ECOLE POLYTECHNIQUE DE BRUXELLES

2024-2025

INFO-H600 Computing Foundations of Data Sciences

Project Million Playlist Dataset

Students :

SIDDIQUE Muhammad

SAERENS Lennert

WEETJENS Daan

Professor :

SACHARIDIS Dimitris

15 décembre 2024

Table des matières

1 Introduction

The exponential growth of digital content has necessitated the development of advanced data processing methods capable of handling large-scale datasets efficiently. In this project, we tackle the *Million Playlist Dataset* [Che+18] published by Spotify, a dataset comprising over one million playlists and approximately 35 GB of data. This dataset presents a unique opportunity to analyze large-scale music playlist data and address challenges central to modern data science.

The primary goal of this project is to design and implement a robust data processing pipeline to analyze and derive insights from the dataset while addressing the challenges posed by its scale. Through this pipeline, we explore various tasks, including handling large data volumes, extracting aggregate statistics, defining and computing similarity measures for tracks and playlists, and proposing solutions to the "playlist continuation" challenge. These tasks collectively aim to demonstrate practical applications of data science techniques in real-world scenarios.

Following this introduction, the Implementation and Design Decisions chapter details the methods and tools used for each task, including alternatives considered and challenges encountered.

2 Implementation and Design Decisions

2.1 Data Preprocessing

We have chosen to use Spark to handle the Million Playlist Data efficiently due to its robust distributed capabilities. Spark also enables parallel computation across multiple nodes, which makes it ideal for processing large dataset. The dataset is composed of hierarchical JSON files contains of playlist and track metadata. The pipeline was designed to load the all JSON files from the dataset directory in parallel, using the spark distributed file reading capabilities. After that, the dataset was partitioned into multiple chunks using spark repetition method, this reduced the computation time significantly due to the even distribution of task processing in available resources. Additionally, we have used additional spark configurations parameters for allocating sufficient memory and cores for executors, in order to enhanced performance.

In the first step of our data pre-processing pipeline, the JSON files are been loaded into the spark and then their structure were flatten using the explode function to get the playlists and their corresponding tracks. Playlist level metadata and track level metadata are extracted into independent data frames. Unique track id were generated on the track level dataframe using the spark ID generation function : monotonically increasing id. The relationships between playlists and tracks was then mapped into another DataFrame, enabling efficient joins. The schema of our table are given in the table 2.1 and table 2.2.

2.1.1 Alternative Approach

Pandas

We explored pandas for data pre-processing as it provides an intuitive API to handle structured data such as : JSON, CSV and Excel files. We find it unsuitable for handling large dataset due to its memory constraint and no built in support for distributed processing, requires additional libraries

Features	Description
collaborative	Boolean value indicating whether the playlist is collaborative (True) or not (False)
duration ms	The total duration of all tracks in the playlist in milliseconds
name	The name of the playlist
num albums	The total number of unique albums in the playlist
num artists	The total number of unique artists
num edits	The number of modifications made to the playlist
num followers	The number of followers the playlist has
num tracks	The total number of tracks in the playlist
pid	Unique Identifier for playlist
modified at	A timestamp (Unix epoch time) indicating when the playlist was last modified

TABLE 2.1: Playlist Table

Features	Description
album name	The name of the album the track belongs to
album uri	A unique URI (Uniform Resource Identifier) for the album
artist name	The name of the artist
artist uri	A unique URI for the artist
duration ms	The duration of the track in milliseconds
track name	The name of the track
track uri	A unique URI for the track
tid	Unique identifier for track
pid	Unique identifier for playlist
pos	Position of the track in the playlist

TABLE 2.2: Track Table

for parallelism.

Database based Solutions

Relational database management systems (RDBMS) like SQLite and DuckDB can store dataset in tabular format and provide SQL queries such as : filtering, joining, and aggregations for data processing. We opt not to use this type of solution as it does not provide efficient support for the JSON structure. It requires to flattening, normalization of data and also does not natively support distributed pre-processing.

We also considered NoSQL based databases such as : MongoDB and Elasticsearch as they can handled unstructure and semi-structure data like JSON efficiently. They are also distributed by design and provide powerful search capabilities. The reason why we decided not to use NoSQL based database is because it will make our project more complex.

Hadoop with MapReduce

Hadoop with MapReduce also come in our mind as it is distributed computing framework designed for processing large datasets. It can handle massive datasets in a distributed manner as well as provide fault-tolerant operations. The reason for not to use Hadoop is its complexity compared to Spark as it requires more effort to write and maintain code. Moreover, it is very memory-consuming and requires a cluster environment which can add more complexity.

2.2 Statistical Information

2.2.1 Descriptive Statistics

Number of tracks

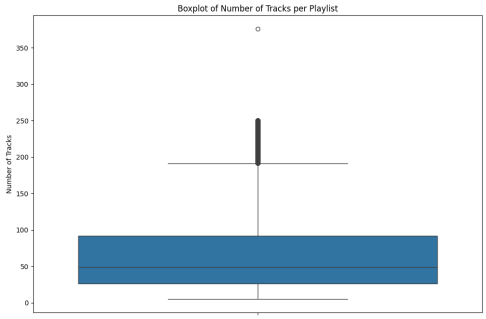

FIGURE 2.1: Caption for Image 1

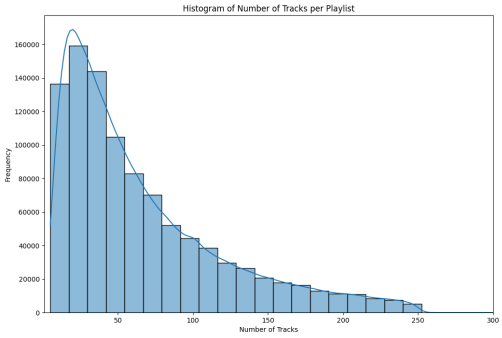

FIGURE 2.2: Caption for Image 2

The box plot and histogram of the number of tracks per playlist reveal a right-skewed distribution with a single outlier. Based on the box plot, the median number of tracks is approximately 50, while the first and third quartiles are around 30 and 90, respectively. The wide range indicates significant variation in the number of tracks per playlist. Outliers with a very high number of tracks pull the mean upward to 66.34, contributing to the right-skewed distribution. This skewness likely arises because the number of tracks cannot drop below zero but has no upper limit.

