CDS513: Predictive Business

AnalyticsAcademic Session:

Semester II, 2022/20223

School of Computer Science

Prediction of musical preference and popularity: (utilizing Market BasketAnalysis, Regression, and Recommendation System methodologies).Group 13

Name	Matric No
Liban Bile Mohamud	P-COM0040/22
Habeb Abdullah Saleh Saleh Alwajeeh	P-COM0122/22
Ahmed Adel Sanhan AL-Haidary	P-COM0113/22

Group contribution

Name of the member	Task handled
Liban Bile Mohamud	Problem Background, problem statement, Methodology, project objectives, scope, and Regression. Model
Habeb Abdullah Saleh Saleh Alwajeeh	Literature review and project motivation, Data exploration, and Market Basket Analysis.
Ahmed Adel Sanhan AL-Haidary	abstract, Dataset description, attributes explanation, Data processing, Recommendation System, and Conclusion.

Contents

Table of Figures

Table List

Abstract

With the introduction of streaming services like YouTube, Soundcloud, and Spotify, which have become the favorite channels for music lovers, the music industry has seen considerable change. This shift has provided record companies and musicians with an unprecedented opportunity to reach a wider audience and market their music effectively. However, catering to the diverse musical preferences of individual listeners presents a complex challenge. The use of data science has become vital in understanding changing consumer preferences and maximizing revenue. By analyzing user data, including explicit and implied comments, artists can make informed decisions about the promotion of songs, the timing of the release, and personalized recommendations. This project utilizes market basket analysis, regression, and recommendation systems to predict music taste and popularity based on Spotify datasets, leveraging the vast amount of data generated by online music streaming platforms. By leveraging user data and applying advanced analytics, record companies and musicians can gain valuable insights, enhance user experiences, and optimize their business strategies. The application of data science methodologies in the music industry has the potential to enhance user satisfaction and drive growth in the online streaming sector.

1.0 Problem background

The reality that consumers' musical preferences have changed significantly over time cannot be denied. Before a few decades ago, the only ways to buy records were on cassette tapes and compact discs, which could only be found at a few stores. For the purpose of engaging with their audience directly, artists would travel abroad. There is presently a change in this situation. The top streaming platforms for fans of music to listen to their favourite bands and artists are now widely recognized as being YouTube, SoundCloud, and Spotify. These platforms give users the option to pay for ad-free, premium services, making them a great way to learn about consumer listening habits. Record companies and musicians may reach more people and grow their followings thanks to these channels.

Additionally, by lowering entrance barriers, social media platforms significantly contribute to the dissemination of musical content by emerging performers. Contrarily, data and the use of data science and music analytics to track what the audience is listening to help music agencies stand out from the crowd and differentiate themselves from their rivals.

The size of the music industry is without a doubt vast. The recorded music industry will generate $23.1 billion in total sales in 2020, according to Statista.com. Streaming services produced $56 percent of this total revenue, which came to $11.9 billion globally (Statista.com, 2020). Many artists and groups are vying for notoriety and recognition, which will probably result in financial success. More admirers are attracted to them as their impact grows. Global Internet access has led to changes in how individuals listen to music. In the past, a musician's success was mostly based on the volume of LPs or CDs (physical music sales) that were sold. Internet streaming, which started with websites similar to Napster and has now become the most popular method of listening to music.

However, the music business is quickly getting more cutthroat. There is constant pressure to create the next massive success. Even if it should be a top priority to produce high-quality music, it is useless to do so if no one will ever hear it. As a result, the industry is shifting its attention to something more concrete in the hopes that doing so will help it better grasp how the general public's musical tastes are changing and increase its ability to make money.

One of the datasets that will be used for this project comes from Spotify. Daniel Ek and Martin Lorentzon founded the Swedish audio streaming and media services company Spotify on April 23, 2006. With more than 381 million monthly active users, including 172 million paying members, as of September 2021, it is the largest music streaming service provider in the world (Wikipedia.com,2022). With the use of a computer, smartphone, or other device, users of Spotify can access millions of songs from different record companies remotely. In order to categorize songs internally and propose new music to consumers, Spotify gives each song thirteen unique properties or features. The majority of these qualities are numerical numbers, but they also include category information.

Spotify is a data-driven music streaming service that compiles enormous amounts of user data, using machine learning and business analytics to generate custom music depending on the user's musical interests, then collects and analyses this data to improve its services. Spotify's rise to prominence as the leading online music streaming service was made possible by the use of business analytics and machine learning.

1.1 Problem issue/s and problem statement

At times, the music industry can be a cruel and heartless dictator. It can provide the artist with stability and solace at times, and then strip them of everything the next. Artists and musicians have the potential for a long and successful career in their respective disciplines if they play the game well, compose the ideal tunes, and plan strategically. Additionally, it has the potential to expose the general public to a vast array of works of art. It is essential for artists to identify a need and desire among people regarding their musical preferences, but this can be difficult due to the fact that people's musical preferences vary.
As a result of the proliferation of online music streaming services, there is an abundance of music consumer data. With all of this new information, distributors such as Spotify and Apple Music can provide artists with audience demographics and location data to assist them in understanding their core audience.

The data obtained from the users can be used to determine which songs to promote and where, the optimal release time for a song, what song to recommend for a specific user based on the implicit and explicit data of the user, and the features and characteristics of big hit songs, among other things, all of which can aid music creators in making good decisions. Data science is used to determine which tracks, genres, or artists will appeal to the greatest number of people in today's music industry.

1.2 Project objectives and motivation

Using business analytics and machine learning techniques in the online streaming music industry can dramatically improve consumer satisfaction, attract new customers, and increase the music industry's revenue by a significant amount. Online music streaming generates enormous quantities of data daily, which can be used to rapidly determine user trends in music preferences. The transformation of vast quantities of data into useful information can provide and present valuable insights. This results in improved user recommendation decisions based on their musical preferences. This project's primary objective is outlined below.

