# Spotify Music Discovery based on Individual User Data

Urja Panchal
21101631
u4pancha@uwaterloo.ca

Rushi Raval
21086166
r3raval@uwaterloo.ca

## Abstract

*This project explores the realm of music recommendation and personalized playlist generation using Spotify's extensive dataset obtained from Kaggle. Leveraging data mining techniques, we aim to analyze users' listening habits and preferences to design an effective recommendation system. The project involves preprocessing the dataset, designing a relational schema, and implementing algorithms to generate personalized playlists based on individual user preferences.*

## 1. Introduction

Spotify has transformed the landscape of music consumption, offering users an extensive library of songs, personalized recommendations, and curated playlists. With its vast user base spanning the globe, Spotify collects a wealth of data on listeners' habits, preferences, and trends.

This data presents a valuable opportunity to gain insights into user behavior and preferences, enabling the creation of personalized playlists tailored to individual tastes.

In this project, we delve into the realm of data analysis and music recommendation using Spotify's vast dataset obtained from Kaggle. Our objective is to unravel patterns within users' listening behaviors and preferences through the application of data mining techniques. By analyzing this data, we aim to design an effective recommendation system that not only provides users with curated playlists but also enhances their music discovery experience.

Through the development of algorithms that leverage user data, we strive to offer Spotify users an unparalleled level of personalized music recommendations, ensuring that they always have the perfect soundtrack for every moment of their day.

## 2. Database design

In the era of digital media consumption, platforms like YouTube and Spotify have revolutionized how people discover, consume, and interact with music and video content. These platforms not only serve as sources of entertainment but also as vast repositories of valuable data. Leveraging this data effectively can provide insights into user behavior, content preferences, and trends in the ever-evolving landscape of media consumption.

For our project, we embarked on a quest to explore the wealth of data available on YouTube and Spotify. Kaggle, a prominent platform for data science enthusiasts, hosts a plethora of datasets encompassing various domains, including media and entertainment. Our search led us to two compelling datasets related to YouTube and Spotify.

After careful consideration, we opted for the second dataset due to its rich complexity and extensive attributes encapsulated within its CSV files. This dataset promises to offer a comprehensive glimpse into the intricacies of user engagement, content metadata, and performance metrics on spotify platform.

In this project report, we will delve into the process of harnessing the potential of this dataset to construct a robust and insightful database. Our aim is to extract meaningful insights, facilitate data-driven decision-making, and ultimately enhance user experiences within the realms of music and video streaming platforms. Through meticulous analysis and database design, we endeavor to unlock the hidden gems within the data and pave the way for future innovations in this dynamic domain

### 2.1. CSV and its explanation

The dataset selected for this project consists of three primary CSV files: Albums, Tracks, and Artists. These files contain a multitude of attributes capturing diverse aspects of music albums, individual tracks, and associated artists, facilitating a comprehensive analysis

of the music ecosystem on the Spotify platform.

**2.1.1. Albums.csv:** The Albums CSV file provides detailed information about music albums available on Spotify. Each entry includes attributes such as album_type, artist_id, available_markets, external_urls, href, id, images, name, release_date, release_date_precision, total_tracks, track_id, track_name_prev, uri, and type. Additionally, the file contains metadata related to album images and track details, contributing to a holistic understanding of album characteristics and distribution.

**2.1.2. Tracks.csv:** The Tracks CSV file offers insights into individual tracks hosted on Spotify, accompanied by a plethora of audio features and metadata. Attributes such as acousticness, album_id, analysis_url, artists_id, available_markets, country, danceability, disc_number, duration_ms, energy, href, id, instrumentalness, key, liveness, loudness, mode, name, playlist, popularity, preview_url, speechiness, tempo, time_signature, track_href, track_name_prev, track_number, uri, and valence provide a nuanced understanding of track composition, metadata, and engagement metrics.

**2.1.3. Artists.csv:** The Artists CSV file delineates information about music artists featured on Spotify, including artist_popularity, followers, genres, id, name, track_id, track_name_prev, and type. Each entry provides a unique artist ID along with the artist's name, facilitating identification and categorization of artists based on their popularity and genre affiliations.

Collectively, these datasets constitute a rich repository of music-related data, encompassing albums, tracks, and artists, along with a diverse array of attributes conducive to comprehensive analysis and exploration. Leveraging these datasets, our project aims to unravel insights into music trends, user preferences, and artist dynamics, fostering a deeper understanding of the digital music landscape on Spotify

## 2.2. ER model

In the initial phase of our project, our focus was on comprehensively identifying the entities within the Spotify music dataset and delineating all relevant attributes associated with each entity. This crucial step laid the groundwork for subsequent data processing and database design activities.

**2.2.1. Identification of Entities:** Our analysis of the Spotify dataset revealed three primary entities central to the music ecosystem: albums, tracks, and artists. These entities represent fundamental components within the Spotify platform, each playing a distinct role in facilitating music discovery, consumption, and artist engagement.

**2.2.2. Attributes of each Entity:** This section outlines the specific attributes associated with each entity in the dataset. It provides a comprehensive list of the key characteristics or properties that describe albums, artists, and tracks within the database.

**Albums Entity:**

- album_type
- artist_id
- available_markets
- external_urls
- href
- id
- images
- name
- release_date
- release_date_precision
- total_tracks
- track_id
- track_name_prev
- uri
- type

**Artist Entity:**

- artist_popularity
- followers
- genres
- id
- name
- track_id
- track_name_prev
- type

**Tracks Entity:**

- acousticness
- album_id
- analysis_url
- artists_id
- available_markets
- country
- danceability
- disc_number
- duration_ms
- energy
- href
- id
- instrumentalness
- key
- liveness
- loudness
- mode
- name
- popularity
- preview_url
- speechiness
- tempo
- time_signature
- uri
- valence

**2.2.3. Attributes of each Entity:** By identifying these entities and enumerating their associated attributes, our aim was to establish a comprehensive understanding of the underlying data structure and content within the Spotify dataset. This initial step enabled us to lay a solid foundation for subsequent data cleaning and refinement processes, ensuring that only relevant and essential attributes are retained for database design and analysis purposes.