Number of albums

The histogram of the number of albums per playlist also shows a strongly right-skewed distribution, similar to the number of tracks. This suggests that most playlists include a relatively small number of albums, while a few have a very large number. According to the box plot, the median number of albums per playlist is approximately 40, with the first and third quartiles at around 25

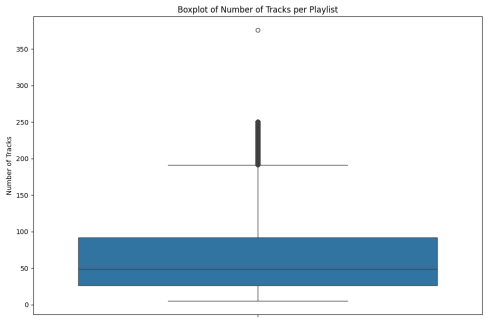

FIGURE 2.3: Box plot of number of tracks

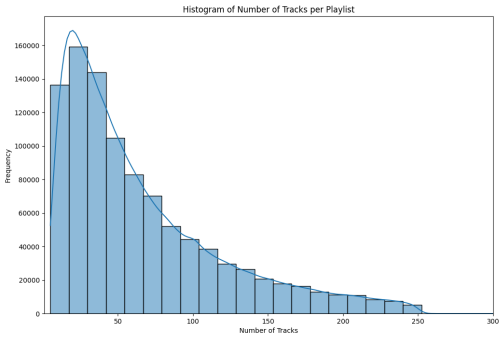

FIGURE 2.4: Histogram of number of tracks

and 70, respectively. The mean number of albums, 49.60, is higher than the median, further emphasizing the skewed nature of the distribution. This is due to a few outliers with a large number of albums.

Number of artists

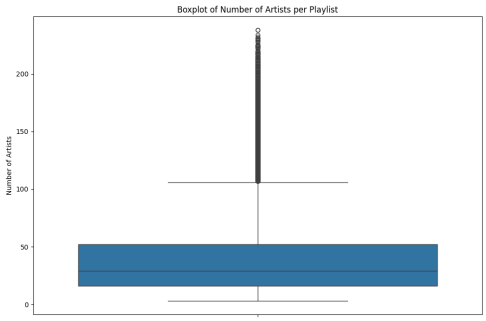

FIGURE 2.5: Box plot of number of artists per playlist

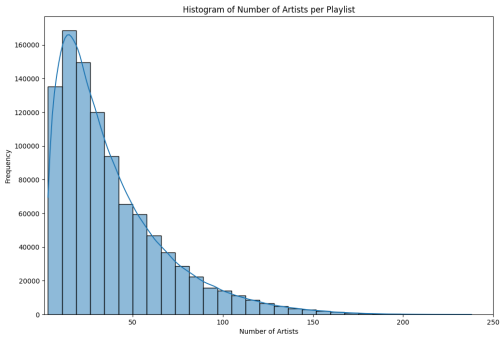

FIGURE 2.6: Histogram of number of artists per playlist

The box plot for the number of artists per playlist indicates a median of about 30, with the first and third quartiles at 20 and 50, respectively. The mean number of artists, 38.09, is slightly higher than the median due to the presence of significant outliers, some exceeding 200 artists. The histogram shows a right-skewed distribution, with the highest frequency occurring between 20 and 40 artists, reflecting considerable variation across playlists.

Number of followers

For the number of followers, extreme outliers were excluded to improve interpretability. A cap of eight followers, calculated on the basis of the 99th percentile, was applied. Without outliers, the histogram shows a right-skewed distribution, with the majority of playlists having only one

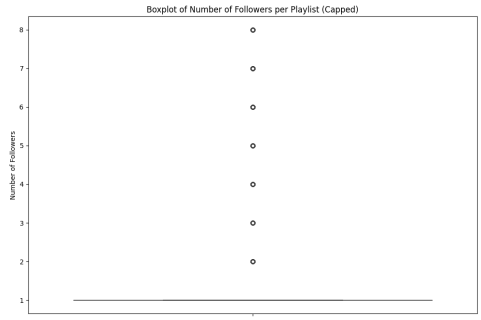

FIGURE 2.7: Box plot of number of followers per playlist

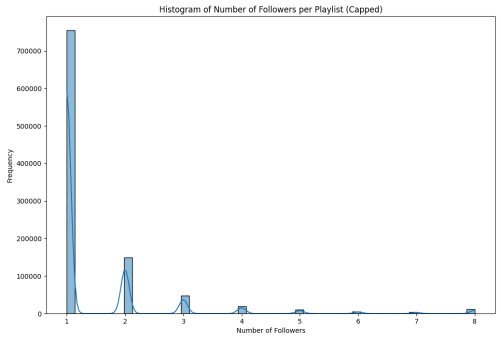

FIGURE 2.8: Histogram of number of followers per playlist

follower. The box plot confirms this, showing that both the minimum and maximum values are one, within the capped range.