1. To determine the association between various songs by analyzing the user’s song preference pattern.

2. To build a song recommender system for users based on their past preferences, and to predict what users will enjoy listening based on their similarity to other users.

3. To predict the popularity of a song based on specific features of the songs.

.

2.0 Literature review

The music industry, particularly in the last ten years, has benefited greatly from the use of big data. Up to 80% of the Recording Industry Association of America's revenue comes from paid memberships, which have over 60 million subscribers, and streaming. Spotify, Apple Music, Pandora, as well as YouTube and other services, track data.

An aggregator called Chartmetric is able to estimate which of the almost 1 million musicians would most likely make the Billboard 200 charts using a patented algorithm in addition to displaying which artist is now dominating the charts [1].

A data mining approach called market basket analysis is used to determine client preferences from their purchases [2]. One of the biggest entertainment firms in Southeast Asia, Astro Malaysia Holdings, offers each customer a certain sort of channel in a bundle that includes channels in a particular genre. Customers were unable to have variety in a single package as a result, and they had to pay more to include different genres for themselves. Dahlan (2015) performed market basket analysis on the purchasing strategies and catered to the customer preferences in a paper titled "Analysing Customer Preferences for Astro Using Market Basket Analysis" which ultimately assisted the marketing team in repackaging their channels [3].

Regression analysis determines the link between two dependent variables in a recommender system, which is able to forecast a user's preference for a particular product's rating [4].

A customised music recommendation system was created by the author Chen for his paper titled "A Music Recommendation System Based on Music and User Grouping." Six features were then retrieved to categorise the various tracks after initially analysing the various tracks. The user's past was then examined to create user profiles. On the basis of the users' preferences for the music groups and the user group it belonged to, content-based, statistically based, or collaborative recommendation approaches were presented [5].

This social influence on song popularity may be captured by the work of Koenignstein, Shavitt, and Zilberman, which forecasts billboard success based on peer-to-peer networks. This team employed numerous regression and classification techniques, and they were quite thorough in their work [6].

3.0 Scope

The selected data set, which contains more than one million records about users and their song preferences, contains crucial information about song details and their components. Using this data set, we will determine how market basket analysis, recommender systems, and regression analysis can be implemented within an online music platform to capitalize on the music industry's growth. This project's scope is limited to the analysis of only these four machine learning methods, and the analysis would include applying the data to the models to obtain a prediction and analyzing the performance of the models accordingly, before compiling the results.

4.0 Methodology

A data science-based project must always follow a set of steps in order to be effective. The underlying structure of every activity is made up of these actions. We will use the best CRISP-DM methodology for our project. CRISP-DM (CRoss Industry Standard Process for Data Mining) is a widely adopted framework for conducting data science projects. First introduced in 1999, it remains the most popular choice for organizing and executing such projects. CRISP-DM offers a clear and intuitive depiction of the data science life cycle, effectively outlining the workflow in data-focused endeavors [7]. Figure 1 illustrates the project framework which will follow CRISP-DM methodology.