With the entities and their attributes clearly defined, our focus shifted towards the refinement of the dataset by eliminating any irrelevant attributes and preparing the data for database schema design. This iterative process allowed us to streamline the dataset and optimize it for database management, facilitating efficient data storage, retrieval, and analysis in our project.

In summary, the identification of entities and their associated attributes served as a crucial precursor to subsequent data processing and database design activities, enabling us to embark on a structured and systematic approach towards achieving our project objectives within the Spotify music ecosystem.

## 2.3. Refinement of Attributes:

Upon thorough examination of the datasets containing information on albums, tracks, and artists from Spotify, a meticulous process was undertaken to identify the most pertinent attributes essential for constructing an Entity-Relationship Diagram (ERD). This process involved scrutinizing each attribute to discern its relevance in capturing the essential characteristics of the respective entities.

**Albums Entity:** After careful consideration, the following attributes were deemed vital for delineating the albums entity:

- album_type
- available_markets
- external_urls
- href
- id
- name
- release_date
- release_date_precision
- total_tracks

– uri

For the Albums entity, we have removed the artist_id from here as irrelevant to the Albums entity.

**Artist Entity:** For the artist entity, the attributes crucial for representing the essence of an artist on Spotify were identified as:

– artist_popularity

– followers

– genres

– id

– name

**Tracks Entity:** Similarly, for tracks, the following attributes were selected to encapsulate the core characteristics of individual songs:

– acousticness

– analysis_url

– artists_id

– country

– danceability

– disc_number

– duration_ms

– energy

– href

– id

– instrumentalness

– key

– liveness

– loudness

– mode

– name

– popularity

– preview_url

– speechiness

– tempo

– time_signature

– uri

– valence

We eliminated the available_markets element from the track entity because it is redundant with the available_market attribute for albums, which are made up of songs and provide the same available market information. Likewise eliminated album_id attribute as irrelevant to tracks entity. By meticulously curating the attributes for each entity, the focus was to streamline the database design process and ensure that only the most relevant and essential information is captured in the ERD. This refined selection of attributes lays the foundation for constructing a robust and efficient database schema, facilitating effective data management and analysis within the Spotify music ecosystem.

Moreover, the objective of creating a database system is to capture the relationships between artists, tracks, and albums comprehensively. Recognizing this aim, the data was thoroughly studied and processed to eliminate any irrelevant attributes, ensuring that the final selection reflects the intrinsic connections and dependencies among these entities.

## 2.4. First Draft ERD:

Following a meticulous cleanup of attributes and identification of entities within the Spotify music dataset, we present the initial version of the Entity-Relationship Diagram (ERD). This draft aims to encapsulate the essential relationships and attributes pertinent to albums, artists, tracks, and track analysis elements, while maintaining clarity and coherence. The following outlines the structure of the first draft of our ERD.

**1. Albums Entity:**

- Retains relevant attributes identified earlier, such as album_type, artist_id, name, and release_info.
- Primary key (PK) for this entity is album_id (rename id to album_id, making every attribute unique across ERD), each id assigned to an album within the dataset will be unique across the entire dataset and within the Spotify platform.
- Consolidates release_date and release_date_precision into a single attribute, release_info, for enhanced clarity.

- Available markets transformed into a multi-attribute to capture diverse market availability.
- Introduces a composite attribute, album_access_link, comprising album_link (URL) and album_api (API) components.

**2. Artists Entity:**

- Maintains relevant attributes, including artist_popularity, followers, genres, and name.
- artist_id serves as the primary key for this entity (rename id to artist_id, making every attribute unique across ERD), each id assigned to an artist within the dataset will be unique across the entire dataset and within the Spotify platform.
- Genres identified as a multi-value attribute to accommodate diverse genre affiliations.

**3. Tracks Entity:**

- Primary key designated as track_id, each id assigned to a track within the dataset will be unique across the entire dataset and within the Spotify platform.
- Attributes include track_name, track_preview_url, duration_ms, track_available_market, and track_uri.
- Introduces a composite attribute, track_access_link, comprising track_access_url and track_access_api.

**4. Track_Analysis_Elements Entity:**

- Represents a weak entity, dependent on the Tracks entity.
- Attributes include track_id (primary key and foreign key), track_analysis_link (as the primary key), and track analysis elements such as valence, time_signature, tempo, speechiness, track_popularity, mode, loudness, liveness, key, instrumentalness, energy, disc_number, danceability, and acousticness.

**5. Relationships:**

- Album - Artist: Depicts the release relationship between albums and artists.
- Album - Tracks: Illustrates the consists relationship between albums and tracks.
- Track - Artist: Denotes the composes relationship between tracks and artists.
- Track - Track Analysis Elements: Depicts the analysed_by relationship between tracks and track analysis elements.

By delineating these entities and relationships, the ERD provides a structured framework for representing the interdependencies and associations within the Spotify music dataset. This draft serves as an initial blueprint for our database design, laying the groundwork for further refinement and optimization as we progress in our project journey.

## 2.5. Trade OFF:

**Design Choices for the Initial ERD**

During the process of designing the initial Entity-Relationship Diagram (ERD) for our Spotify music dataset, several design choices were made to ensure efficiency, data integrity, and simplicity. Below, we outline the key design decisions made for the Albums entity:

**1. Handling Release Date Information:**

- Initially, the dataset presented two distinct columns: release_date, indicating the date or year of release, and release_date_precision, denoting whether the provided date is complete or solely represents the year.
- Contemplating the structuring of this information, we deliberated the creation of a composite attribute, integrating day, month, and year components. However, upon thorough examination, we opted for a more streamlined approach. Thus, we consolidated these attributes into a single, versatile attribute named release_info.
- With the introduction of release_info, our schema can seamlessly accommodate various formats, encompassing both complete dates in ”YYYY-MM-DD” format and sole years represented as ”YYYY”. This decision was made to enhance simplicity and flexibility in data handling.
- Additionally, by adopting release_info as a singular attribute, we circumvent the unnecessary complexity that would arise if day and month columns were included with potential null values. This streamlined approach to release_info ensures efficient data management and alleviates the burden of unnecessary complexities.