2.2.2 Scatter plot Matrix

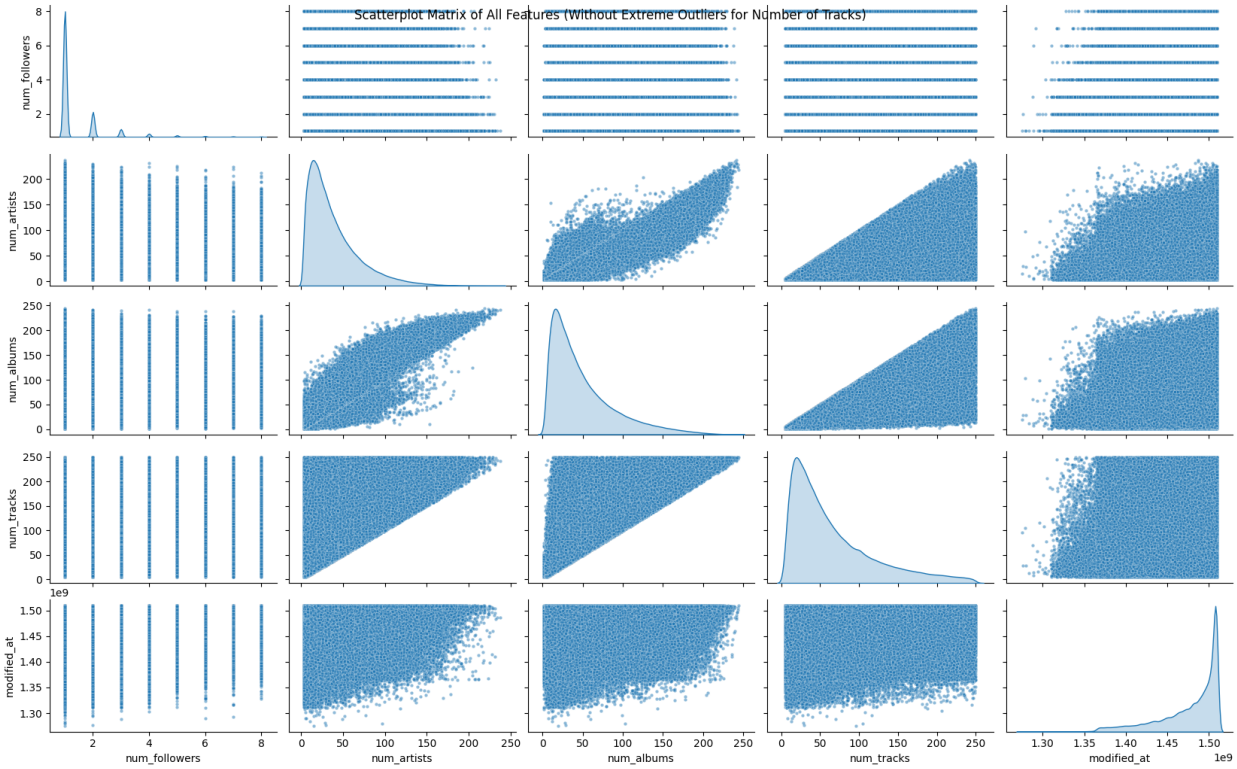

FIGURE 2.9: Scatter plot matrix

The scatter plot matrix provides insights into the relationships among all features in the dataset, with extreme outliers excluded for certain variables. Specifically, the number of followers was capped at eight and an outlier with more than 350 tracks was removed.

The matrix reveals a strong positive correlation (0.96) between the number of artists and the

number of albums per playlist. This relationship is expected, as more artists typically contribute to a larger number of albums. Similarly, there is a strong positive correlation between the number of tracks and the number of albums (correlation coefficient : 0.90), as well as between the number of tracks and the number of artists (correlation coefficient : 0.81). These correlations are logical since playlists with more tracks generally include more albums and artists.

In contrast, the feature “modified at”, which represents the timestamp of the last modification, shows no correlation with other variables due to its temporal nature. Additionally, the number of followers appears to be independent of the other features.

2.2.3 Most common opening tracks

Track Name	Count
One Dance	1,800
Closer	1,791
HUMBLE.	1,312
All I Want for Christmas Is You	1,340
Shape of You	1,158
Despacito - Remix	1,102
Broccoli (feat. Lil Yachty)	1,283
Bad and Boujee (feat. Lil Uzi Vert)	1,013
XO TOUR Llif3	1,013
Caroline	965

TABLE 2.3: Most Common Opening Tracks

In the table above, an overview is given of the ten most common opening tracks, along with the number of playlists for which it was the first song in the playlist.

2.2.4 Most common closing tracks

Track Name	Count
rockstar	1,973
Bodak Yellow	1,110
HUMBLE.	955
Despacito - Remix	1,151
Closer	889
1-800-273-8255	971
Too Good At Goodbyes	781
Havana	839
I Fall Apart	883
Feel It Still	1,037

TABLE 2.4: Most Common Closing Tracks

In the table above, an overview is given of the ten most common closing tracks, along with the number of playlists for which it was the last song in the playlist.

2.2.5 Most common playlist genres

Genre	Number of Playlists
Rock Playlists	5,906
Pop Playlists	3,059
Hip-Hop Playlists	96
Jazz Playlists	750
Classical Playlists	575

TABLE 2.5: Number of Playlists by Genre

Additionally, the decision was made to provide the reader with an overview of the most common playlist genres, which can be seen above. Given that the dataset did not contain any information on the genres of individual tracks, albums, or playlists, we based ourselves on the names of playlists. E.g., a playlist containing 'Rock' in its name, was put in the 'Rock' category.