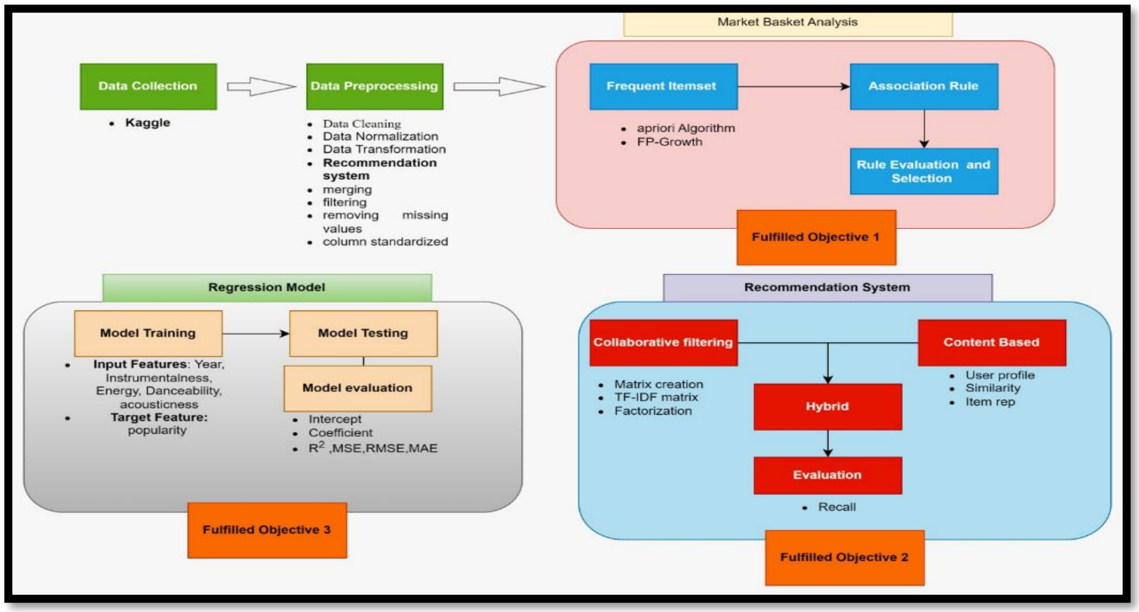


Figure 1: project framework

Project steps are as follows:
I. Data Understanding: In this phase we focus on identifying, collecting, and analyzing relevant data sets. It begins with the collection and loading of initial data into the analysis tool. The data is then described in terms of its properties, such as format, record count, and field identities. Further exploration of the data is conducted to identify relationships and patterns, while data quality is assessed and documented.
II. Data Preparation: The Data Preparation phase, often referred to as "data munging," is a critical step that consumes a significant portion of project effort. It encompasses tasks such as selecting the appropriate data sets and justifying their inclusion or exclusion. Cleaning the data by correcting errors, inputting missing values, and removing irrelevant data is a lengthy but necessary process. New attributes may be derived from the data. Lastly, the data is formatted as needed for analysis.
III. Modeling: The Modeling phase is considered the most exciting aspect of data science. It involves building and assessing various models using different techniques. The selection of modeling techniques and algorithms is determined, and a test design is generated to evaluate the models' performance. Models are built based on the chosen techniques, and their results are assessed against predefined success criteria, domain knowledge, and the test design.
IV. Evaluation: In the Evaluation phase, the focus shifts to assessing the models and determining their suitability for meeting the business objectives. The results are evaluated against the established success criteria, and a thorough review of the entire process is conducted. Any overlooked steps or errors are identified and corrected if necessary. Based on the evaluation, decisions are made regarding the next steps, whether it involves deployment, further iteration, or initiating new projects.

The objective of a data science project is of utmost importance as it serves as a guide and defines the purpose of the project. It provides a clear direction and desired outcome, ensuring that all efforts and resources are aligned toward achieving a specific goal. By establishing a well-defined objective, stakeholders can focus their efforts and make informed decisions throughout the project lifecycle.
Furthermore, the objective of a data science project contributes to the decision-making process. It influences the selection of data sources, variables, and analytical techniques that are most relevant to the desired outcome. This ensures that the project remains focused on extracting meaningful insights and patterns from the available data, ultimately leading to actionable results. We have identified three objectives, which are detailed in the sections below, that describe how each will be implemented.

Objective 1: Market Basket Analysis The first objective of this paper is to conduct a market basket analysis (MBA) using frequent itemset mining and association analysis techniques. The purpose is to identify co-occurrence patterns and relationships among music tracks that are frequently listened to together by customers. By applying MBA, we aim to gain insights into customer preferences and identify commonly listened to items.

Objective 2: Recommendation Systems The second objective is to build and evaluate different recommendation systems. This includes content-based filtering, collaborative filtering, and hybrid approaches. The goal is to provide personalized music recommendations to users based on their preferences and behaviors. We aim to leverage user-item interactions, track characteristics, and collaborative patterns to generate accurate and diverse recommendations. By evaluating the performance of these recommendation systems using metrics such as recall and hit rate, we aim to identify the most effective approach for providing relevant and engaging music recommendations.

Objective 3: Regression Analysis The third objective is to perform a regression analysis to explore the relationship between song attributes and popularity. We aim to analyze the impact of variables such as the year of release, loudness, instrumentals, energy, danceability, and acoustics on the popularity of music tracks. By employing multiple regression analysis, K-Nearest Neighbors Regressor (KNNR), and Support Vector Regression (SVR), we aim to uncover significant factors influencing the popularity of songs. This analysis will provide valuable insights for understanding customer preferences.

5.0 Data description

We use 2 datasets related to Spotify company. The first dataset is a comprehensive dataset obtained from Kaggle [8]. They collect the dataset using Spotify Web API, consisting of over 160,000 songs. This dataset, sourced directly from Spotify, encompasses songs released between the years 1921 and 2020, with each year featuring the top 100 songs. The dataset contains 19 attributes that provide insights into the musical characteristics and popularity of the songs. The dataset contains quantitative and categorical features. Table 1 shows the features of the dataset and their descriptions.

Table 1: Spotify dataset description

Attributes	Description	Data Type
acousticness	Ranges from 0 to 1 The index of track (Arbitrarily ordered)	Decimal
artists	List of artists mentioned on the song	Nominal
danceability	The list of artists credited for production of the track tag duration_ms sort The relative measurement (Ranges from 0 to 1)	Decimal

duration_ms	The song duration in millisecond	Integer
energy	The energy of the track, Ranges from 0 to 1	Decimal
explicit	The binary value whether the track contains explicit content or not, (0 = No explicit content, 1 = Explicit content)	Integer
id	The primary identifier for the track, generated by Spotify 169909 unique values	Nominal
instrumentalness	The relative ratio of the track being instrumental, (Ranges from 0 to 1)	Decimal
key	All keys on octave encoded as values ranging from 0 to 11, starting on C as 0, C# as 1 and so on	Integer
liveness	The relative duration of the track sounding as a live performance, (Ranges from 0 to 1)	Decimal
loudness	Relative loudness of the track in decibel (dB), (Float typically ranging from -60 to 0)	Decimal
mode	The binary value representing whether the track starts with a major (1) chord progression or a minor (0)	Integer
name	The title of the track (132940 unique values)	Nominal
popularity	The popularity of the song lately (Ranges from 0 to 100 with 100 is the most popular)	Integer
release_date	Date of release mostly in yyyy-mm-dd format	Nominal
speechiness	The relative length of the track containing any kind of human voice, (Ranges from 0 to 1)	Decimal
tempo	The tempo of the track in Beat Per Minute (BPM), (Range from 0 to 150)	Decimal
valence	The positiveness of the track, (Ranges from 0 to 244)	Decimal
year	The release year of track, (Ranges from 1921 to 2020)	Integer

The second dataset is Spotify playlists which were collected from Kaggle [9]. This dataset is derived from a subset of users in the #nowplaying dataset who share their #nowplaying tweets through Spotify. It has the data for more than 12 million user IDs. This dataset focuses on users, their playlists, and the tracks with artist names that are included in those playlists. We will use this data set to perform the Market basket analysis and recommendation system analysis. Table 2 shows the features of the dataset and their descriptions.