**Design Choices for Composite Access Links**

In our design process for the initial Entity-Relationship Diagram (ERD) of the Spotify music dataset, we carefully considered the representation of access links for albums and tracks. Here's a detailed explanation of our design choices regarding composite access links:

1. **Access Links for Albums and Tracks:**

- Each album and track on Spotify has unique access links that facilitate user access through different platforms, such as the Spotify web app and desktop app.
- Recognizing the importance of capturing both access points for comprehensive data representation, we decided to model access links as composite attributes.

2. **Composite Attributes:**

- For albums, the composite attribute album_access_link consists of both the album's URL and API endpoint.
- Similarly, for tracks, the composite attribute track_access_link comprises the track's URL and API endpoint.

3. **Rationalization of Design Choices:**

- By utilizing composite attributes for access links, we ensure that both access points—via URL and API endpoint—are captured within our database schema.
- This design choice reflects the versatility of Spotify's platform, where users can access albums and tracks through various channels.
- Additionally, modeling access links as composite attributes simplifies data management and ensures clarity in representing the multiple access points available for each album and track.

## 2.6. Final ERD

After careful consideration of the design choices and content discussed above, we have finalized the Entity-Relationship Diagram (ERD) for the Spotify music dataset. This comprehensive ERD encapsulates the essential entities, relationships, primary keys (PK), cardinality, and discriminators for weak entities, providing a clear and structured representation of the dataset. Below is an overview of the finalized ERD:

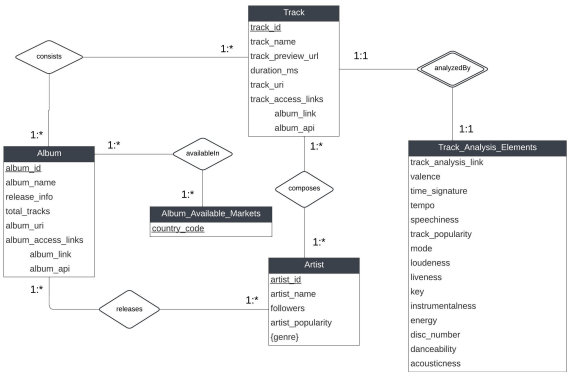


**Figure 1. Final ER Diagram**

**Entities:**

1. Albums:

- Primary Key (PK): album_id
- Attributes: album_type, name, release_info, total_tracks, album_access_link (consisting of album_link and album_api)

2. Tracks:

- PK: track_id
- Attributes: track_name, track_preview_url, duration_ms, track_uri, track_access_link (consisting of track_link and track_api)

3. Artists:

- PK: artist_id
- Attributes: artist_popularity, followers, genres, name

4. Album_Available_Markets:

- No Primary Key (PK)
- Attributes: album_id (FK), available_market

5. Track_Analysis_Elements:

- Discriminator: track_analysis_link
- Attributes: track_id (FK), valence, time_signature, tempo, speechiness, track_popularity, mode, loudness, liveness, key, instrumentalness, energy, disc_number, danceability, acousticness

**Relationships:**

- **release:** Album – Artist
  Cardinality:

  - Album w.r.t. Artist (1:*)
  - Artist w.r.t. Album (1:*)

- **consists:** Album – Tracks
  Cardinality:
  - Album w.r.t. Tracks (1:*)
  - Tracks w.r.t. Album (1:*)

- **composes:** Track – Artist
  Cardinality:
  - Track w.r.t. Artist (1:*)
  - Artist w.r.t. Track (1:*)

- **analyzedBy:** Track – Track_Analysis_Elements
  Cardinality:
  - Track w.r.t. Track_Analysis_Elements (1:1)
  - Track_Analysis_Elements w.r.t. Track (1:1)

**Design Considerations:**

- Primary Keys (PK): Each entity has a designated primary key to uniquely identify its records.
- Cardinality: Relationships are defined with appropriate cardinality to indicate the number of instances of one entity related to another entity.
- Composite Attributes: Both album_access_link and track_access_link are composite attributes consisting of component attributes, providing clarity and structure to access links for albums and tracks, respectively.
- Discriminator for Weak Entity: The Track_Analysis_Elements entity is identified as a weak entity, with track_analysis_link serving as the discriminator to uniquely identify its records.

## 2.7. Relational Schema

**2.7.1. Design Choices:** In our conversion process from the Entity-Relationship Diagram (ERD) to a relational schema, we encountered multi-valued attributes such as country codes in the Albums entity and genres in the Tracks entity.

To mitigate potential challenges such as multiple entries, data entry errors, data loss issues, and difficulties in updating and inserting data, we opted for separate entities (Album_Available_Market and Artist_Genre, respectively).

This decision ensures data integrity, minimizes redundancy, facilitates efficient data management, and enhances query performance within the relational schema.

By segregating multi-valued attributes into distinct entities, we aim to streamline data handling and ensure the accuracy and reliability of album and track data.

**2.7.2. Conversion from ERD to Relational Schema** In our transition from the Entity-Relationship Diagram (ERD) to a relational schema, we've made another design decision regarding the track_access_links attribute. This attribute consists of a unique track_id followed by the DNS name, generated to ensure correct URL links pointing to the respective track ID. This approach serves multiple purposes:

1. **Ensuring Correct URL Links:** By generating the track_access_links as a composite attribute, we ensure that each track_id is correctly associated with its corresponding URL link. This helps maintain the integrity of data and ensures accurate navigation within the application.
2. **Reducing Data Entry Errors:** Generating the URL links automatically reduces the likelihood of data entry errors that may occur if URLs were manually entered. This helps maintain data accuracy and reliability.
3. **Error Identification:** If an incorrect track ID is inadvertently inserted into the database, the generated track_access_links will correspond to incorrect URLs. This discrepancy acts as a marker, aiding in the identification and correction of data entry errors.

While designing the schema, we had the option to separate the URL into individual components such as track ID and DNS of the URL and API. However, considering the application's requirement to use the entire URL or API link to generate playlists via Spotify API, we decided against dividing the URL.