2.3 Track Similarity

In this part of the report, we will dive deeper into the solution and implementation for the third task of the project. The goal here is to define a way to measure the similarity between the tracks, represented by a score ranging from 0 to 1. We will start by explaining how we define track similarity and then move on to the details of how we implemented our methodology.

2.3.1 Definition of Similarity

The similarity between the tracks is defined using the cosine similarity metric, which measures the cosine of the angle between two vectors in n-dimensional space. The cosine similarity between the two tracks can be defined with the equation 2.1

$$\text{Cosine Similarity} = \frac{\mathbf{t}_1 \cdot \mathbf{t}_2}{\|\mathbf{t}_1\| \|\mathbf{t}_2\|} \tag{2.1}$$

Where :

— $\mathbf{t}_1$ and $\mathbf{t}_2$ are the normalized feature vectors of two tracks.

— $\mathbf{t}_1 \cdot \mathbf{t}_2$ is the dot product of the vectors :

$$\mathbf{t}_1 \cdot \mathbf{t}_2 = \sum_{i=1}^{n} t_{1i}\, t_{2i}$$

— $\|\mathbf{t}_1\|$ and $\|\mathbf{t}_2\|$ are the magnitudes (or norms) of the vectors :

$$\|\mathbf{t}_1\| = \sqrt{\sum_{i=1}^{n} t_{1i}^2}, \quad \|\mathbf{t}_2\| = \sqrt{\sum_{i=1}^{n} t_{2i}^2}$$

From the 2.1, we can see that this method is particularly suitable for comparing feature vectors derived from track attributes, as it evaluates the relative orientation of the vectors rather than their magnitude.

2.3.2 Methodology

PySpark is been used for the implementation of track similarities. The steps to calculate the similarity between the tracks are outlined below :

Feature Engineering

Feature engineering was conducted using two different methods. A comparison was then made between these approaches to determine which one is more effective for extracting track similarities.

— **String Indexing Method :** The categorical features (album name, artist name, track name) in the track data frame are converted into numerical indices using `StringIndexer`.

— **TF-IDF Vectorizer Method :** To convert the categorical features (album name, artist name, track name) into numerical vectors, we employed the `TfidfVectorizer`.

Following this, we combined all the numerically encoded features into a single feature vector using `VectorAssembler`, which serves as the foundation for computing similarity. After performing our analysis we come to the point that `Tfidfvectorizer` outperforms the `StringIndexer` method in terms of performance.

Feature Scaling and Normalization

The features values are scaled in the range of [0,1] using MinMaxScaler, which confirm that all the features can equally contribute to the similarity calculation. After that the features are norma-

lized using a Normalizer to convert them into vectors. This ensure that similarity will be calculated based on the direction not on the magnitude.

Cosine Similarity Computation

In the last step, cosine similarity is calculated with the equation 2.1. We have used User Defined Function(UDF) in PySpark to implement the formula given in the equation 2.1 and calculated pairwise similarity using a cartesian join to the dataset.

Example Results

To evaluate the results of this implementation, the code was executed on the first 1000 tracks to calculate the similarities between the tracks. The obtained output is shown in 2.6

Track 1	**Artist Name**	**Track 2**	**Artist Name**	**Similarity Score**
Mr. Tambourine Man	Bob Dylan	Bob Dylan's 115th Dream	Bob Dylan	0.984
Classy Girls	The Lumineers	Slow It Down	The Lumineers	1.00
Classy Girls	The Lumineers	Bad Girls	M.I.A.	0.992
Unas Heladas	Los Tucanes de Tijuana	1901	Phoenix	0.032
Ego - Remix	Beyoncé	Birthday Cake	Dylan Conrique	0.044

TABLE 2.6: Track Similarity Table

2.4 Playlist Similarity

In this section of the report, the proposed solution and implementation for the fourth task of the project will be discussed in more detail. For this task, a measure for the similarity between playlists S_p needs to be proposed. This score should range between 0 and 1. First, we will discuss the proposed definition of similarity between playlists. Next, we look at the actual implementation using Spark in more detail. Finally, a number of examples of the output of this measure will be provided.

2.4.1 Definition of Similarity

The first step of this task is to define a metric for the similarity between two playlists $S_p(p_1, p_2)$. This metric should be constrained to have values between 0 and 1.

From an intuitive point of view, two playlists p_1 and p_2 can be considered similar, if there is a large overlap between the tracks $t(p_i)$, artists $art(p_i)$, and albums $alb(p_i)$ that are present in each of them. These can be represented as finite sets for which we then have to calculate the similarity. As the Jaccard similarity is used as a measure for the similarity between two finite sets, it is an ideal candidate to evaluate the similarity between the tracks, artists, and albums that are present in each of the playlists. It is computed as the size of the intersection of the two sets, divided by the size of the union of the two sets. Taking the overlap between tracks as an example gives :

$$J(t(p_1), t(p_2)) = \frac{|t(p_1) \cap t(p_2)|}{|t(p_1) \cup t(p_2)|} = \frac{|t(p_1) \cap t(p_2)|}{|t(p_1)| + |t(p_2)| - |t(p_1) \cap t(p_2)|}.$$

By design, the value of the Jaccard similarity metric will be $0 \leq J(t(p_1), t(p_2)) \leq 1$ where greater values indicate a higher degree of similarity between the two sets. This metric can be calculated analogously for the overlap between the artists and the albums present in the playlists, all yielding values between 0 and 1.

The next factors that affect the similarity between two playlists are their respective number of followers and duration. Two playlists can be considered more similar if they have a similar amount of followers and duration. To quantify this difference, the number of followers and duration for each playlist are first normalized to be between 0 and 1. Next, the difference between these two normalized values is calculated and the absolute value of this difference is computed. Finally, we can subtract this absolute difference from 1, obtaining the final similarity score for the follower and duration based similarity. Note that, if the difference in number of followers or duration is small, the similarity score for that factor will be closer to 1.