Table 2: Spotify playlist dataset

Attributes	Description	Data Type
user_id	Hash of the user's Spotify username (more than 12,000,000 user)	Nominal
artistname	Name of the artist (290003 Unique values)	Nominal
trackname	Title of the track (2036738 Unique value)	Nominal
playlistname	Name of the playlist that contains this track.	Nominal

6.0 Data Exploration

Figure 2 illustrates the popularity of various songs, with "death bed (coffee for your head)" emerging as the most popular. This is followed by "supalonely," which, while not as popular as the former, still holds a significant position in the chart. The song "ily" is also featured, ranking third in popularity. The chart provides a clear visual representation of the relative popularity of these top songs, allowing for easy comparison.

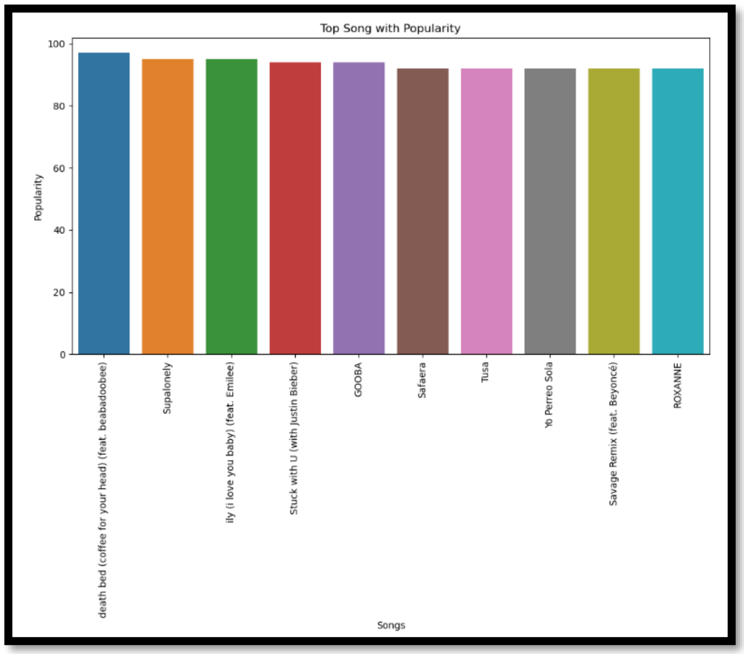


Figure 2: Top 10 songs with popularity.

Figure 3 presents a graphical representation of the top 10 artists based on their popularity. The Beatles hold the premier position, indicating they are the most popular among the artists represented. The Rolling Stones follow in second place, demonstrating a high level of popularity, albeit not surpassing The Beatles. Bob Dylan is positioned third, further underscoring his significant popularity, albeit less than the two preceding artists. The remaining artists, while not explicitly mentioned, also hold positions within this top 10 ranking, each with their respective levels of popularity. This chart provides a clear visual depiction of the relative popularity of these artists, facilitating an easy comparison of their respective standings.

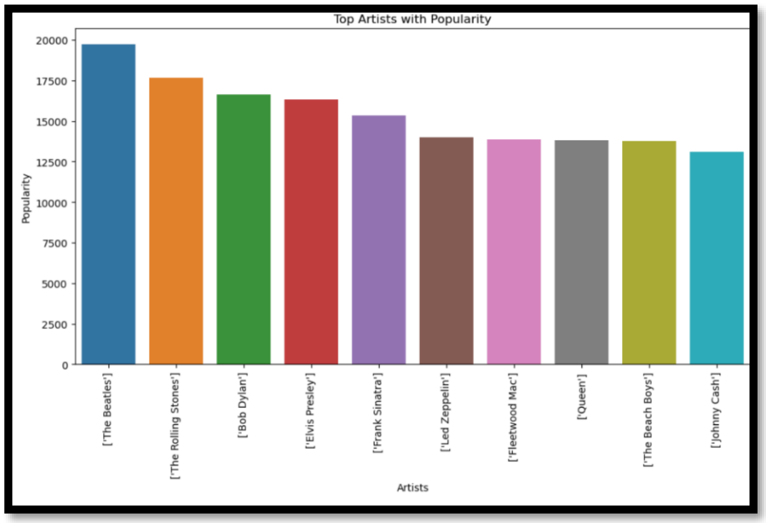


Figure 3: Top 10 Artist with popularity.

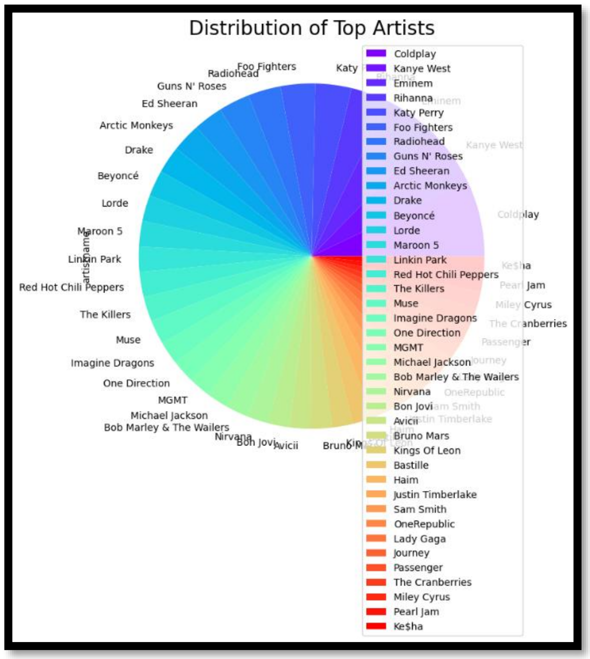


Figure 4: Top 40 artists

Figure 4 shows a pie chart displaying the top 40 artists according to the distribution of their dataset count. Each slice of the pie represents a different artist, and the size of each slice corresponds to the number of various artists. The legend contains the name of each artist as a means of identification.