Instead, we opted to generate a composite column. This approach ensures the seamless integration of data across the application and simplifies the process of playlist generation.

Similar design choices have been applied to other attributes, such as album_access_links (albums), track_uri (track), and track_analysis_link in the weak entity track_Analysis_Elements, to maintain consistency and optimize data handling within the relational schema.

**Unknown/ Not available data:**

We encountered instances where certain attributes, such as genre for tracks and country codes for albums,

contained records marked with the value []. These values denoted unknown or unavailable data. To ensure consistency and improve data clarity, we made the decision to represent unknown or unavailable data by using NULL values instead of [].

By adopting NULL values, we signify that there is no known data or that the data is unavailable for those specific records. This approach enhances the readability of the data and aligns with standard database practices.

**Check constraints and data type:**

For the column album_type in the album table, we have chosen the enum data type as it can only have three possible values: single, album, or compilation. By utilizing the enum data type, we restrict the allowed values to these specific options, thereby preventing any erroneous data entry and ensuring consistency in the album types recorded in the database.

For the release_info attribute, we chose the varchar data type to accommodate both year and full date values that may be loaded from the dataset.

Similarly, for the mode column in the track_analysis_elements table, we have also selected the enum data type as it can only have two possible values: 0 or 1. This ensures that only valid mode values are accepted, preventing any invalid or incorrect data from being stored in the database.

Furthermore, we have implemented check constraints to enforce additional data constraints and ensure data integrity. These constraints are visible in the database schema code and are designed to validate the values stored in certain columns against specified criteria. By employing check constraints, we enforce data consistency and prevent the insertion of erroneous or out-of-range values into the database.

**Album_Available_Market:**

In the creation of the Album_Available_Market table, we have isolated the country_code column within this table and established a relationship set named availableIn to capture the album_id and country_code records. This design choice was made to address several key considerations:

1. **Ease of Maintenance:** By segregating the country codes into a separate table (Album_Available_Market), any updates or changes to the standard list of country codes can be efficiently managed. Updates only need to be applied to the Album_Available_Market table, ensuring simplicity and ease of maintenance.

2. **Simplified Updates:** Maintaining a separate table for country codes streamlines the process of updating or modifying records associated with specific country codes. Without this segregation, any changes required to either the country code list or the album_id mapped records would necessitate alterations to all affected records. This could potentially result in a complex and error-prone task.

We have divided the genre attribute into a separate table and established a relationship set to hold the artist_id and genre records. This decision was made to address several key considerations:

1. **Reduction of Data Redundancy:** Segregating genre into a separate table allows us to avoid data redundancy by storing each genre as a distinct record. This approach eliminates the need to repeat genre information for each artist, contributing to a more efficient and streamlined database structure.

2. **Simplified Updates:** By maintaining genre information in a separate table, updates or changes to artist genres can be managed more easily. Without this segregation, modifying or updating artist genres would require changes to multiple records, leading to potential data inconsistencies and complexities.

Initially, we considered encoding genre as a datatype set to accommodate multiple values. However, this approach would still result in data redundancy and could complicate the process of updating or changing artist genres. Therefore, we opted to segregate genre into a separate table, ensuring a more efficient and manageable solution.

**2.7.3. Create Queries** To create the necessary tables for the database schema, the following SQL queries are executed:

```
CREATE TABLE Artist (
    artist_id CHAR(23),
    artist_name VARCHAR(255),
    followers INT,
    artist_popularity INT,
    PRIMARY KEY(artist_id));

CREATE TABLE Genres (
    genre VARCHAR(255),
    PRIMARY KEY(genre));

CREATE TABLE Artist_Genres (
    artist_id CHAR(23),
```

```
    genre VARCHAR(255),
    PRIMARY KEY(artist_id, genre),
    FOREIGN KEY (genre) REFERENCES
    Genres(genre),
    FOREIGN KEY (artist_id) REFERENCES
    Artist(artist_id));

CREATE TABLE Album (
    album_id CHAR(23) PRIMARY KEY,
    album_name VARCHAR(255),
    album_type ENUM('single', 'album',
    'compilation'),
    release_info VARCHAR(10),
    total_tracks INT,
    album_uri CHAR(37) GENERATED
    ALWAYS AS (concat('spotify:album:',
    album_id)) STORED,
    album_access_url VARCHAR(68)
    GENERATED ALWAYS AS
    (concat('{\'spotify\': \'
    https://open.spotify.com/album/',
    album_id, '\'}')) STORED,
    album_access_api VARCHAR(56)
    GENERATED ALWAYS AS (concat
    ('https://api.spotify.com/v1/albums/',
    album_id)) STORED);

CREATE TABLE Album_Available_Market (
    country_code CHAR(2) PRIMARY KEY);

CREATE TABLE availableIn (
    country_code CHAR(2),
    album_id CHAR(23),
    PRIMARY KEY(album_id, country_code),
    FOREIGN KEY (album_id) REFERENCES
    Album(album_id),
    FOREIGN KEY (country_code)
    REFERENCES
    Album_Available_Market(country_code));

CREATE TABLE Track (
    track_id CHAR(23) PRIMARY KEY,
    track_name VARCHAR(255),
    track_preview_url VARCHAR(255),
    duration_ms INT,
    track_uri CHAR(36) GENERATED ALWAYS
    AS (concat('spotify:track:',
    track_id)) STORED,
    track_access_url VARCHAR(69)
    GENERATED ALWAYS AS
    (concat('{\'spotify\': \
    'https://open.spotify.com/tracks/'
    , track_id, '\'}')) STORED,
    track_access_api VARCHAR(56)
    GENERATED ALWAYS AS (concat('
    https://api.spotify.com/v1/tracks/'
    , track_id)) STORED);

CREATE TABLE Track_Analysis_Elements (
    track_id CHAR(23),
    track_analysis_link CHAR(64)
    GENERATED
    ALWAYS AS (concat(
    'https://api.spotify.com/v1/
    audio-analysis/', track_id)) STORED,
    valence DECIMAL(5,4),
    time_signature INT,
    speechiness DECIMAL(5,4),
    track_popularity INT,
    mode ENUM('0', '1'),
    loudness DECIMAL(5,3),
    liveness DECIMAL(5,4),
    `key` INT,
    instrumentalness DECIMAL(9,8),
    disc_number INT,
    danceability DECIMAL(5,4),
    acousticness DECIMAL(9,8),
    energy DECIMAL(5,4),
    PRIMARY KEY(track_id,
    track_analysis_link),
    FOREIGN KEY(track_id) REFERENCES
    Track(track_id),
    CONSTRAINT check_time_signature
    CHECK (time_signature >= 3
    AND time_signature <= 7),
    CONSTRAINT check_valence
    CHECK(valence >= 0 AND valence <= 1),
    CONSTRAINT check_speechiness
    CHECK(speechiness >= 0
    AND speechiness <= 1),
    CONSTRAINT check_liveness
    CHECK(liveness >= 0
    AND liveness <= 1),
    CONSTRAINT check_key
    CHECK(`key` >= 1 AND `key` <= 11),
    CONSTRAINT check_danceability
    CHECK(`key` >= 0 AND `key` <= 1),
    CONSTRAINT check_acousticness
    CHECK(`key` >= 0 AND `key` <= 1),
    CONSTRAINT check_energy
    CHECK(`key` >= 0 AND `key` <= 1));

CREATE TABLE consists (
    track_id CHAR(23),
    album_id CHAR(23),
    PRIMARY KEY(album_id, track_id),
```

```
    FOREIGN KEY (album_id)
    REFERENCES Album(album_id),
    FOREIGN KEY (track_id)
    REFERENCES Track(track_id));

CREATE TABLE composes (
    artist_id CHAR(23),
    track_id CHAR(23),
    PRIMARY KEY(artist_id, track_id),
    FOREIGN KEY(artist_id)
    REFERENCES Artist(artist_id),
    FOREIGN KEY(track_id)
    REFERENCES Track(track_id));

CREATE TABLE releases (
    artist_id CHAR(23),
    album_id CHAR(23),
    PRIMARY KEY(artist_id, album_id),
    FOREIGN KEY(artist_id)
    REFERENCES Artist(artist_id),
    FOREIGN KEY(album_id)
    REFERENCES Album(album_id));
```

