist
            - platform_list(list): a list of average number of platforms for each artist
    - visualize_platforms_and_popularity(calculation):
        - Visualizes the platforms and popularity of artists using scatter plots
            - Parameters: calculation (tuple): a tuple containing three lists - popularity scores, number of platforms, and artist names
            - Returns: None
- visualization2.py
    - get_song_platform_data(cur):
        - Retrieves the song name and the number of platforms on which the song is available
        - Parameters:
            - cur(cursor): the database cursor object
        - Returns
            - List: a list of tuples containing the song name and the number of platforms
    - plot_bar_chart(data):
        - Plots a bar chart to visualize the number of platforms per song
        - Parameters:
            - Data (list): a list of tuples containing songs names and corresponding platform counts
            - Returns: None
        -
- visualization3.py
    - create_histogram(cur, conn):
        - Creates a histogram of the average popularity of songs by artists
        - Parameters:
            - cur(sqlite3.Cursor): the cursor object for executing SQLite queries
            - conn(sqlite3.Connection): the connection object for the SQLite database
        - Returns: None
- visualization4.py
    - get_song_and_setlist(con,curr)

        - Selects values from tables to get song popularity and occurrences on setlists.
        - Parameters:
            - curr:database cursor object
            - conn:database connection object
        - Returns:
            - Popularity_list, occurrences: tuples containing values for popularity and setlist occurrences
    - visualize_song_and_popularity(calculation)
        - Creates a scatterplot of popularity and setlist occurrences
        - Parameters:
            - calculation: Tuples returned in first function
        - Returns:
            - None

Resources we used

Date	**Issue Description**	**Location**	**Result**
12/1/2023	Writing documentation for functions for Spotify API	Github copilot	Yes, it helped me outline how to organize + create functions on my own
12/2/2023	Accessing Spotify API, table was being outputted but no data was being shown	Office hours	Issue was not resolved
12/2/2023	Accessing Spotify API data, specifically artist top 10 songs	https://youtu.be/WAmEZBEeNmg?feature=shared	Yes, it was very helpful and walked me through how to create an access token and gather the information I needed
12/2/2023	Spotify API data table	Chatgpt	Yes, it helped me write and access popularity score from the Spotify database and write it into the database table
12/10/2023	Debugging	Github copilot	Yes, it taught my how

	visualization3		to use AVG() within cur.execute
12/10/2023	Debugging visualization2	Github copilot	Yes, was not appending the correct item to list (i.e. indexing 1 instead of 2)
12/12/2023	Debugging visualization4	Chatgpt	Yes, it helped give me ranges on the x and y axis since there were so many data point to graph

Grade Change Notes

- Visualization 4 axis (gave the y and x ranges)
- Duplicate string in albums column in spotify_data table (deleted the table)
- Duplicate venues in setlist table (gave venues an ID)
- + 2 bonus visualizations
- + 1 bonus API