1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

For our project, we planned to integrate with two major APIs: Spotify's Web API and Kroger's Location API. Our primary goal with Spotify was to explore the relationship between different music genres and popular playlists created by Spotify by fetching data on featured playlists and their respective tracks, including artist names. For Kroger, we aimed to analyze the geographical distribution and the presence of Kroger stores across various states to potentially understand market saturation and regional preferences.

Our data gathering was focused on two fronts. From Spotify, we intended to pull information about playlists and the tracks contained within them. This included tracks and artists' names. From Kroger, our objective was to retrieve details about store locations, specifically the number of stores per state. Since we can't tell the number of stores per state directly, our plan was to gather the information about the store, with unique store ids, names, and states.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

We were able to achieve our plan, successfully pulling and storing the planned data from both APIs. Our approach ensured we captured a dataset from Spotify that included 20 of the featured playlists, as well as another table that included the tracks in all of these playtlists. Similarly, the Kroger data we compiled allowed us to map out and analyze the distribution of stores, providing valuable insights into the company's market presence across different states.

By the project's conclusion, we had managed to store 500 tracks linked to specific playlists from Spotify. On the Kroger front, we gathered and stored all of the stores that are Kroger, not including subsidiaries, which totalled to 1311 stores.

3. The problems that you faced (10 points)

From our initial plan, we encountered problems with making sure our project would meet the rubric requirements. We had to adjust our endpoints and change some of our original plans to accommodate that.

Throughout the project, we encountered several issues with our code and retrieving the API. Our most difficult part of the project was actually retrieving Spotify's API. One of the primary issues we faced was related to accessing Spotify's API. The challenge was navigating the authentication process, and writing code to successfully access it.

4. The calculations from the data in the database (i.e. a screen shot) (10 points)

Kroger

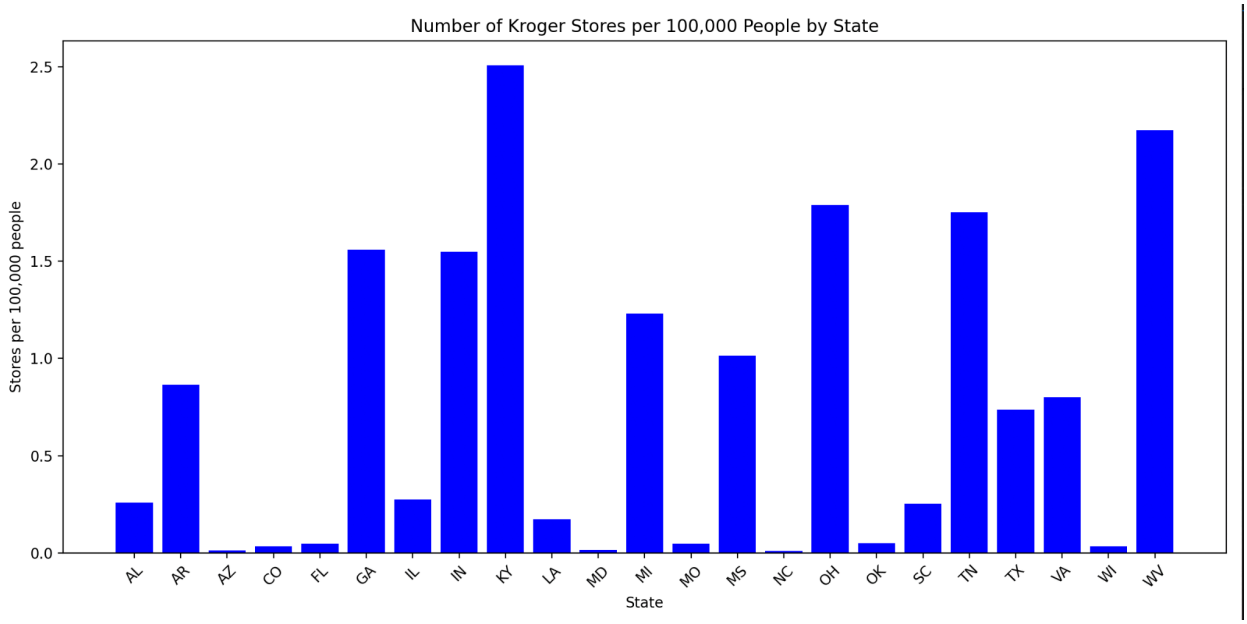
```
def count_stores_by_state():
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute('SELECT state, COUNT(*) FROM kroger_stores GROUP BY state')
    results = c.fetchall()
    store_per_capita = {}
    for state, count in results:
        if state in POPULATION_DATA:
            # Now using 100000 as the multiplier for per 100,000 people calculation
            store_per_capita[state] = (count / POPULATION_DATA[state]) * 100000
    conn.close()
    return store_per_capita
```