**2.7.4. View Query** These views are necessary as they will be utilized in the web application created to load the appropriate data.

```
CREATE VIEW view_app AS (
    SELECT t1.track_id, t4.artist_name,
    track_name, genre FROM track t1
    INNER JOIN composes t2 ON
    t1.track_id = t2.track_id
    INNER JOIN artist_genres t3
    ON t2.artist_id = t3.artist_id
    INNER JOIN artist t4 ON
    t4.artist_id = t2.artist_id
);

CREATE VIEW view_db AS (
    SELECT track_id, artist_name,
    track_name, track_popularity,
    danceability, energy,
    `key`, loudness, mode,
    speechiness, acousticness,
    instrumentalness, liveness,
    valence, time_signature, genre
    FROM view_app
    INNER JOIN track_analysis_elements
    USING(track_id)
);
```

By creating these views, we simplify the process of accessing and retrieving the required information from the underlying database.

The ”view_app” view consolidates track information along with artist details and genre, providing a comprehensive dataset for the application’s functionalities.

Similarly, the ”view_db” view aggregates track attributes, including popularity, danceability, energy, and other essential metrics, enabling efficient data retrieval for the web application’s operations.

Utilizing views streamlines the data retrieval process within the web application, enhancing its performance and usability for end-users.

## 3. Client Overview

The application serves as a user-friendly platform for managing music preferences and exploring new tracks, utilizing the extensive Spotify dataset. Users can effortlessly navigate through a range of features designed to enhance their music listening experience, including playlist creation, personalized song recommendations, and collaborative playlist generation.

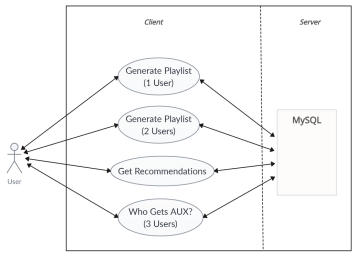


**Figure 2. Use case**

### 3.1. Client Requirements:

**3.1.1. Streamlit:** Streamlit is a user-friendly and intuitive framework designed for data scientists and developers to quickly create interactive web applications directly from Python scripts.

**Installation Steps using pip:** To install Streamlit using pip, follow these simple steps:

1. Open your command-line interface (CLI).
2. Run the following command to install Streamlit:

```
pip install streamlit
```

3. Once installation is complete, you can start using Streamlit to develop and deploy your interactive web applications for data mining and analysis projects.

**Usage of Streamlit:** Streamlit seamlessly integrates with existing data science libraries and workflows, making it a preferred choice for developing interactive data visualization tools, dashboards, and machine learning applications.

**3.1.2. Spotipy :** Spotipy is a Python library that provides easy access to the Spotify Web API. It enables developers to retrieve information about tracks, albums, artists, playlists, and user profiles from the Spotify platform.

**Installation Steps using pip:** To install Spotipy using pip, follow these simple steps:

1. Open your command-line interface (CLI).
2. Run the following command to install Spotipy:

```
pip install spotipy
```

3. Once installation is complete, you can start using Spotipy to interact with the Spotify Web API and access Spotify data within your application.

**Usage of Spotipy:** As a Python library, Spotipy provides an intuitive interface for accessing Spotify data, making it a preferred choice for developers building music-related applications, recommendation systems, and data analysis tools. We utilize Spotipy to generate playlists based on the tracks generated within the application.

## 3.2. User Requirements:

In order to utilize the features of the application, such as playlist generation and collaborative playlist creation, users will need to provide a CSV file containing their listening activity. This CSV file contains valuable data that the application uses to understand the user's music preferences and history, enabling the generation of personalized playlists.

1. **Login to Your Spotify Account:**

   Visit the Spotify website or open the Spotify app on your device. Sign in with your username and password to access your account.

2. **Navigate to Account Settings:**

   Once logged in, navigate to your account settings. This can typically be found by clicking on your profile icon or username, then selecting "Account" or "Settings" from the dropdown menu.