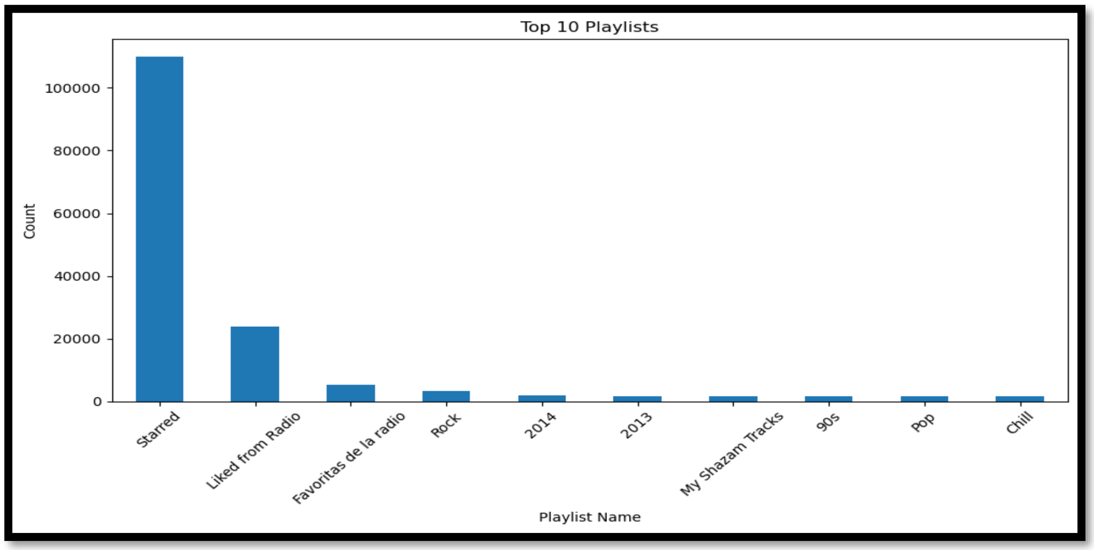


Figure 5: Top ten playlists in the dataset.

Figure 5 shows a bar graph displaying the top ten playlists in the dataset. Each bar represents a different playlist, and the height of each one corresponds to the quantity of occurrences for each playlist. The x-axis shows the playlist names, the y-axis shows the count, and the title gives a broad overview of the chart.

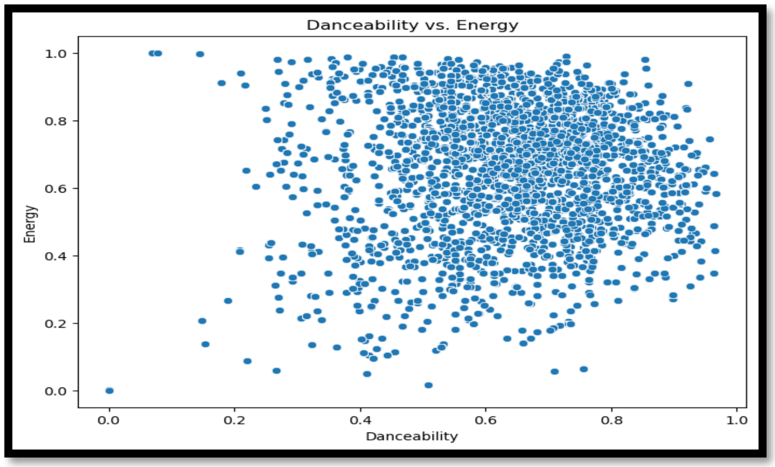


Figure 6: Scatter plot

figure 6 shows a scatter plot demonstrating how danceability and energy relate to one another in the dataset. a scatter plot demonstrating how danceability and energy relate to one another in the dataset. Each song will be represented by a point on the scatter plot. Songs with greater danceability scores will be plotted higher on the plot, while songs with higher energy scores will be plotted further to the right. The more similar the danceability and energy scores of the two songs are, the closer two points are together. Songs with comparable danceability and energy scores can be found using the scatter plot. Consider music that is plotted towards the upper right corner of the plot, for instance, if you're looking for a song with a lot of danceability and enthusiasm.

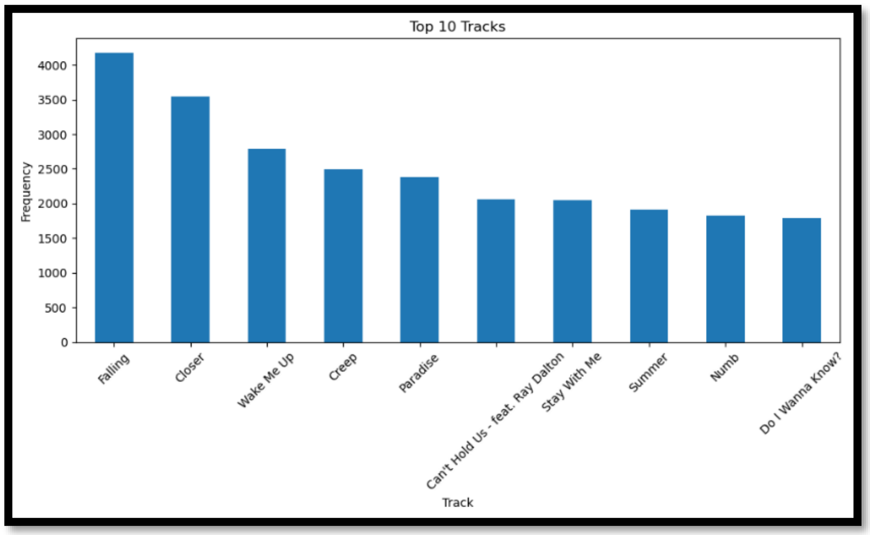


Figure 7. top 10 tracks in terms of frequency.

