Music Database

Report for Data Engineer Program Project

July 2024 Cohort

Written by

Duygu Dalman Ciplak

Hakan Dogan

Supervisor

Antoine Fradin

Cohort Leader

Vincent

Table of Contents

1. Introduction

According to Wikipedia [1], there are many online music databases, which probably contain millions of songs in total. In this project, we aim to create a database for songs, and try to include information (metadata) on essentially the title, the artist, genre, release year and other fields (where applicable). The lists for music databases and resources on Wikipedia are exhaustive; however, collecting consistent information has challenges such as access conditions to the API, difficulties encountered in Web-scraping (if this is the chosen method), the metadata fields that are obtainable and language processing related issues like typographical errors and uppercase/lowercase typing etc.

1. Context and Scope of the Project

1.1 Project Outline

This project aims to build a comprehensive music database that consolidates information about songs from various online sources. Initially, the focus will be on extracting song data from a Mendeley dataset [2] (also available in Kaggle [3]) that is given as a csv file. After establishing the foundation with Mendeley data, the project will be expanded by integrating daily song data from the Spotify API [4]. Specifically, the Top 50 Global playlist will be fetched daily to enhance the database with real-time data.

The database will be designed to facilitate data analysis, visualization, and machine learning models, supporting a recommendation system based on song similarities. By incorporating dynamic Spotify data, we aim to continuously enrich our dataset and strengthen our machine learning algorithms with the latest trends in music. This will allow us to deliver more accurate and relevant recommendations.

1.2 Intended Implementation

1. Python Programming Language

- **Description**: The entire project is developed using Python. Python is highly effective for data processing, database integration, and interacting with APIs.

2. Data Manipulation Libraries

- **Pandas**: Used for reading, processing, and writing data from CSV files to the Mysql database. Pandas is a powerful library for working with structured data.

- **Requests**: Utilized to make HTTP requests to the Spotify API. It is a popular library for interacting with web APIs in Python.
- **BeautifulSoup**: (If scraping was involved) Used for web scraping, particularly when parsing HTML or XML data.

3. Data Storage and Management

- **MySQL**: An embedded relational database used for storing and managing data locally. It is a lightweight and file-based DBMS ideal for small to medium-scale applications.
- **SQL**: SQL queries were embedded in Python code to handle database operations such as creating tables, inserting, and updating records.

4. Web Scraping and API-Based Data Collection

- **BeautifulSoup**: Used to scrape data from websites, particularly for parsing HTML content from Kworb.net, Genius.com or other sources.
- **Spotify API**: Spotify's Web API was used to fetch real-time data from the Top 50 Global Playlist, including artist and track information.

5. Database Management Systems (DBMS)

- **MYSQL**: Chosen for its ease of use and file-based database management. SQL queries were used to create tables and insert or retrieve data in the database.

6. Environment Variables and Secrets Management

- **dotenv**: Used to securely manage sensitive information such as API keys through environment variables. This ensures that confidential data is not hardcoded in the scripts.

7. Timestamp and Date Operations

- **datetime**: Employed to generate timestamps, which are then stored in the database along with the inserted data to track when entries were made.

8. Database Containerization

- **Docker**: Docker is used to containerize the application and isolate the environment, ensuring consistent performance across different systems.
- **FastAPI:** Currently two APIs are planned. One API could be used for querying the database for various information. This can include, for example, the statistical information on the number of tracks per artist, grouped information of the number of tracks per year, total number of artists or total number of

tracks. For each of these operations, different endpoints can be implemented, with appropriate query parameters. The second API could contain the recommendation system algorithm. Using a dedicated endpoint, it is planned to list similar songs, for an example song given as input.

9. Database Automation

- **Airflow**: The data included in the database from Spotify API has a dynamic nature in certain ways. For example, the most popular tracks list can change on a daily or weekly basis. To handle such updates to the data, we foresee that Airflow can be used for re-fetching the playlist information on a regular basis. In Airflow, one can create an automated (scheduled) process to insert data to the database.

2. Data Collection and Storage

Initial Data Collection: The preliminary part of the database in the current project is imported from a dataset that exists in Mendeley datasets [2], and also available in Kaggle website [Ref. 3]. According to the authors of the dataset [2], they have collected the data from Spotify using a Python tool called ***spotipy*** [5]. In addition, they have collected lyrics information from the Lyrics Genius API. The data is available in CSV format (also with the name **tcc_ceds_music.csv** in our repository) and contains approximately 29000 songs with several fields including textual and numerical values. The numerical values stem from the audio recordings of the songs and they represent music-related metrics such as loudness, acousticness, danceability etc. which are excluded in the current project. A small sample of the data with the relevant fields is displayed in Table 1.