3. **Open Up Privacy Settings:**

   Within your account settings, locate the privacy settings section. This is where you can manage your data and privacy preferences.

4. **Click on Request Data:**

   Look for the option to request your data. In Spotify's privacy settings, there should be a feature that allows you to request a copy of your data, including your listening history.

5. **Verify Your Identity:**

   Follow the prompts to request your data. You may need to verify your identity to ensure the security of your account and personal information.

Once you've completed these steps and requested your data, Spotify will generate a CSV file containing your listening activity. This file will be sent to you via email or made available for download through your Spotify account. You can then use this CSV file to access the features of the application that require your music listening history.

## 3.3. Client Design

The application has been designed and implemented using Streamlit, a powerful Python library for building interactive web applications. Streamlit provides a straightforward and efficient way to create web-based interfaces directly from Python scripts, enabling rapid development and deployment of data-driven applications.

**Justification of Design:** The multipage app structure in Streamlit has been adopted to enhance user experience and streamline navigation within the application. Each page serves a specific functionality, catering to different user needs and tasks. The justification for this design is as follows:

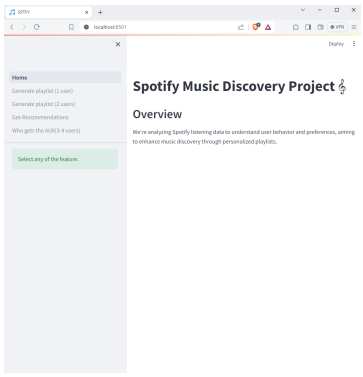


**Figure 3. User Interface**

1. **Home Page (Page 1):**
   The home page serves as the entry point to the application, providing users with an overview of available features and options.
2. **Generate Playlist (1 User) (Page 2):**
   This page is dedicated to generating personalized playlists for individual users based on their listening activity.
3. **Generate Playlist (2 User) (Page 3):**
   Users can collaborate to create playlists by comparing their listening activities. This page facilitates the generation of collaborative playlists for two users.
4. **Get Recommendation (1 User) (Page 4):**
   Users can receive personalized song recommendations based on their current listening activity. This page displays a list of recommended tracks, artists, and their links.
5. **Who Gets the Aux (3 User) (Page 5):**
   Designed for group activities, this page allows participants to vote on the next tracks to be played. The page dynamically updates based on user preferences, ensuring an inclusive music listening experience.

**Feature Pages Design:** Each feature page includes a file uploader component, enabling users to upload their Spotify data in CSV format. Upon uploading the file, users can submit it to the server using a submit button. Depending on the feature selected, the application will either generate a Spotify playlist link or display a list of tracks, artists, and their links accordingly.

This design approach ensures a user-friendly and intuitive interface, allowing users to seamlessly interact with the application and access its various functionalities.

## 3.4. Features

1. **Playlist Generation from CSV**

   This feature offers users the ability to craft personalized playlists based on their individual listening history. By simply uploading a CSV file containing their listening activity, users can leverage the application's sophisticated algorithms to curate a playlist that reflects their unique musical tastes and preferences.

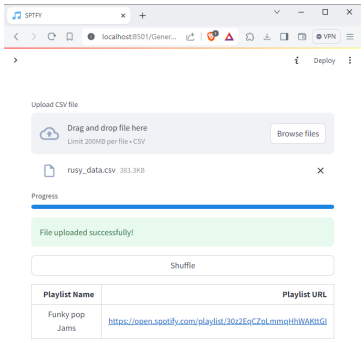


**Figure 4. Feature 1**

Through careful analysis of the provided data, including favorite artists, genres, and frequently played tracks, the application ensures that the generated playlist resonates harmoniously with the user's musical preferences.

**Expected Output:**

Upon uploading the CSV file, the application meticulously processes the data, extracting key insights into the user's music consumption patterns. Subsequently, it compiles a customized playlist comprising a diverse selection of songs tailored to the user's musical preferences and listening habits. This curated playlist serves as a personalized soundtrack, enriching the user's music listening experience. Upon completion, the application generates a Spotify playlist and provides a link for easy access.

2. **Playlist Generation Based on User Similarities**

   This collaborative feature enables users to discover new music and foster shared listening experiences with friends or family. By providing CSV files containing the listening activities of two

users, the application identifies commonalities in their music preferences and generates a collaborative playlist that reflects their shared interests. Through an analysis of overlapping artists, genres, and favorite tracks, the application facilitates the creation of a playlist that resonates with both users, promoting collaborative music exploration and discovery.

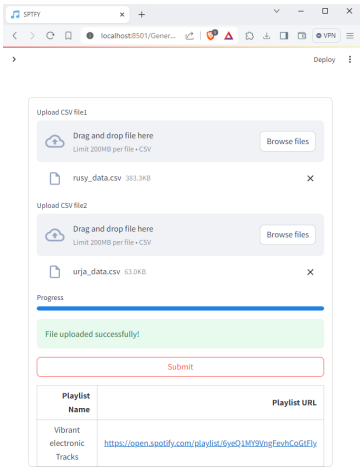


**Figure 5. Feature 2**

**Expected Output:**

Upon submission of the CSV files for both users, the application conducts a comprehensive analysis to identify areas of musical overlap and shared preferences. Leveraging this insight, it compiles a collaborative playlist featuring tracks that align with the musical tastes of both users. Upon completion, users can access the generated Spotify playlist via a provided link, ready to enjoy their shared musical adventure.

3. **Song Recommendations**

Users can discover new music tailored to their individual preferences through personalized song recommendations. By analyzing the user's current listening activity, favorite artists, and genres, the application provides tailored suggestions that align with the user's musical tastes.

**Expected Output:**

Upon request, the application generates a curated list of recommended songs based on the user's music preferences and listening history. These personalized recommendations offer users a diverse selection of tracks that complement

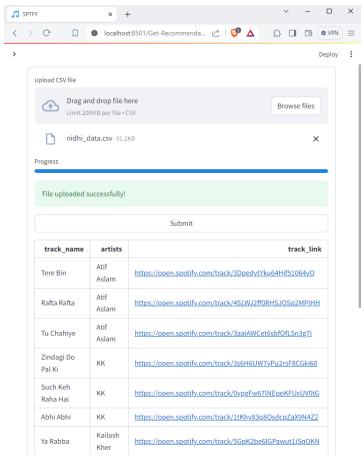


**Figure 6. Feature 3**

their musical tastes and preferences, fostering a rewarding music discovery experience.