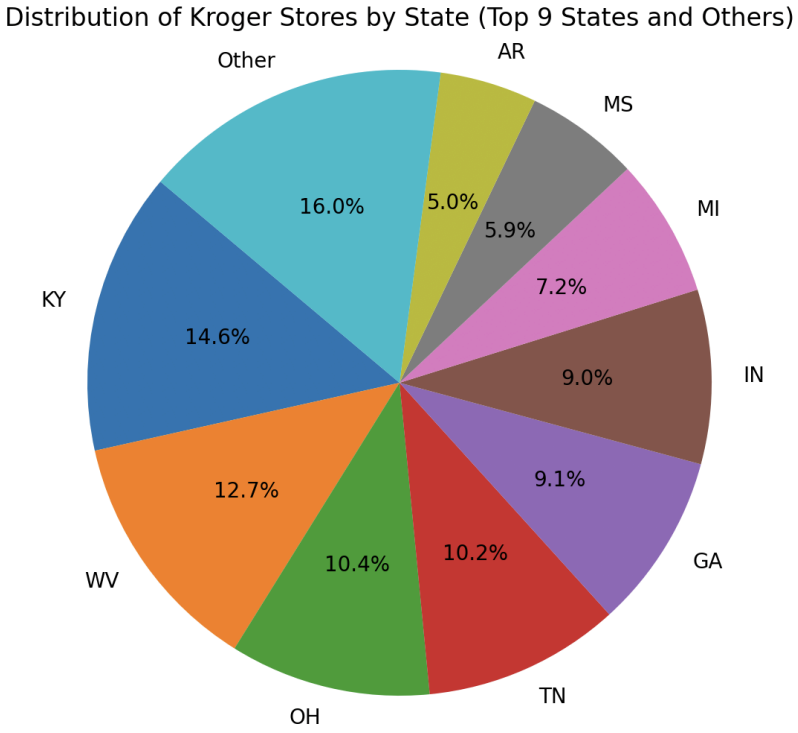
Spotify

```
def fetch_playlist_data():
    """ Fetches the count of unique artists and total tracks per playlist from the database. """
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    query = '''
    SELECT playlist_name, COUNT(DISTINCT artist) AS unique_artists_count, COUNT(*) AS total_tracks
    FROM playlist_tracks
    GROUP BY playlist_name;
    '''
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()
    conn.close()
    return results
```

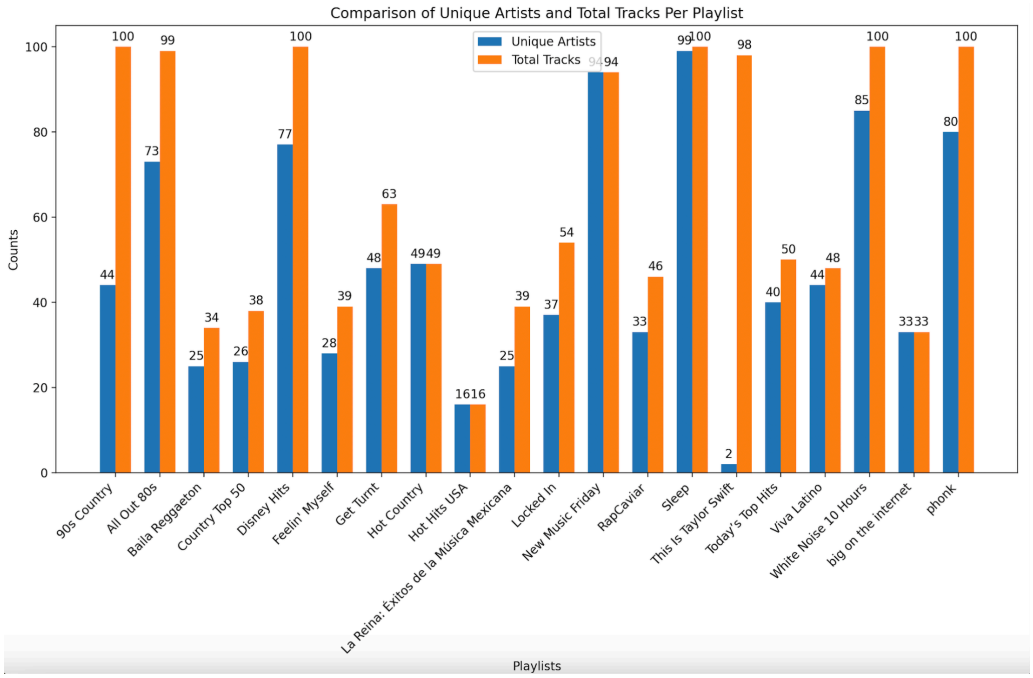
5. The visualization that you created (i.e. screen shot or image file) (10 points)