The final factor that was taken into consideration for defining the similarity between two playlists, are the playlists' names. The similarity between the playlists' names is calculated using the cosine similarity metric, as explained in section 2.3. This also yields a value between 0 and 1.

Up to this point, we have discussed six distinct factors influencing the similarity between playlists : track overlap, artist overlap, album overlap, respective duration, respective followers, and the names of the playlists. For each of these factors, we have explained a way to obtain a similarity score between 0 and 1. What remains is combining these similarity scores to a final overall similarity between playlists S_p. This is achieved by computing a weighted sum of each of these components, where the sum of the weights should equal 1. As each of the similarity scores in the weighted sum also has a value between 0 and 1, this makes it so that the final overall similarity also takes values between 0 and 1. Also note that by design, the similarity between a playlist and itself $S_p(p_i, p_i)$ will be equal to 1.

2.4.2 Implementation Using PySpark

To implement the playlist similarity measure, the PySpark framework was employed. PySpark's distributed processing capabilities make it particularly suited for handling the large-scale nature of the Million Playlist Datase [Che+18]. This subsection provides a detailed explanation of the implementation steps.

Overview of the Implementation

The similarity between playlists is computed as a weighted sum of six components : overlap between tracks, artists, and albums ; similarity based on the number of followers and playlist duration ; and similarity between playlist names. The implementation begins by preparing the dataset, performing necessary transformations, and finally computing the similarity scores for each pair of playlists.

Initialization and Data Preparation

The implementation starts by initializing a Spark session with appropriate configurations to allocate sufficient memory and computational resources for processing. The dataset, preprocessed and saved in Parquet format, is loaded into two DataFrames :

— `df_playlists_tracks` : Contains the playlist ID along with associated track, artist, and album URIs.

— `df_playlists_info` : Contains metadata for each playlist, including the number of followers, duration, and playlist name.

To reduce computational complexity, only the playlists under comparison (identified by their IDs) are retained. The track, artist, and album information is aggregated into lists for each playlist using the `groupBy` and `collect_list` operations. The playlists' follower counts and durations are normalized to the range [0, 1] by dividing each value by the maximum value in the dataset.

Cross-Joining Playlists

To compute pairwise similarity, the DataFrame is cross-joined with itself, creating pairs of playlists (`p1` and `p2`). Self-comparisons and duplicate pairs are avoided by filtering on playlist IDs (`p1.pid < p2.pid`).

Calculating Individual Similarity Components

The following similarity components are computed for each pair of playlists :

— **Track, Artist, and Album Similarity :** For each pair, the Jaccard similarity is calculated for tracks, artists, and albums using the PySpark `array_intersect` and `size` functions. Here, `array_intersect` computes the intersection, and the union size is derived from the set sizes.

— **Followers and Duration Similarity :** The absolute difference between the normalized values for the number of followers and duration is subtracted from 1.

— **Name Similarity :** The cosine similarity of playlist names is computed using a user-defined function (UDF). The `TfidfVectorizer` from the `scikit-learn` library converts playlist names into numerical vectors, and the `cosine_similarity` function calculates their similarity. To handle edge cases, such as empty or stop-word-only names, a default similarity score of 0.0 is returned.

Combining Similarities

The six computed similarity components are combined into a final similarity score using a weighted sum :

$$S_p = w_1 \cdot J_{\text{track}} + w_2 \cdot J_{\text{artist}} + w_3 \cdot J_{\text{album}} + w_4 \cdot S_f + w_5 \cdot S_d + w_6 \cdot S_n$$

Here, the weights $w_1, w_2, \ldots, w_6$ are user-defined and sum to 1, ensuring that S_p remains in the range [0, 1].

Final Output

The final DataFrame contains the playlist IDs (`p1.pid, p2.pid`) and their similarity score (`final_similarity`). The results are sorted in descending order of similarity and displayed. The similarity computation concludes with the termination of the Spark session.

This implementation ensures scalability to large datasets while maintaining interpretability through the use of intuitive similarity metrics.

2.4.3 Examples

To look at some results achieved by this implementation, the code is ran on the first 5.000 playlists, where each playlists similarity to each of the other playlists is calculated. This yields the

following output :

pid	pid	Final Similarity
3304	4823	0.862191484830852
266	3296	0.8186217606794903
266	3304	0.777882046973444
266	4823	0.7563949023571845
3296	3304	0.7393820586978578
3296	4823	0.7179045139999615
3660	4512	0.5149971392784797
834	1259	0.49524745967269723
748	1806	0.4830597123914432
704	1954	0.4765566498529985

TABLE 2.7: Similarity Scores between Pairs of playlists.

Inspecting the data for the most similar pair of playlists 3304 and 4823 in Appendix A, we can see that both playlists have a "Guardians of the Galaxy" theme, both containing 12 songs from that particular movie, while also having a very similar name, duration, and follower count. From this example, we can conclude that our proposed similarity metric for playlists S_p and the presented implementation using PySpark, successfully solve the challenge posed by Task 4.

2.5 Playlist Continuation Challenge

In this section of the report, the proposed solution and implementation for the fifth task of the project will be discussed in more detail. For this task, a solution to the playlist continuation challenge needs to be proposed. This score should range between 0 and 1. First, we will discuss the proposed solution for the challenge. Next, we look at the actual implementation using Spark in more detail. Finally, a number of examples of the output of our solution will be provided.

