Final Project Report

Zehuan Yu

Rommy Toufaily

In this project, we used <https://developer.spotify.com/documentation/web-api> and [https://www.billboard.com/charts/year-end/2020/hot-100-songs/#](https://www.billboard.com/charts/year-end/2020/hot-100-songs/) two websites to collect data of top 100 songs in 2020 and 2021. We develop a Python application that scrapes song data from Billboard's website, processes it, and stores it in a SQLite database for analysis. We gathered song rankings, names, and artists for the years 2020 and 2021 from Billboard. We also developed an application that sends requests to Spotify's API to retrieve the top tracks from the 2020 and 2021 playlists. It gathers track names, artists, their rankings, and generates specific IDs for each track to check if they appear in both years.

We successfully scraped song data from the specified Billboard URLs for both 2020 and 2021 using requests and BeautifulSoup for web scraping. We stored data in an SQLite database with appropriate tables and relationships, and retrieved song rankings, artists, and song names for 2020 and 2021, and associated artist IDs. We also successfully used Spotify’s API to retrieve the 2020 and 2021 playlist data information, which then we stored the track name, artist, rank, and id in a data base. We successfully used the information gathered from the API to check for artists in both years and compare them with artists unique to 2020 and 2021.

The most difficult part of this project is to satisfy the requirements of storing data to the database to 25 or fewer items in each execution. We needed to make a state table to record data processing progress within a database, keep track of where the data processing left off, and allow for the resumption of processing from this point in future executions. Moreover, we should ensure artist names are consistent to avoid duplicates in the database. We created a shared key – ArtistID to keep tracking artist names. Another issue encountered was with matplotlib we wanted our visuals to stand out, so we investigated some unique features matplotlib offered to do so.

In the data calculation process, an SQLite database is accessed to calculate statistics related to the number of unique songs per artist for each year (2020 and 2021). This is achieved by executing SQL queries that involve JOIN operations between the "BillboardTopSongs" tables and the "ArtistIDs" table. The result is a dictionary containing calculated statistics, with each year as a key and a list of artist-song count pairs as the associated data. This process provides valuable insights into how many songs each artist produced in the specified years.

A screenshot of a computer

Description automatically generated

In the data visualization process, Matplotlib is employed to create horizontal bar plots for each year. These plots represent the number of songs per artist, with artist names on the y-axis and the count of songs on the x-axis. To ensure readability, the figure size is adjusted, and the bar height is modified accordingly. The resulting visualizations provide a clear graphical representation of how song distribution varies among artists in 2020 and 2021, making it easier to identify artists with the most songs during these years.

A graph of a number of people

Description automatically generated with medium confidence A graph of a number of people

Description automatically generated with medium confidence

In the data visualization process, Matplotlib is employed to create a pie chart showing the percentages of artists uniquely in 2020, 2021, or both years. To ensure readability each statistic is assigned its own color and is labeled with its percentage directly on the slice of pie. We also added labels and a legend for clear identification. For fun we designed the chart to slightly resemble the Spotify logo featuring a circle shape and Spotify text in the center.

A screenshot of a computer

Description automatically generatedA pie chart with numbers and text

Description automatically generated

For Billboard:

Start with running the finalproject.py for at least 9 times. Databases will store 12 items for each execution, so you will get tables with 100 items each after executing 9 times. Run measurement.py to calculate number of unique songs per artist for each year and get the text of results. Finally, run Visualization.py to get 2 pictures.

For Spotify:

Start with running new.py, it needs the .env in the same directory. Running it once will create 2020.json, 2021.json, artist\_in\_both\_years.txt, and Data base. The data base will only add 24 artists (12 from each year), 25 tracks in 2020, and 25 tracks in 2021. You will need to run it more than 4 times, there should be 151 artists in artists, 157 tracks in 2020, and 147 tracks in 2021. The visuals should also pop up after running new.py.

For the Billboard code, we used 6 functions:

**1. get\_last\_processed\_index(db\_path):**

Description:

This function retrieves the last processed index from the State table in the specified SQLite database. It is used to determine where to resume data scraping.

Input:

-db\_path(str): The file path to the SQLite database.

Output:

-int: The last processed index from the database or 0 if no records exist.

**2. set\_last\_processed\_index(db\_path, index):**

Description:

This function sets the last processed index in the State table of the specified SQLite database. It is essential for tracking progress and ensuring data isn't reprocessed unnecessarily.

Input:

-db\_path(str): The file path to the SQLite database.

-index(int): The index to be set as the last processed index in the database.

Output:

-None

**3. create\_artist\_id\_table(db\_path):**

Description:

This function creates the ArtistIDs table in the specified SQLite database if it doesn't already exist. The ArtistIDs table is used to store artist names and their associated IDs, which is crucial for maintaining a normalized database structure.

Input:

-db\_path(str): The file path to the SQLite database.

Output:

-None

**4. get\_or\_create\_artist\_id(db\_path, artist\_name):**

Description:

This function retrieves or creates an artist ID for a given artist name in the specified SQLite database. It helps avoid duplicate entries for the same artist.