Kroger

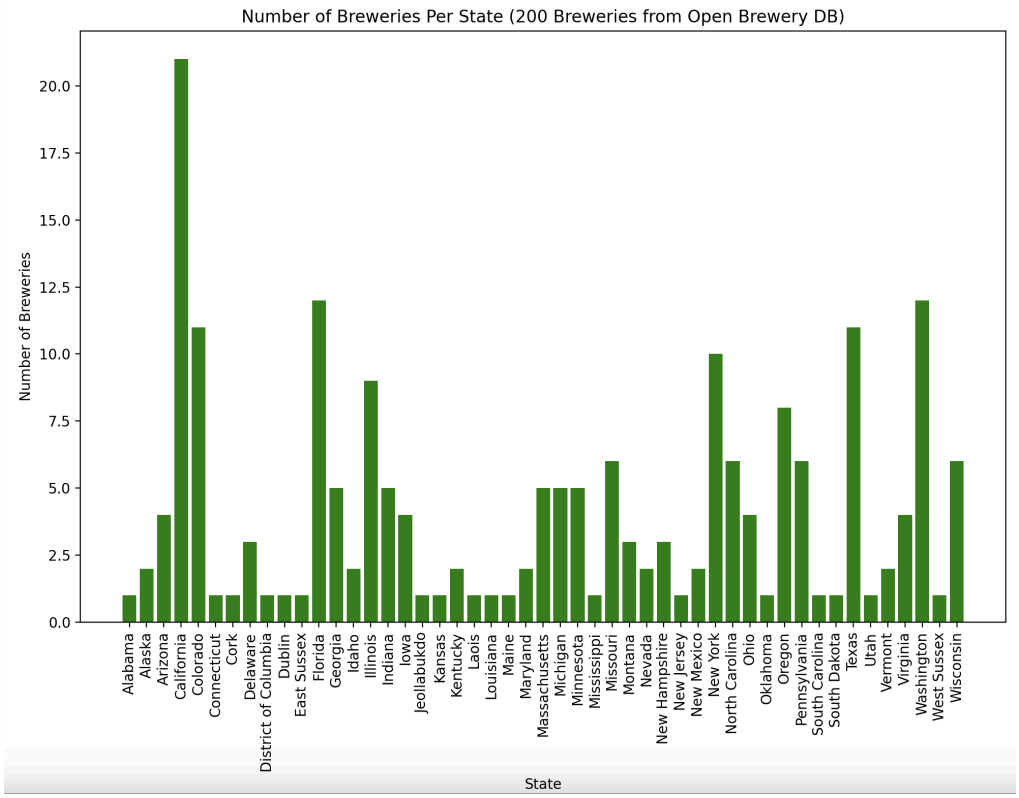

Number of Kroger Stores per 100,000 People by State
Stores per 100,000 people
0.0
0.5
1.0
1.5
2.0
2.5
AL
AR
AZ
CO
FL
GA
IL
IN
KY
LA
MD
MI
MO
MS
NC
OH
OK
SC
TN
TX
VA
WI
WV
State