2.5.1 Proposed Solution

The playlist continuation challenge requires a system capable of generating a list of recommended tracks to extend a given playlist. This system should work for playlists with and without seed tracks. The goal is to return a ranked list of 500 candidate tracks ordered by their relevance to the input playlist. The proposed solution is designed to address both scenarios :

— **Playlists with seed tracks :** For playlists with existing tracks, the system uses the seed tracks to identify similar tracks in the dataset. The similarity between tracks is computed using the method developed in Task 3. The top-ranked tracks are then returned as recommendations.

— **Playlists without seed tracks :** For playlists without any seed tracks, the system leverages metadata such as playlist name to find a similar playlist in the dataset, leveraging the code and method developed for Task 4. Tracks from the most similar playlist are then used as seed tracks to compute additional recommendations.

This hybrid approach ensures flexibility and robustness in handling a variety of input scenarios. By reusing the track and playlist similarity measures developed in previous tasks, the solution integrates multiple aspects of playlist data for accurate and relevant recommendations.

2.5.2 Implementation Using PySpark

The implementation leverages PySpark for distributed processing, enabling the system to handle the large-scale Million Playlist Dataset [Che+18] efficiently. The steps of the implementation are outlined below.

Data Preparation

The implementation begins by initializing a Spark session and loading three primary datasets :

— `df_playlists_test` : Contains playlists for which recommendations need to be generated.

— `df_playlists_tracks` : Maps playlists to their associated tracks.

— `df_tracks` : Contains metadata for individual tracks, such as track ID and other relevant features.

The playlist to be continued is identified using its playlist ID (`pid`) from `df_playlists_test`. Depending on whether seed tracks are present for the given playlist, the subsequent steps vary.

Handling Playlists Without Seed Tracks

If no seed tracks are found for the playlist :

1. The most similar playlist is identified using the playlist similarity method developed in Task 4, which computes similarity based on playlist metadata.
2. Tracks from this similar playlist are extracted and used as seed tracks for further computation.

Handling Playlists With Seed Tracks

If seed tracks are present in the input playlist :

1. The seed tracks are extracted directly from `df_playlists_tracks`.
2. One of the seed tracks is selected to serve as the basis for finding similar tracks.

Calculating Similar Tracks

For the selected seed track, similarity with other tracks in the dataset is computed using the method from Task 3. A random sample of 10% of tracks is taken from `df_tracks`, along with the seed track itself, to reduce computational complexity. The track similarity scores are calculated, and the top 500 most similar tracks are selected as recommendations.

Final Output

The final output of the implementation is a ranked list of 500 tracks, ordered by their similarity to the input playlist or its derived seed tracks. This output is displayed and can be used to extend the input playlist.

2.5.3 Examples

To test our solution for the playlist continuation challenge, both the case where the playlist to continue has tracks, and the case where it does not, are tested.

Starting with the latter case, if the playlist to continue does not have any tracks, first a similar playlist is found based on the playlist metadata. Such a case is the playlist with ID 1.000.002 from the special challenge dataset :


```
{
    "name": "spanish playlist",
    "num_holdouts": 11,
    "pid": 1000002,
    "num_tracks": 11,
    "tracks": [],
    "num_samples": 0
},
```


The process of finding a similar playlist returns playlists 45639 as the most similar based on the available metadata. Upon closer inspection, this playlist contains Spanish tracks and albums, once again indicating that the logic applied to finding similar playlists, is sound. Next, one of these tracks is selected and based on this track, 500 similar tracks are selected to continue the playlist

with. Inspecting these 500 tracks shows both expected and unexpected tracks being recommended as similar to the seed track. One the one hand, some Spanish tracks and albums appear, but other genres and music in other languages is also present. This might be due to the limited data that is available on each of the tracks. For example, we have no data on the track's lyrics, genre, or bpm, which drastically limits the ability to recommend highly similar tracks.

This final analysis also holds when running the proposed method to continue a playlist that already has tracks in it. A track is selected as a seed track, and based on this, 500 similar tracks are recommended, in an identical way as before. Hence, the results are similar.