Input:

-db\_path(str): The file path to the SQLite database.

-artist\_name(str): The name of the artist.

Output:

-int: The artist ID.

**5. scrape\_songs(url, db\_path, start\_index=0):**

Description:

This function scrapes song data from a specified URL and stores it in the specified SQLite database. It collects information about song rankings, names, and artists, then associates them with artist IDs.

Input:

-url(str): The URL to scrape song data from.

-db\_path(str): The file path to the SQLite database.

-start\_index(int): The starting index for scraping (default is 0).

Output:

-list: A list of tuples containing song data.

**6. main(url\_2020, url\_2021):**

Description:

This main function orchestrates the scraping and storage of song data. It coordinates the creation of database tables, retrieval of the last processed index, scraping of song data for 2020 and 2021, and insertion of data into the database.

Input:

-url\_2020(str): URL for 2020 song data.

-url\_2021(str): URL for 2021 song data.

Output:

-None

For the Spotify API we used 11 functions:

### 1. get\_token()

* Input:
  + None
* Output:
  + token(str): Authentication token
* Description: This function retrieves an authentication token for the spotify API. Using the client id and secret it makes a post request to Sportify authentication url, returning an access token needed for further API requests.

### 2. get\_auth\_header(token)

* Input:
  + token(str): Authentication token
* Output:
  + Authorization header (dictionary): Necessary header needed for requests
* Description: This function creates the authorization header needed to make a spotify api request, it takes in the authentication token and outputs a dictionary with the correctly formatted request header.

### 3. get\_playlist(playlist\_id, token)

* Input:
  + playlist\_id (str): The id for a specific playlist
  + token(str): Authentication token
* Output:
  + Playlist data(json): The playlists data in json format
* Description: This function gets the playlist details of a specific spotify playlist using its id. It sends a get request passing in the playlists id and authentication token and returns that playlists data in json format.

### 4. get\_track\_info(playlist\_data)

* Input:
  + playlist\_data(json): The playlists data in json format
* Output:
  + List of tracks (list of dictionaries)
* Description: This function loops through the playlists data to extract the info about each of the tracks, it gets the name of the track and the artist's name of that trac, then returns a list of dictionaries with that information.

### 5. setup Database(db\_name)

* Input:
  + db\_name(str):  The name of the database
* Output:
  + Database cursor and connection
* Description: This function establishes a connection to a sqlite database with the specified name, returns the database cursor and connection objects.

### 6. create\_table(cur, conn)

* Input:
  + cur: the database cursor
  + conn: the database connection
* Output:
  + None
* Description: This function creates tables in the database for storing artists and track information. It created tables for artists, tracks in 2020, and tracks in 2021.

### 7. add\_artists(data, cur, conn)

* Input:
  + data (list of dictionaries): A list of track info each in a dictionary containing song name and artist
  + conn: the database connection
  + cur: the database cursor
* Output:
  + None
* Description: This function adds the artists name and specific artist id to the table Artists in the database.

### 8. add\_track(name, data, cur, conn)

* Input:
  + name(str): The name of the database tables to add tracks to
  + data (list of dictionaries): A list of track info each in a dictionary containing song name and artist
  + conn: the database connection
  + cur: the database cursor
* Output:
  + None
* Description: This function adds playlist information including song name, artist, rank, and their specific id from the Artists table into the specified database table.

### 9. artists\_in\_both\_years(cur)

* Input:
  + cur: the database cursor
* Output:
  + List of artists(list of tuples): List of artists names
* Description: This function retrieves a list of artists who appear in both 2020 and 2021 playlist tables in the database.

### 10. unique\_artists(cur, year)

* Input:
  + cur: the database cursor
  + year(int):The year to filter which artists to return
* Output:
  + List of artists(list of tuples): List of artists names
* Description: This function retrieves a list of artists who are unique to either 2020 or 2021 playlist tables in the database.

### 11. write\_artists(filename, artists)

* Input:
  + filename(str): The name of the file to write to
  + artists(list of tuples): List of artist names
* Output:
  + None
* Description: This function writes the names of artists to the specified file, making a newline for each  name.

|  |  |  |
| --- | --- | --- |
| Issue Description | Location Of Resource | Result |
| Circular shape for pie chart | <https://stackoverflow.com/questions/9215658/plot-a-circle-with-pyplot> stack overflow | Yes, it solved the issue |
| Push the legend into the corner | <https://stackoverflow.com/questions/25068384/bbox-to-anchor-and-loc-in-matplotlib> stack overflow | Yes, it solved the issue |
| Draw horizontal bar chart with python | <https://matplotlib.org/stable/gallery/statistics/barchart_demo.html#sphx-glr-gallery-statistics-barchart-demo-py> | Yes, it solved the issue |
| Scrap only 12 songs at a time | <https://www.geeksforgeeks.org/enumerate-in-python/> | Yes, it solved the issue |