The figure 7 presented offers a comprehensive overview of the top listening tracks, providing valuable insights into their popularity measured by the number of listens. Notably, "Falling" emerges as the frontrunner with an impressive listener count exceeding 4000, underscoring its widespread appeal and ability to resonate with a broad audience. Additionally, "Closer" secures the second position, amassing approximately 3500 listeners, further attesting to its substantial popularity and significant engagement.

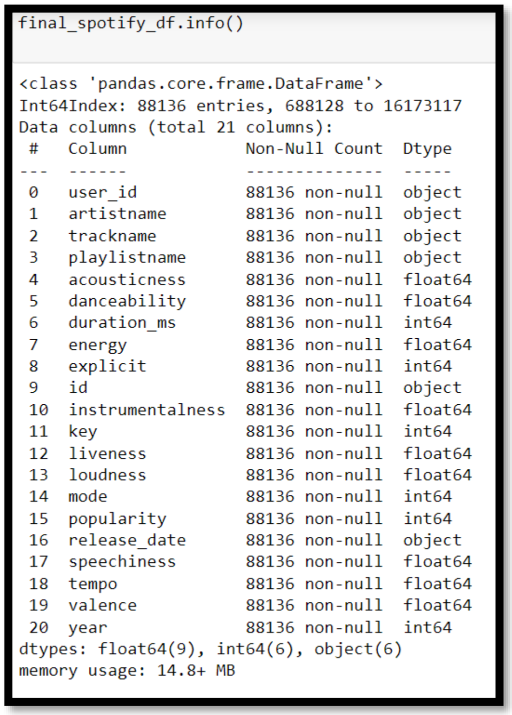
7.0 Data Pre-Processing

To ensure data accuracy and consistency, several data pre-processing steps were performed. First, the column names were cleaned and standardized. The strip() and replace() functions were used to remove any unnecessary spaces or quotation marks from the column names. Additionally, the column name 'name' was replaced with 'track name' to provide a clearer label.

Next, missing values in the dataset were addressed. The isnull().sum() function was used to identify missing values in each column. Rows containing missing values were then removed using the dropna() function, ensuring data completeness and reliability.

To perform deep analysis, we merged both the datasets that contain the user and the track information. The merge operation was performed using the merge() function, with the 'track name' column serving as the key for merging. An inner join was applied to retain only the rows with matching track names in both datasets, resulting in a merged dataset.

Further filtering was applied to focus on the most famous tracks with a popularity score above 80. This filtering criterion allowed for a more targeted analysis of popular tracks.
Through these data processing steps, the raw data was transformed into a clean and reliable dataset, free from missing values and aligned with the specific objectives of the study. The resulting dataset serves as a solid foundation for further analysis and insights into user preferences and music popularity.



```
final_spotify_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 88136 entries, 688128 to 16173117
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   user_id            88136 non-null  object
 1   artistname         88136 non-null  object
 2   trackname          88136 non-null  object
 3   playlistname       88136 non-null  object
 4   acousticness       88136 non-null  float64
 5   danceability       88136 non-null  float64
 6   duration_ms        88136 non-null  int64
 7   energy             88136 non-null  float64
 8   explicit           88136 non-null  int64
 9   id                 88136 non-null  object
 10  instrumentalness   88136 non-null  float64
 11  key                88136 non-null  int64
 12  liveness           88136 non-null  float64
 13  loudness           88136 non-null  float64
 14  mode               88136 non-null  int64
 15  popularity         88136 non-null  int64
 16  release_date       88136 non-null  object
 17  speechiness        88136 non-null  float64
 18  tempo              88136 non-null  float64
 19  valence            88136 non-null  float64
 20  year               88136 non-null  int64
dtypes: float64(9), int64(6), object(6)
memory usage: 14.8+ MB
```

Figure 8: Cleaned Dataset

8.0 Experiment and Analysis

8.1 Market Basket Analysis

Market basket analysis (MBA) is a data mining technique that examines co-occurrence patterns to assess the degree to which products are related when they are bought together. Additionally, we use the terms frequent itemset mining and association analysis. By figuring out how the relationships between the things that customers purchase may be understood, it makes use of patterns that are common in any retail environment. Simply said, market basket analysis assists retailers in keeping certain commodities always in stock by letting them know about the products that are commonly purchased together [10].

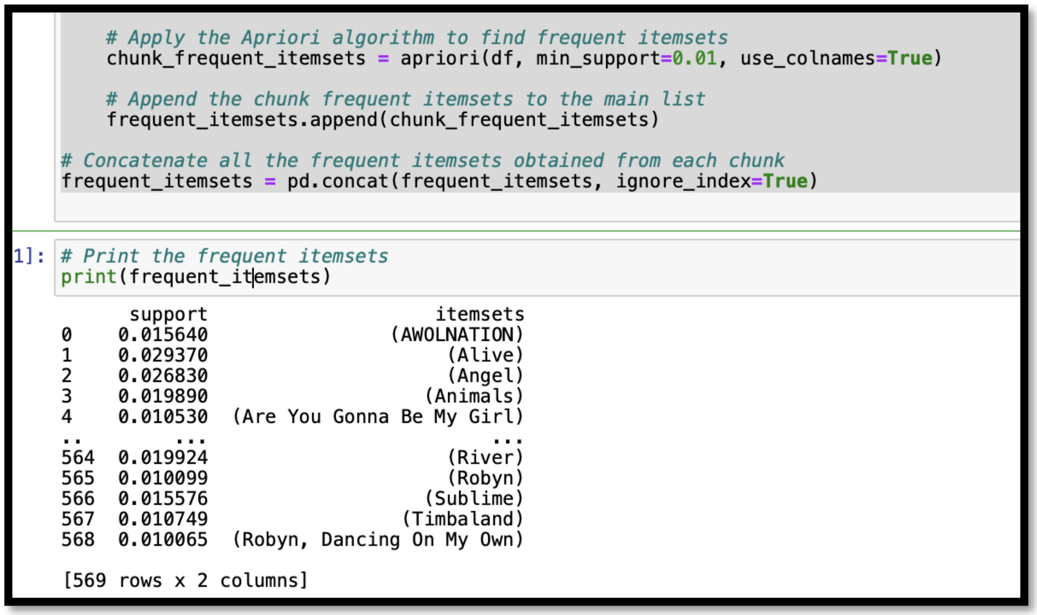
Market basket analysis makes use of numerous methods and algorithms. Predicting the likelihood that clients will purchase many goods at once is one of the key goals.