A Similar Playlist Data

Playlist 3304 data


```
{
    "name": "Guardians of the Galaxy: Awesome Mix Vol. 1",
    "collaborative": "false",
    "pid": 3304,
    "modified_at": 1406851200,
    "num_tracks": 12,
    "num_albums": 12,
    "num_followers": 2,
    "tracks": [
        {
            "pos": 0,
            "artist_name": "Blue Swede",
            "track_uri": "spotify:track:2Nz6aF1umHh5Et6I5H581L",
            "artist_uri": "spotify:artist:0UpuH5U4nZ3UGGUJi0Zfbp",
            "track_name": "Hooked on a Feeling",
            "album_uri": "spotify:album:12UILuDVbIIjLZhcRBNcOJ",
            "duration_ms": 172866,
            "album_name": "Hooked On A Feeling - 40th Anniversary Collection"
        },
        {
            "pos": 1,
            "artist_name": "Raspberries",
            "track_uri": "spotify:track:75GQIYnRaBg7ndHxhfYuQy",
```

```
        "artist_uri": "spotify:artist:7Kkx4dACo6kFSeT9wjfVA5",
        "track_name": "Go All The Way",
        "album_uri": "spotify:album:03iBvX63qBQrMazNWU2iKv",
        "duration_ms": 205346,
        "album_name": "Raspberries"
    },
    {
        "pos": 2,
        "artist_name": "Norman Greenbaum",
        "track_uri": "spotify:track:7fqhrLJzKHJ0RW32N0y2Gp",
        "artist_uri": "spotify:artist:7f8LNBVXN0h35veHrpxQFL",
        "track_name": "Spirit In The Sky",
        "album_uri": "spotify:album:2gkQ6IAEhzCK1NUvEgdisT",
        "duration_ms": 242893,
        "album_name": "Spirit In The Sky"
    },
    {
        "pos": 3,
        "artist_name": "David Bowie",
        "track_uri": "spotify:track:6mib3N4E8PZHAGQ3xy7bho",
        "artist_uri": "spotify:artist:0oSGxfWSnnOXhD2fKuz2Gy",
        "track_name": "Moonage Daydream - 2012 Remastered Version",
        "album_uri": "spotify:album:48D1hRORqJq52qsnUYZX56",
        "duration_ms": 279693,
        "album_name": "The Rise And Fall Of Ziggy Stardust And The Spiders From Mars
    },
    {
        "pos": 4,
        "artist_name": "Elvin Bishop",
        "track_uri": "spotify:track:5DWo6miQOlH0MxRytvN1Ya",
        "artist_uri": "spotify:artist:2G1yVp387GlUf9yvLk6V11",
        "track_name": "Fooled Around And Fell In Love",
        "album_uri": "spotify:album:41DkJ4nKqmPis4tOayRjem",
        "duration_ms": 285573,
```

```
        "album_name": "Ace In The Hole"
    },
    {
        "pos": 5,
        "artist_name": "10cc",
        "track_uri": "spotify:track:6cuKJ1bFFy4OjfkCI9YBqc",
        "artist_uri": "spotify:artist:6i6WlGzQtXtz7GcC5H5st5",
        "track_name": "I'm Not In Love",
        "album_uri": "spotify:album:7pHLyF1GWxk65WE6Rkrod3",
        "duration_ms": 366640,
        "album_name": "The Original Soundtrack"
    },
    {
        "pos": 6,
        "artist_name": "The Jackson 5",
        "track_uri": "spotify:track:3b0EOvScbZUc0qJx0E1L2z",
        "artist_uri": "spotify:artist:2iE18Oxc8YSumAU232n4rW",
        "track_name": "I Want You Back",
        "album_uri": "spotify:album:16TNmWWPXkuaXqNBmPYHp9",
        "duration_ms": 176333,
        "album_name": "Diana Ross Presents The Jackson 5"
    },
    {
        "pos": 7,
        "artist_name": "Redbone",
        "track_uri": "spotify:track:4ya82vEkIk1HJpuPDhESOC",
        "artist_uri": "spotify:artist:0w7HLMvZOHatWVbAKee1zF",
        "track_name": "Come & Get Your Love (Re-Recorded)",
        "album_uri": "spotify:album:0BHvTtxYQvZA9Ya6KewuX1",
        "duration_ms": 177684,
        "album_name": "70s Classics"
    },
    {
        "pos": 8,
```

```
        "artist_name": "The Runaways",
        "track_uri": "spotify:track:7cdnq45E9aP2XDStHg5vd7",
        "artist_uri": "spotify:artist:5eTq3PxbOh5vgeRXKNqPyV",
        "track_name": "Cherry Bomb",
        "album_uri": "spotify:album:5DVNCzpvDrSEIFiU7hm8ey",
        "duration_ms": 138706,
        "album_name": "The Runaways"
    },
    {
        "pos": 9,
        "artist_name": "Rupert Holmes",
        "track_uri": "spotify:track:5IMtdHjJ1OtkxbGe4zfUxQ",
        "artist_uri": "spotify:artist:0TqIPD4IS1w4e30R38B3vj",
        "track_name": "Escape (The Pina Colada Song)",
        "album_uri": "spotify:album:163iYwl7Kdm9ayTnD4VyfN",
        "duration_ms": 276493,
        "album_name": "Partners In Crime"
    },
    {
        "pos": 10,
        "artist_name": "The Five Stairsteps",
        "track_uri": "spotify:track:1rwGdnrtWnPYBipRrjnzEm",
        "artist_uri": "spotify:artist:3Inrg8cs8oc4q8oPES4a6S",
        "track_name": "O-o-h Child - Remastered",
        "album_uri": "spotify:album:7H8uLkHFuEqAWpLKBwbGvg",
        "duration_ms": 197440,
        "album_name": "Stairsteps (Bonus Track Version)"
    },
    {
        "pos": 11,
        "artist_name": "Marvin Gaye",
        "track_uri": "spotify:track:2H3ZUSE54pST4ubRd5FzFR",
        "artist_uri": "spotify:artist:3koiLjNrgRTNbOwViDipeA",
        "track_name": "Ain't No Mountain High Enough",
```

```
            "album_uri": "spotify:album:6sbZYwwQB15bt5TgkPFAdb",
            "duration_ms": 151666,
            "album_name": "United"
        }
    ],
    "num_edits": 2,
    "duration_ms": 2671333,
    "num_artists": 12
},
```