Table 1: Sample data from the Mendeley dataset

	artist_name	track_name	release_date	genre	lyrics	len
0	mukesh	mohabbat bhi jhoothi	1950	pop	hold time feel break feel untrue convince spea...	95
1	frankie laine	i believe	1950	pop	believe drop rain fall grow believe darkest ni...	51

Among the fields, the column names are quite descriptive, whereas the column *len* contains the information about how many words are included in the *lyrics* field.

Daily Data Collection: We will utilize Spotify's Global Top 50 Songs list for our data maintenance system. Each day, we will retrieve detailed track information from this playlist and store it in our relational database model. This data will help us maintain

up-to-date insights into popular tracks, including track metadata, artist details, and album information, ensuring the system reflects current trends in the music industry. The task is performed via requests to the API of Spotify. An example of the returned content from the "/playlist" endpoint is given in Table 2, where the *json* output is reprocessed for visual clarity.

Table 2: Sample data from Spotify API

```
Sample Track Informations Return From Spotify API:
--------------------------------------------------
Track Name: Die With A Smile
Artist Name: Lady Gaga
Album Name: Die With A Smile
Release Date: 2024-08-16
Album Type: single
Total Tracks in Album: 1
Artist ID: 1HY2Jd0NmPuamShAr6KMms
Artist URI: spotify:artist:1HY2Jd0NmPuamShAr6KMms
Duration (ms): 251667
Popularity: 99
Genres: ['art pop', 'dance pop', 'pop']
```

Web Scraping and other API requests: In addition to the above-mentioned data resources, web scraping and other API requests were implemented to collect song data. In fact, these methods can be applied sequentially in order to fill in several fields of the data. These steps are listed as follows.

1. Web scraping - Kworb.net: The kworb website contains rich information about the Spotify songs. In particular, the URL link https://kworb.net/spotify/artists.html contains the list of most streamed 3000 artists of all time. The web scraping of this url gives the list of artists. In a second step, the reference link for each artist leads to a new url, where a full list of songs for the artist can be reached (max 500 items are displayed). A Sample of the collected data is given in Table 3. The full size of song data scraped from Kworb amounts to ~500,000 (available in the repository with the name ***spotify_track_ids.csv***).

   Table 3. An example of data collected from Kworb.net via web scraping

Song_title	Spotify_track_id	Artist
Bienvenidos a Jalisco	3UoM05HDtbGb1jw029Hox3	Grupo Arriesgado
Creepin'	0G6bhqsZpxapSAoKLjHK86	Bootsy Collins
Di Ujung Malam	33LIJCqwuyvSB78SYK8AWM	Payung Teduh
Contigo	5ye0hEA5G6Zfhz2Ejx6NIo	Antonio Aguilar

2. Genius API - search request. With the information obtained in Table 3, an API request can be performed to the Genius API in order to obtain the distinct web URL of a specific song. The search parameters for the API query includes the song name and the artist name, which are obtained from the data in Table 3.

3. Genius Web Scraping - lyrics information. Consequent to the step 2 given above, a web-scraping method can be applied for an url from the Genius website to extract the lyrics information. See an example link at Ref. [7].

Application of all three consequent steps above results in collecting data that is in identical form with the Mendeley data from Ref. 2. Hence, this allows us to expand the data size in the database. Initially, approximately 1000 songs with this methodology were collected. In order to retrieve data for more songs would require automated scripts and/or scheduling.

Data Ingestion: The data from the CSV file will be imported into a MYSQL database. The database follows a star schema data model, which is designed to support efficient queries for data analysis and machine learning tasks.

DATA MODEL :