4. **"Who Gets the Aux" Feature**

Designed for social settings and group activities, this feature streamlines the process of selecting music by allowing participants to vote on the next tracks to be played.

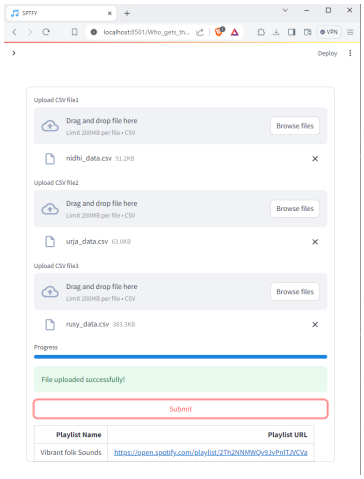


**Figure 7. Feature 4**

**Expected Output:**

During social events and get-togethers, the playlist is modified constantly according to the group's tastes, promoting participation and shared enjoyment. Three users' listening behaviors are used to construct the playlist, which guarantees a varied and inclusive selection that appeals to all listeners. After finishing, users receive a link to the shared playlist, making it easy to include into their music-listening pattern.

### 3.5 Test Cases

| **Test Case Description** | **Expected Result** | **Actual Result** | **Pass/Fail** |
|---|---|---|---|
| App ran in environment where streamlit and spotipy not installed | NameError: library 'spotipy' is not defined | NameError: library 'spotipy' is not defined | Pass |
| App ran in environment where streamlit and spotipy installed | Ran successfully | Ran successfully | Pass |
| For feature 1, csv not provided and clicked on shuffle | Error thrown: Please submit a CSV File< | Error thrown: Please submit a CSV File< | Pass |
| For feature 1, csv provided and clicked on shuffle once | Generated playlist and displayed link | Generated playlist and displayed link | Pass |
| For feature 1, csv provided and clicked on shuffle more than twice | Count incremented and displayed more relevant results | Count incremented and displayed more relevant results | Pass |
| For feature 2, user 1 csv provided, user 2 csv not provided and clicked on submit | Error thrown: Please submit a CSV File for user 2! | ValueError: Invalid file path or buffer object type: <class 'NoneType'> | Fail |
| For feature 3, file uploaded but file type is not csv | Error thrown: Please submit a CSV File! | Error thrown: Please submit a CSV File! | Pass |
| For feature 4, 2 user uploaded csv with correct format but 1 user did not | Error thrown: incorrect or corrupted csv! | KeyError: 'artistName' | Fail |

Table 1: Test Results

# 4. Data Mining

## 4.1. Objective:

In this section, we aim to explore and analyze the user's listening activity to uncover possible new likable songs. By leveraging data mining techniques, we seek to identify patterns and trends within the user's music preferences, ultimately providing personalized recommendations for new songs that align with their tastes. The user's CSV files contain valuable information about the tracks and artists they have been listening to, offering insights into their musical preferences and interests.

By analyzing this data, we can generate recommendations tailored to the user's unique tastes and preferences. We will utilize data mining techniques to analyze the user's listening activity and extract meaningful patterns and associations. By examining factors such as the frequency of track plays, preferred genres, and artists, we aim to identify recurring themes and similarities in the user's music preferences.

## 4.2. Proposed Techniques:

In our exploration of recommendation techniques, we have identified two primary approaches: content-based filtering and collaborative filtering.

**4.2.1. Content-Based Filtering:** Content-based filtering is a recommendation technique that analyzes the attributes or features of items to recommend similar items to the user. In the context of music recommendation, this technique leverages information about the content of songs, such as genre, artist, and track attributes, to suggest songs that are similar to those the user has previously enjoyed.

**Metrics for Similarity:** Within content-based filtering, we have identified two metrics for measuring similarity between songs:

1. **Jaccard Similarity:** Jaccard similarity is a measure of similarity between two sets, calculated as the size of the intersection divided by the size of the union of the sets. In the context of music recommendation, Jaccard similarity can be used to compare sets of attributes such as artist names or genre tags to determine the similarity between songs.

2. **Cosine Similarity:** Cosine similarity is a measure of similarity between two vectors, calculated as the cosine of the angle between them. In content-based filtering, cosine similarity is often used to compare the feature vectors representing songs, where each feature corresponds to attributes such as danceability, energy, key, loudness etc. This metric captures the angle between the feature vectors, indicating their similarity in a multidimensional space.

**4.2.2. Collaborative Filtering:** Collaborative filtering is another popular recommendation technique that relies on user feedback or interactions to make recommendations. Instead of analyzing item attributes, collaborative filtering examines user behavior, such as ratings or preferences, to identify patterns and make personalized recommendations.

In collaborative filtering, recommendations are generated based on similarities between users or items. User-based collaborative filtering compares the preferences of similar users to make recommendations, while item-based collaborative filtering identifies similar items based on user interactions and recommends items that are frequently consumed together.

## 4.3. Feature Selection:

In the process of building recommendation engine, feature selection was conducted to identify the most relevant attributes for generating personalized song recommendations. Through principal component analysis (PCA) analysis, we aimed to reduce the dimensionality of the dataset while preserving the most significant information.