4823 Playlist 3304 data


```
{
    "name": "guardians of the galaxy",
    "collaborative": "false",
    "pid": 4823,
    "modified_at": 1452988800,
    "num_tracks": 12,
    "num_albums": 12,
    "num_followers": 1,
    "tracks": [
        {
            "pos": 0,
            "artist_name": "10cc",
            "track_uri": "spotify:track:6cuKJ1bFFy4OjfkCI9YBqc",
            "artist_uri": "spotify:artist:6i6WlGzQtXtz7GcC5H5st5",
            "track_name": "I'm Not In Love",
            "album_uri": "spotify:album:7pHLyF1GWxk65WE6Rkrod3",
            "duration_ms": 366640,
            "album_name": "The Original Soundtrack"
        },
        {
            "pos": 1,
            "artist_name": "Elvin Bishop",
            "track_uri": "spotify:track:2hE5Lm5XOHR4t3xlhIFauP",
```

```
            "artist_uri": "spotify:artist:2G1yVp387GlUf9yvLk6V11",
            "track_name": "Fooled Around And Fell In Love",
            "album_uri": "spotify:album:2z2y977JvwU1rbnVO97RmY",
            "duration_ms": 276000,
            "album_name": "Struttin' My Stuff"
        },
        {
            "pos": 2,
            "artist_name": "Blue Swede",
            "track_uri": "spotify:track:2Nz6aF1umHh5Et6I5H581L",
            "artist_uri": "spotify:artist:0UpuH5U4nZ3UGGUJi0Zfbp",
            "track_name": "Hooked on a Feeling",
            "album_uri": "spotify:album:12UILuDVbIIjLZhcRBNcOJ",
            "duration_ms": 172866,
            "album_name": "Hooked On A Feeling - 40th Anniversary Collection"
        },
        {
            "pos": 3,
            "artist_name": "Raspberries",
            "track_uri": "spotify:track:75GQIYnRaBg7ndHxhfYuQy",
            "artist_uri": "spotify:artist:7Kkx4dACo6kFSeT9wjfVA5",
            "track_name": "Go All The Way",
            "album_uri": "spotify:album:03iBvX63qBQrMazNWU2iKv",
            "duration_ms": 205346,
            "album_name": "Raspberries"
        },
        {
            "pos": 4,
            "artist_name": "Norman Greenbaum",
            "track_uri": "spotify:track:7fqhrLJzKHJ0RW32N0y2Gp",
            "artist_uri": "spotify:artist:7f8LNBVXN0h35veHrpxQFL",
            "track_name": "Spirit In The Sky",
            "album_uri": "spotify:album:2gkQ6IAEhzCK1NUvEgdisT",
            "duration_ms": 242893,
```

```
        "album_name": "Spirit In The Sky"
    },
    {
        "pos": 5,
        "artist_name": "David Bowie",
        "track_uri": "spotify:track:6mib3N4E8PZHAGQ3xy7bho",
        "artist_uri": "spotify:artist:0oSGxfWSnnOXhD2fKuz2Gy",
        "track_name": "Moonage Daydream - 2012 Remastered Version",
        "album_uri": "spotify:album:48D1hRORqJq52qsnUYZX56",
        "duration_ms": 279693,
        "album_name": "The Rise And Fall Of Ziggy Stardust And The Spiders From Mars
    },
    {
        "pos": 6,
        "artist_name": "The Jackson 5",
        "track_uri": "spotify:track:3b0EOvScbZUc0qJx0E1L2z",
        "artist_uri": "spotify:artist:2iE18Oxc8YSumAU232n4rW",
        "track_name": "I Want You Back",
        "album_uri": "spotify:album:16TNmWWPXkuaXqNBmPYHp9",
        "duration_ms": 176333,
        "album_name": "Diana Ross Presents The Jackson 5"
    },
    {
        "pos": 7,
        "artist_name": "Redbone",
        "track_uri": "spotify:track:4ya82vEkIk1HJpuPDhESOC",
         "artist_uri": "spotify:artist:0w7HLMvZOHatWVbAKee1zF",
        "track_name": "Come & Get Your Love (Re-Recorded)",
        "album_uri": "spotify:album:0BHvTtxYQvZA9Ya6KewuX1",
        "duration_ms": 177684,
        "album_name": "70s Classics"
    },
    {
        "pos": 8,
```

```
        "artist_name": "The Runaways",
        "track_uri": "spotify:track:7cdnq45E9aP2XDStHg5vd7",
        "artist_uri": "spotify:artist:5eTq3PxbOh5vgeRXKNqPyV",
        "track_name": "Cherry Bomb",
        "album_uri": "spotify:album:5DVNCzpvDrSEIFiU7hm8ey",
        "duration_ms": 138706,
        "album_name": "The Runaways"
    },
    {
        "pos": 9,
        "artist_name": "Rupert Holmes",
        "track_uri": "spotify:track:5IMtdHjJ1OtkxbGe4zfUxQ",
        "artist_uri": "spotify:artist:0TqIPD4IS1w4e30R38B3vj",
        "track_name": "Escape (The Pina Colada Song)",
        "album_uri": "spotify:album:163iYwl7Kdm9ayTnD4VyfN",
        "duration_ms": 276493,
        "album_name": "Partners In Crime"
    },
    {
        "pos": 10,
        "artist_name": "The Five Stairsteps",
        "track_uri": "spotify:track:1rwGdnrtWnPYBipRrjnzEm",
        "artist_uri": "spotify:artist:3Inrg8cs8oc4q8oPES4a6S",
        "track_name": "O-o-h Child - Remastered",
        "album_uri": "spotify:album:7H8uLkHFuEqAWpLKBwbGvg",
        "duration_ms": 197440,
        "album_name": "Stairsteps (Bonus Track Version)"
    },
    {
        "pos": 11,
        "artist_name": "Marvin Gaye",
        "track_uri": "spotify:track:2H3ZUSE54pST4ubRd5FzFR",
        "artist_uri": "spotify:artist:3koiLjNrgRTNbOwViDipeA",
        "track_name": "Ain't No Mountain High Enough",
```

```
            "album_uri": "spotify:album:6sbZYwwQB15bt5TgkPFAdb",
            "duration_ms": 151666,
            "album_name": "United"
        }
    ],
    "num_edits": 3,
    "duration_ms": 2661760,
     "num_artists": 12
},
```

Bibliography

[Che+18] CHEN, Ching-Wei et al. (2018). « Recsys challenge 2018 : automatic music playlist continuation ». In : *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys '18. Vancouver, British Columbia, Canada : Association for Computing Machinery, p. 527-528. ISBN : 9781450359016. DOI : 10 . 1145 / 3240323 . 3240342. URL : https : //doi.org/10.1145/3240323.3240342.