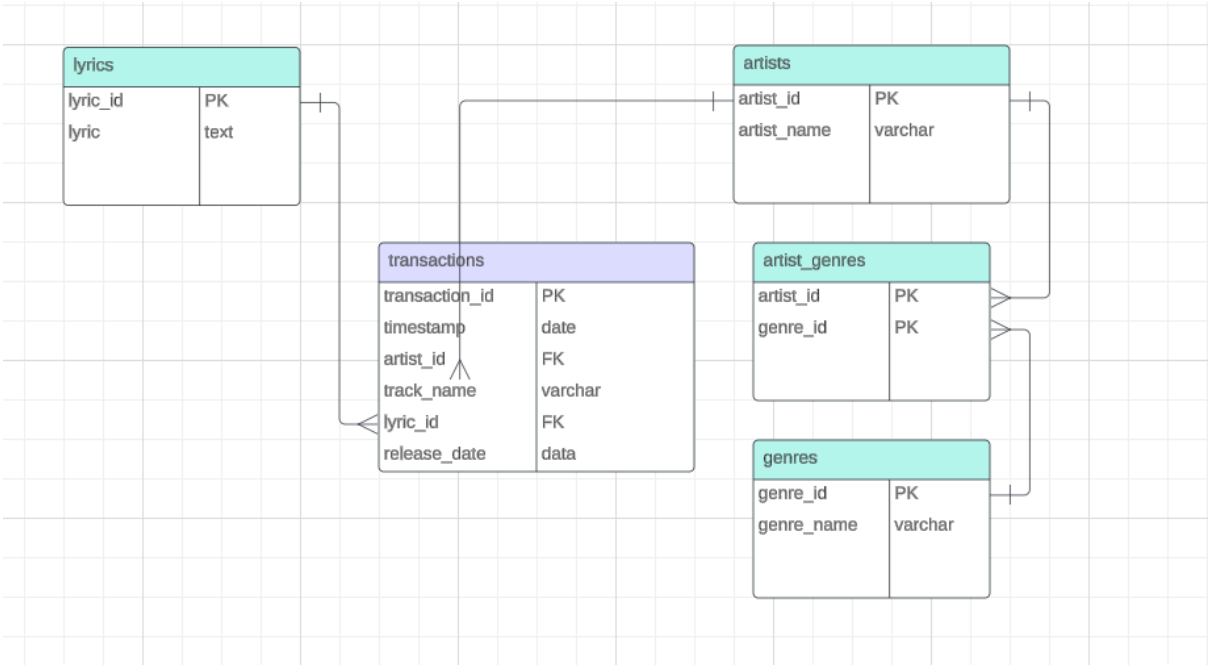

Fig. 1: UML Diagram for the relational database

3. Data Consumption

Machine Learning - Recommender system

As a machine learning model for the database, a recommender system is implemented in this project. The fields like the *track_name* and the *lyrics* contain specific textual information that can be processed for certain purposes; in this case the similarity between the songs in terms of their language content.

The recommendation system (RecSys) algorithm is implemented based on the *CountVectorizer* operator from the Scikit-learn library. The CountVectorizer class tokenizes the words included in a data field; in our case it is the *track_name* field. When the database contains songs from several different languages, the number of tokens in the data grows rapidly. This effect can be restrained using the *max_features* parameter, which is set as 15000 in the current work considering the capabilities of computer resources.

The application of the vectorizer on all data rows yields a sparse matrix with the dimensions M x N (*count_matrix*), where M is the number of rows and N is the max_features parameter. Currently, 29,000 tracks from Kaggle and 100,000 tracks from Spotify are used in the training of the algorithm.

Multiplication of the *count_matrix* with its transpose constructs the so-called *cosine-similarity* matrix, the column-wise values of which are then used to detect the similarity between different tracks. During inference, for a track from the test data, the similarity values can be obtained by a vector-matrix product, i.e. by multiplying the feature vector (1xN) with the transpose of the *count_matrix* (NxM).

4. Container Architecture and Services

In this project, we used Docker containers to manage the services, allowing for easy deployment and scalability. The following section describes the role of each container and the associated functionalities.

1. MySQL Database Container

- **Container Name**: mysql-container
- **Description**: This container hosts the MySQL database, which stores all the project-related data, such as the processed song information and user inputs.
- **Purpose**: The database handles the data model required for storing song metadata and user interaction records.

2. Music Data API Container

- **Container Name**: music-db-api

- **Description**: This container runs several Python scripts that interact with the MySQL database to automate tasks like creating tables, inserting data from CSV files, and pulling data from external APIs (e.g., Spotify Top 50).

- **Scripts**:
    - **create_tables.py**: Creates the necessary tables in the database.
    - **inserts_from_csv.py**: Reads the data from the CSV file (/app/data/tcc_ceds_music.csv) and inserts it into the respective database tables.
    - **spotifytop50.py**: Fetches the top 50 songs from the Spotify API and stores them in the database.

- **API Endpoints** (Using FastAPI):
    - **GET /status**: Checks if the API is running and returns the status of the container.
    - **POST /create-tables**: Executes the create_tables.py script to create the database tables.
    - **POST /insert-from-csv**: Runs the inserts_from_csv.py script to insert data from the CSV file into the database.
    - **POST /spotify-top-50**: Executes the spotifytop50.py script to fetch and insert the Spotify Top 50 songs into the database.

3. ML Recommender System API Container

- **Container Name**: music-recsys-api
- **Description**: This container includes the endpoints necessary for obtaining recommendation, i.e. similar tracks for a given input track name. The users can either input the track name of their own choice or use the query function to obtain one song from the ones that are already inserted into the database.