8.1.1 Apriori Algorithm

To understand the rules of associations and find common items sets, Apriori relies on an algorithm that runs in Relational Data Bases. It will continue to move forward until those sets of items show more often in the database, by identifying frequently occurring individual items and expanding them to larger or larger sets. In order to identify relationships which, draw attention to general trends of the database, Apriori's frequent item sets are available; they can be used in fields like market basket analysis [11].

8.1.2 FP Growth

The initial itemsets of the database are used to create the tree-like Frequent Pattern Tree structure. The FP tree's goal is to mine the most prevalent pattern. The FP tree's nodes each correspond to a component of the itemset. While the lower nodes reflect the itemsets, the root node is a representation of null. The association between the nodes and the lower nodes, or the itemsets and the other itemsets, is preserved while the tree is formed [12].



```
    # Apply the Apriori algorithm to find frequent itemsets
    chunk_frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

    # Append the chunk frequent itemsets to the main list
    frequent_itemsets.append(chunk_frequent_itemsets)

# Concatenate all the frequent itemsets obtained from each chunk
frequent_itemsets = pd.concat(frequent_itemsets, ignore_index=True)
```

```
1]: # Print the frequent itemsets
print(frequent_itemsets)
```

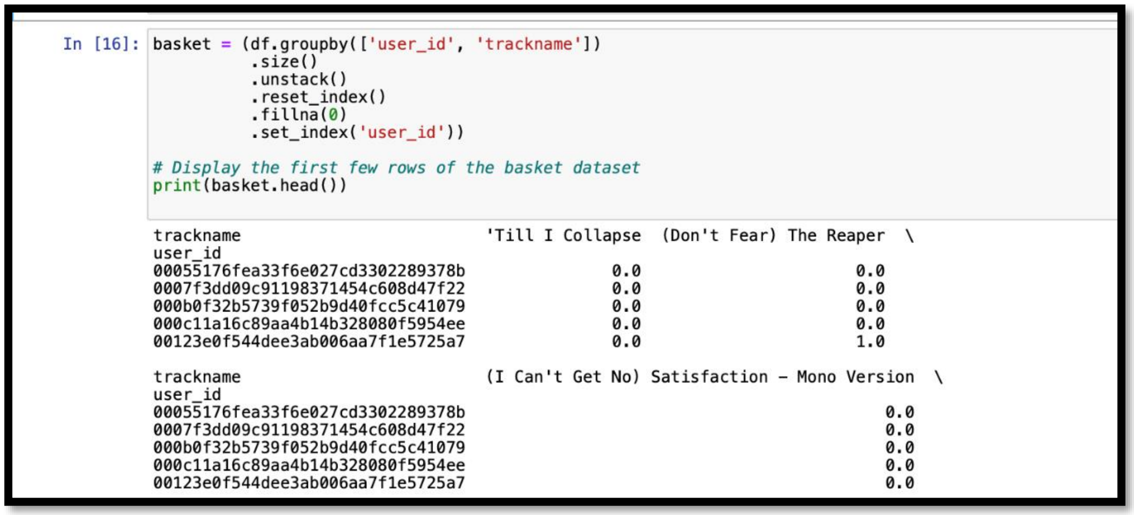
```
      support                     itemsets
0    0.015640                 (AWOLNATION)
1    0.029370                      (Alive)
2    0.026830                      (Angel)
3    0.019890                    (Animals)
4    0.010530    (Are You Gonna Be My Girl)
..        ...                          ...
564  0.019924                      (River)
565  0.010099                      (Robyn)
566  0.015576                    (Sublime)
567  0.010749                  (Timbaland)
568  0.010065   (Robyn, Dancing On My Own)

[569 rows x 2 columns]
```

Figure 9: Apirori Algorithm

"AWOLNATION," which appears in 1.56 percent of the transactions, has the highest level of support. The second-highest supported itemset, "Alive," is present in 2.93 percent of the transactions. "Angel" is the itemset that receives the third-highest support, appearing in 2.68 percent of the transactions. "Dancing On My Own, Robyn" is the itemset that receives the least support; it is used in 1.0065 percent of the transactions.

To identify frequent itemsets, we employ the Apriori algorithm. Due to its effectiveness and ability to work with huge datasets, this technique is frequently used for locating frequent itemsets. The Apriori algorithm is instructed to only take into account itemsets that are present in at least 1% of the transactions by the min support option. This indicates that the output only includes sets of things that are regarded as common.