Distribution of Kroger Stores by State (Top 9 States and Others)
Other
AR
MS
MI
IN
GA
TN
OH
WV
KY
16.0%
5.0%
5.9%
7.2%
9.0%
9.1%
10.2%
10.4%
12.7%
14.6%

Spotify



Extra Credit (OpenBreweryDB):



6. Instructions for running your code (10 points)

To get the Kroger data and Visualization:

1. Run “kroger_gathering_data.py” to get data in the table.
2. There is a tracker in the terminal for the number of items inserted, each run is 25 items. Run up to 45 times (1125 items) to get all the data
3. To get the visualization, run “kroger_analysis.py”

To get the Spotify data and Visualization:

1. Run “spotify_gathering_data.py” to get data in the table
2. There is a tracker in the terminal for the number of items inserted, each run is 25 items. Run up to 20 times (500 items) to get all the data
3. Run “spotfiy_db_join.py” for the table with the join up to 20 times (500 items)
4. Run “spotify_calculations.py” to get visualization

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

kroger_gathering_data.py:

Function: get_kroger_token

This function is responsible for authenticating with the Kroger API. It sends a POST request to Kroger's OAuth 2.0 token endpoint with the client ID and client secret. If the request is successful, it extracts and returns the access token from the response JSON. If unsuccessful, it returns None. This token is crucial for authenticating subsequent requests to the Kroger API.

Function: get_kroger_data

After obtaining an access token through get_kroger_token, this function makes a GET request to the Kroger API to fetch data about Kroger store locations, specifically querying for stores under the "Kroger" chain. It sets the appropriate authorization headers to include the bearer token. If the request succeeds, it returns the list of store data provided in the API response; otherwise, it returns an empty list. This data includes store identifiers, names, and the states they are located in.

Function: setup_database

This function sets up the necessary infrastructure for data storage by connecting to a SQLite database and creating two tables if they do not already exist. The first table, kroger_stores, stores the Kroger store data, and the second table, kroger_insert_tracker, is used to track the progress of data insertion, ensuring that each batch of data insertions can be resumed if interrupted. It ensures that the database schema is prepared before any data insertion occurs.

Function: insert_kroger_data

This function inserts a batch of store data into the SQLite database. It first retrieves the last index of data insertion from the kroger_insert_tracker to ensure continuity and avoid duplication. It then proceeds to insert up to 25 store records, starting from this index. This function checks each store record against existing entries in the database to prevent duplicate entries. After inserting the batch, it updates the tracker with the new index position. This function is designed to run multiple times until all store data is inserted, facilitating the management of large datasets by breaking down the insertion process into manageable batches.

kroger_analysis.py:
Function: count_stores_by_state()

Calculating the number of Kroger stores per 100,000 people in each state. It does so by executing a SQL query that counts the number of stores grouped by state from a SQLite database. It then computes the per capita store count using predefined population data. The output is a dictionary where each key is a state's abbreviation and the value is the calculated number of stores per 100,000 people.

Function: visualize_data(data)

This function takes the output from count_stores_by_state() and creates a visual representation. It plots the number of Kroger stores per 100,000 people for each state on a bar chart, enhancing the data's readability and making regional comparisons easier. The chart includes labels for states on the x-axis and the number of stores on the y-axis, complete with a title and a blue color scheme for the bars.

spotify_gathering_data.py:

Function: authenticate_spotify

This function does not require any input parameters. It utilizes the client credentials (CLIENT_ID and CLIENT_SECRET) defined in the global configuration to request an OAuth token from Spotify's API. It constructs a POST request to the Spotify authentication endpoint and sends the client credentials in a Basic Auth header. If the authentication is successful, it returns the access token as a string, which is necessary for subsequent API requests. If the request fails due to an error (e.g., incorrect credentials or network issues), it returns None.

Function: get_featured_playlists
This function takes a single parameter, access_token, which is used to authenticate requests to Spotify's Web API. It sends a GET request to the 'featured-playlists' endpoint to fetch a list of Spotify's featured playlists. If successful, the function parses and returns a list of playlist items

from the JSON response. In case of an error (such as an expired or invalid token), it logs the error message and returns an empty list.