- **API Endpoints** (Using FastAPI):
    - **GET /status**: Checks if the API is running and returns the status of the container.
    - **POST /track-name-query**: Performs a query to the database to receive a track name. The query parameter is the *transaction_id,* which is the primary key of the Transactions (facts) table.
    - **POST /get-recommendation**: This endpoint lists the 10 most similar track names for given track name input. The query parameter is the *track_name* which is a string variable that can contain space characters. Hence, it is possible to input track titles consisting of several words. The results (recommended tracks) are listed starting from the best match.

5. Unit Tests

To ensure the reliability and functionality of the scripts in our project, unit tests have been developed using the pytest framework. The tests are designed to validate the behavior of the following scripts:

1. **create_tables.py:** This script is responsible for creating the necessary database tables.
    - **Test Script:** test_create_tables.py
    - **Testing Approach:** The test mocks the database connection using unittest.mock to verify that the correct SQL queries are executed for table creation. It checks for the existence of the tables after execution.

2. **inserts_from_csv.py:** This script handles inserting data from a CSV file into the database.
    - **Test Script:** test_inserts_from_csv.py
    - **Testing Approach:** The test simulates reading a CSV file and mocks the database insertions using unittest.mock. It ensures that the correct insert queries are generated based on the mock data and validates the interaction with the database.

3. **spotifytop50.py:** This script retrieves data from the Spotify API and inserts it into the database.
    - **Test Script:** test_spotifytop50.py
    - **Testing Approach:** The test mocks the API responses using unittest.mock to simulate the data retrieved from Spotify. It then verifies that the data is correctly processed and inserted into the database.

Mocking

Due to the nature of the project, where the unit tests run locally while the actual code executes within a container, the tests do not require a live database connection. Instead, the mysql.connector.connect method is mocked to prevent any real database interactions. This allows for testing the SQL logic without the need for a database, making the tests faster and more reliable.

Test Coverage

Each of the test scripts targets specific functions within their corresponding source files. The use of mock objects allows the tests to focus on the functionality of the code without being affected by external dependencies, such as database states or API availability.

Testing ML Inference

Unit tests are implemented for the inference function of the Machine Learning algorithm. The inference function of the recommender system takes two inputs: *i*) the fold number of the data to identify the test partition of the train-test splits and *ii*) a random row index which is used to select a single track name from the dataset.

The tests are implemented in two parts. One section tests that the corresponding Python module takes the correct parameters. Namely, the permissible values for fold number range from 0 to 4, and the row index for each fold should be between 0 and the length of the csv file for each fold.

The second part of the unit tests is implemented for the standalone function. The standalone *get_recommendation* function takes the *track_name* as the input parameter. Here, the tests assure that the input is a string variable, and not null nor a numeric value. Moreover, the results (the 10 recommended tracks) are returned as Pandas DataFrame, which is asserted in a separate test function.

Conclusions

An overview of the data exploration, database creation and data registration, as well as a basic machine learning algorithm for song recommendation was presented in this report.

Database:

The database design is foundational to the project's success, providing a robust structure to store and manage song metadata and user interactions. By utilizing a MySQL database, we ensured that data was organized efficiently, enabling quick access and retrieval. The database supports various functionalities, such as storing processed song information, managing user inputs, and facilitating data updates from external APIs like Spotify. The implementation of automated scripts for data insertion and table creation further streamlines database operations, ensuring data integrity and reducing manual effort.

API & Containers:

The use of Docker containers has greatly enhanced the project's deployment and scalability. By compartmentalizing services into distinct containers, we can manage and update each component independently, which simplifies maintenance and allows for efficient resource utilization. The music data API and the ML recommender system API work seamlessly to facilitate data interaction and provide song recommendations to users. This modular architecture not only supports rapid development and testing but also positions the project for future enhancements and scalability, making it adaptable to varying workloads and user demands.

ML algorithm:

As a data consumption method, one version of the Recommender System algorithm is implemented, allowing up to 15,000 word counts and using approximately 130,000 track names in training. One can enhance the versions of the RecSys algorithm by adding other data fields like genre and lyrics, and adjusting the hyperparameters of the algorithm. As the included text fields grow, the variety of the extracted features would increase, as well as the total size of the training data. However, this could then bring in necessities for better computational resources regarding CPU and RAM capacities.

References

1. https://en.wikipedia.org/wiki/List_of_online_music_databases
2. Mendeley Music Dataset 1950 to 2019. https://data.mendeley.com/datasets/3t9vbwxgr5/3
3. Kaggle URL for Music Dataset - 1950 to 2019 https://www.kaggle.com/datasets/saurabhshahane/music-dataset-1950-to-2019
4. Spotify Web API: https://developer.spotify.com/documentation/web-api
5. Spotipy Python library. https://spotipy.readthedocs.io/en/2.24.0/
6. Genius API - search endpoint: https://docs.genius.com/#/search-h2
7. Genius Web, example URL for a song. https://genius.com/Bootsy-collins-leakin-lyrics