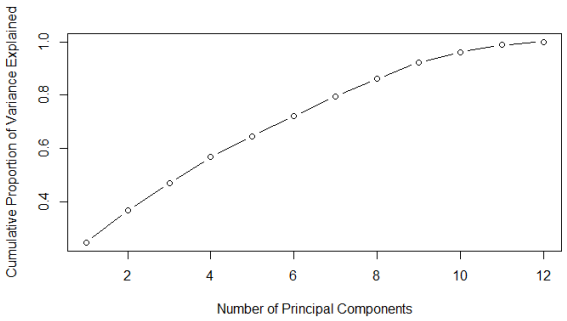


**Figure 8. PCA-analysis**

After performing PCA analysis on the dataset, we identified the following features that contribute most significantly to the variance in the data:

- "danceability"

- ”energy”
- ”key”
- ”loudness”
- ”mode”
- ”speechiness”
- ”acousticness”
- ”instrumentalness”
- ”liveness”
- ”valence”
- ”tempo”
- ”time_signature”

These features were selected based on their ability to capture the most variance in the dataset, indicating their importance in characterizing the musical attributes of songs. It is noteworthy that all of these features contribute equally to the variance in the data, highlighting their significance in our recommendation model.

By considering these selected features in our recommendation algorithm, we aim to ensure that our system effectively captures the diverse musical preferences and characteristics of the user's listening activity, ultimately leading to more accurate and personalized song recommendations.

### 4.4. Choice of Content-Based Filtering over Collaborative Filtering:

In our recommendation system, we opted for content-based filtering over collaborative filtering due to the nature of the available data and the suitability of each approach.

Content-based filtering was deemed the best choice for building our recommendation system as we have access to individual user data. This approach leverages the attributes and characteristics of items (in this case, songs) to make recommendations based on the similarity between items and the user's preferences. In contrast, collaborative filtering relies on similar user interactions and feedback to make recommendations. However, since we do not have sufficient user data available to model user preferences accurately, collaborative filtering may not perform optimally in our scenario.

For content-based filtering, we considered two metrics: Jaccard distance and cosine similarity. While both metrics are commonly used to measure similarity between items, we ultimately chose cosine similarity due to its ability to provide more accurate results in our experiments.

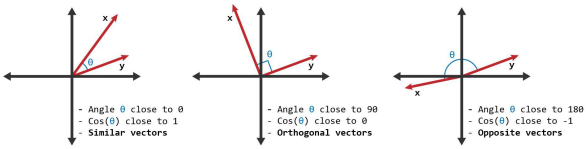


**Figure 9. Cosine Simiarity**

Cosine similarity measures the cosine of the angle between two vectors, representing the similarity between the feature vectors of items. This metric effectively captures the similarity between songs based on their attributes, allowing us to generate more precise recommendations tailored to the user's preferences.

### 4.5. Implementation of Technique:

The implemented technique utilizes a combination of data preprocessing, feature extraction, and similarity computation to generate personalized song recommendations for users based on their listening activity.

**Algorithm:**

1. **Read the User Uploaded CSV:** Read the CSV file uploaded by the user, containing their listening activity data.
2. **Fetch Total Minutes Listened Group by the Artists Played:** Calculate the total minutes listened to each artist by grouping the user's listening activity by artists.
3. **Sort Them Descending:** Sort the total minutes listened to each artist in descending order.
4. **Fetch the Genres Based on the Artists:** Fetch the genres associated with the top artists from a music database or online source.
5. **Fetch Other Artists from Universal Dataset Based on the User Genres:** Retrieve additional artists from a universal dataset based on the genres identified from the user's listening activity.
6. **Get User Playlist or Fetch Top 10 Songs of Each User Artist (User Tracks):** Obtain the user's playlist or fetch the top 10 songs of each artist in the user's listening activity, referred to as user tracks.

7. **Find the Songs Present in Both User Tracks and Universal Dataset:** Identify the songs that are present in both the user tracks and the universal dataset of artists and songs.

8. **Remove User Tracks from the Universal Dataset:** Exclude the user tracks from the universal dataset to avoid recommending songs the user has already listened to.

9. **Find the Mean of User Tracks Grouped by Genre:** Calculate the mean of the user tracks grouped by genre to represent the average preferences of the user within each genre.

10. **Find Cosine Similarity Between Each Song in Universal Dataset and the Mean of the Track for Each Genre:** Calculate the cosine similarity between each song in the updated universal dataset and the mean of the tracks for each genre to determine the similarity between songs and the user's preferences within each genre.

11. **Obtain 20 Track IDs with Highest Similarity:** Select the 20 track IDs with the highest cosine similarity scores as the recommended tracks for the user.

12. **Pass Track IDs to Spotipy to Generate a Playlist:** Pass the selected track IDs to the Spotipy library to generate a playlist.

For feature 2 and 4, there is a slight modification in the algorithm. When fetching each user's genres, we compute the intersection of genres from both users, combined with the union of genres from each user individually. This approach ensures that the recommended songs are based on shared interests between the two users.

### 4.6. Model Validation:

During the validation of our recommendation model, we experimented with different similarity metrics to assess their effectiveness in generating accurate song recommendations.

Initially, we employed the Jaccard index as a similarity metric to measure the similarity between songs based on their attributes. However, during model validation, we observed that the Jaccard index did not perform well in recommending songs to users. The recommendations provided using the Jaccard index were not as accurate or relevant to the user's preferences.

Subsequently, we switched to cosine similarity as our primary similarity metric for calculating the similarity between songs. Through rigorous model validation and testing, we found that cosine similarity significantly improved the performance of our recommendation model. The recommendations generated using cosine similarity were more accurate and aligned with the user's listening preferences, leading to a better overall user experience.

This validation process underscores the importance of thorough testing and experimentation in the development of recommendation systems. By systematically evaluating different approaches and metrics, we can identify the most effective techniques for generating personalized song recommendations and enhance the performance of our recommendation model.

### 4.7. Results:

The implementation of our recommendation algorithm has yielded promising results (that are within the scope of our given dataset available), in generating personalized song recommendations based on user listening activity.

Through extensive testing and evaluation, we observed that our algorithm successfully provided users with relevant and accurate song recommendations tailored to their individual preferences.

However, it is important to note that while our algorithm has shown good performance, there is still room for improvement. One limitation we encountered was the availability and size of the dataset. A larger dataset with more diverse and up-to-date music data could potentially enhance the accuracy and coverage of our recommendations.

## References


[1] Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S. (2007). Collaborative Filtering Recommender Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds).

[2] Pazzani, M.J., Billsus, D. (2007). Content-Based Recommendation Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds).

[3] https://www.learndatasci.com/glossary/cosine-similarity/.