Function: get_playlist_tracks
Accepting access_token and playlist_id as inputs, this function requests the tracks within a specific playlist from Spotify. It constructs the request URL dynamically based on the playlist ID and sends a GET request with the access token for authentication. The function returns a list of track objects if the request is successful. Each track object includes essential information such as track ID, name, and artist details. If the request fails (e.g., due to invalid parameters or API limits), it outputs an error message and returns an empty list.

Function: setup_database
This function sets up the SQLite database for the project. It does not require any input and operates solely within the database path (DB_PATH). The function creates two tables: 'playlists' and 'tracks', using SQL commands to ensure they exist before any data insertion occurs. This setup process does not return any data but ensures the database schema is prepared for storing playlist and track information.

Function(s): insert_playlists and insert_tracks: These functions are responsible for inserting data into the database. insert_playlists takes access_token to fetch playlists and stores them in the 'playlists' table. It does not enforce any insertion limits and logs successful insertions. On the other hand, insert_tracks also starts by fetching playlists but focuses on inserting tracks associated with each playlist. It inserts no more than 25 items per run. It updates a tracking table to remember the last position of inserted tracks. The function outputs the count of tracks inserted during the current run and updates the database accordingly.

spotify_db_join:

Function: setup_database

Establishes the necessary database structure for storing playlist track data and tracking insertions. It creates two tables: playlist_tracks, which stores the name of the playlist, track name, and artist; and playlist_insert_tracker, which is used to track the index of the last playlist processed to ensure only a subset of data is handled per execution. This function does not accept any inputs and does not return any outputs, but it ensures the database is correctly set up for data insertion and tracking.

Function: execute_join_and_store_results:

Performs a database join operation between the playlists and tracks tables to fetch combined data, and then stores this data into the playlist_tracks table. It manages batch processing by inserting only a portion of the results at a time, specifically up to 25 rows per run, based on the last index recorded in the playlist_insert_tracker table. The function adjusts the tracking after

each execution to continue from where it left off in the next run, ensuring that data insertion is manageable and controlled. The function reports the number of rows inserted in each run and updates the tracking index, but it does not return any data.

spotifiy_calculations.py

Function: fetch_playlist_data

This function in the spotify_calculations module retrieves statistical data from the playlist_tracks table, specifically counting the number of unique artists and total tracks per playlist. It executes a SQL query that groups entries by playlist name and calculates the counts, returning a list of tuples where each tuple contains the playlist name, the count of unique artists, and the total track count for that playlist. This function is designed to supply the data necessary for visual analysis and does not modify the database.

Function: plot_data

Takes the data retrieved by fetch_playlist_data and visualizes it using Matplotlib. It constructs a bar chart that displays the number of unique artists and total tracks for each playlist side by side, facilitating a comparative analysis. The function sets up the plot with appropriate labels, titles, and axes annotations to ensure clarity in the visual representation. Additionally, it includes interactive labels on the bars to display exact counts, enhancing the informative quality of the visualization. This function is purely for visual output and does not interact with the database or modify any data.

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	**Issue Description**	**Location of Resource**	**Result (did it solve the issue?**
April 14th	Struggled with implementing OAuth authentication for the Spotify API.	ChatGpt	Yes, the issue was resolved. ChatGPT helped debug the code for token uaw, leading to successful authentication with the Spotify API.
April 18th	Struggled with the structure of the	Office Hours	Yes, clarified what to do for that part. Just

	project. Didn't understand what the 25 item insertion limit meant meant		adjusting my insert data function to only insert 25 at a time
April 18th	Struggled with db_join, didn't know what to do or how to do it	Office Hours	IA helped explain what I needed to do. Helped clarify what a db join was.
April 19th	Struggled with matplotlib, didn't know how to run it	Lecture 21 Slides	Retried the installations, it worked
April 22nd	Spotify Database help, db join help	Office Hours	Wanted to make sure our db join aligned with the rubric

Extra Credit:

Additional API: OpenBreweryDB

brewery_api_extra_credit.py:

Using OpenBreweryDB, we were able to gather a list of 200 breweries in the United States with their details to store in our database. We calculated the number of breweries per state and created a visualization.

We created a total of 4 visualizations, 2 for extra credit.