We are aware of what clients like to buy. For instance, the 1.56 percent of transactions that include the itemset "AWOLNATION" implies that buyers who purchase music by AWOLNATION are also likely to purchase music by other artists in the same genre. Similarly, the fact that only 1.0065 percent of transactions contain the itemset "Dancing on My Own, Robyn" shows that buyers who purchase this song are less likely to purchase music from other artists.



```
In [16]: basket = (df.groupby(['user_id', 'trackname'])
                   .size()
                   .unstack()
                   .reset_index()
                   .fillna(0)
                   .set_index('user_id'))

         # Display the first few rows of the basket dataset
         print(basket.head())

trackname                         'Till I Collapse  (Don't Fear) The Reaper  \
user_id
00055176fea33f6e027cd3302289378b               0.0                      0.0
0007f3dd09c91198371454c608d47f22               0.0                      0.0
000b0f32b5739f052b9d40fcc5c41079               0.0                      0.0
000c11a16c89aa4b14b328080f5954ee               0.0                      0.0
00123e0f544dee3ab006aa7f1e5725a7               0.0                      1.0

trackname                         (I Can't Get No) Satisfaction - Mono Version  \
user_id
00055176fea33f6e027cd3302289378b                                           0.0
0007f3dd09c91198371454c608d47f22                                           0.0
000b0f32b5739f052b9d40fcc5c41079                                           0.0
000c11a16c89aa4b14b328080f5954ee                                           0.0
00123e0f544dee3ab006aa7f1e5725a7                                           0.0
```

Figure 10 basket implantation

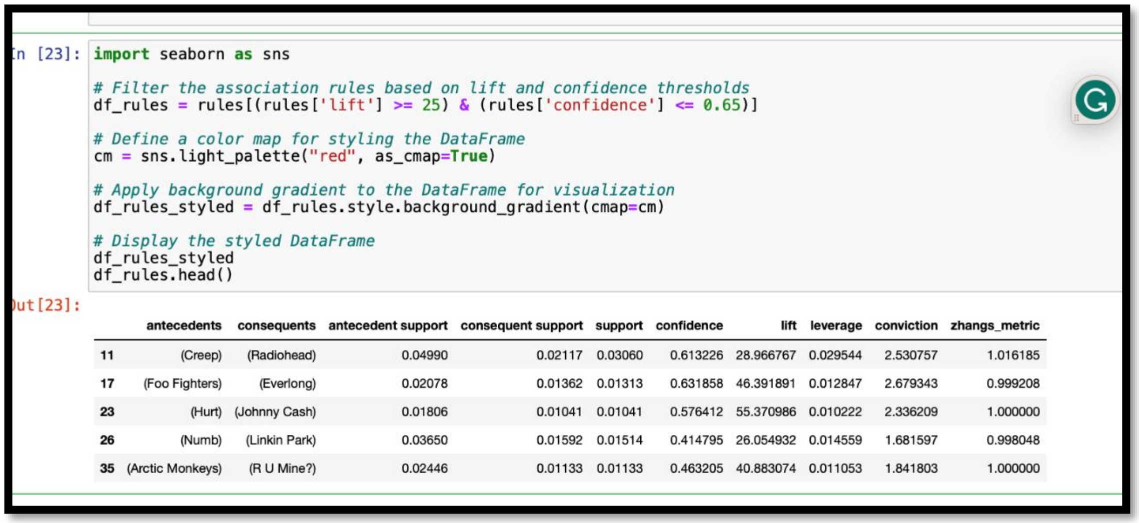
Output displays the first three entries in the list, as can be seen. The track name that a certain user buys is shown next to each item. The strings for the tracks serve as a representation of their names, while the numbers in the list indicate whether the track was purchased (1). (0).

As evidenced by the first entry, none of the tracks on the list were purchased by the person with the ID 00055176fea33f6e027cd3302289378b. According to the second item, the user with the ID 0007f3dd09c91198371454c608d47f22 did not buy any of the tracks on the list either. The third item indicates that the track Yummy was purchased by the person with the ID 000b0f32b5739f052b9d40fcc5c41079.

We are familiar with what people buy. One indication that a user may not be a frequent music shopper is the fact that the user with the ID 00055176fea33f6e027cd3302289378b did not buy any of the tracks on the list.

Like the previous user, it appears that the one with the ID 0007f3dd09c91198371454c608d47f22 may not be a frequent music buyer because they too did not buy any of the tunes on the list.

to recognise popular songs. For instance, the fact that the person with the ID 000b0f32b5739f052b9d40fcc5c41079 purchased the track Yummy shows that this track may be well-liked.



```
In [23]: import seaborn as sns

         # Filter the association rules based on lift and confidence thresholds
         df_rules = rules[(rules['lift'] >= 25) & (rules['confidence'] <= 0.65)]

         # Define a color map for styling the DataFrame
         cm = sns.light_palette("red", as_cmap=True)

         # Apply background gradient to the DataFrame for visualization
         df_rules_styled = df_rules.style.background_gradient(cmap=cm)

         # Display the styled DataFrame
         df_rules_styled
         df_rules.head()
Out[23]:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
11	(Creep)	(Radiohead)	0.04990	0.02117	0.03060	0.613226	28.966767	0.029544	2.530757	1.016185
17	(Foo Fighters)	(Everlong)	0.02078	0.01362	0.01313	0.631858	46.391891	0.012847	2.679343	0.999208
23	(Hurt)	(Johnny Cash)	0.01806	0.01041	0.01041	0.576412	55.370986	0.010222	2.336209	1.000000
26	(Numb)	(Linkin Park)	0.03650	0.01592	0.01514	0.414795	26.054932	0.014559	1.681597	0.998048
35	(Arctic Monkeys)	(R U Mine?)	0.02446	0.01133	0.01133	0.463205	40.883074	0.011053	1.841803	1.000000

Figure 11: Association Rule

The output shows the association rule as following:

1. A transaction is 28.97 times more likely to also contain the song "Radiohead" if it contains the song "Creep" by Radiohead.
2. A transaction is 46.39 times more likely to also contain the music "Foo Fighters" if it includes the Foo Fighters song "Everlong."
3. A transaction is 55.37 times more likely to also contain the music "Johnny Cash" if it contains the Johnny Cash song "Hurt".
4. A transaction's likelihood of additionally including the music "Linkin Park" increases by 26.05 times if it includes the Linkin Park song "Numb."
5. It is 40.88 times more likely for a transaction to also include the music "Arctic Monkeys" if it includes the Arctic Monkeys song "R U Mine?"

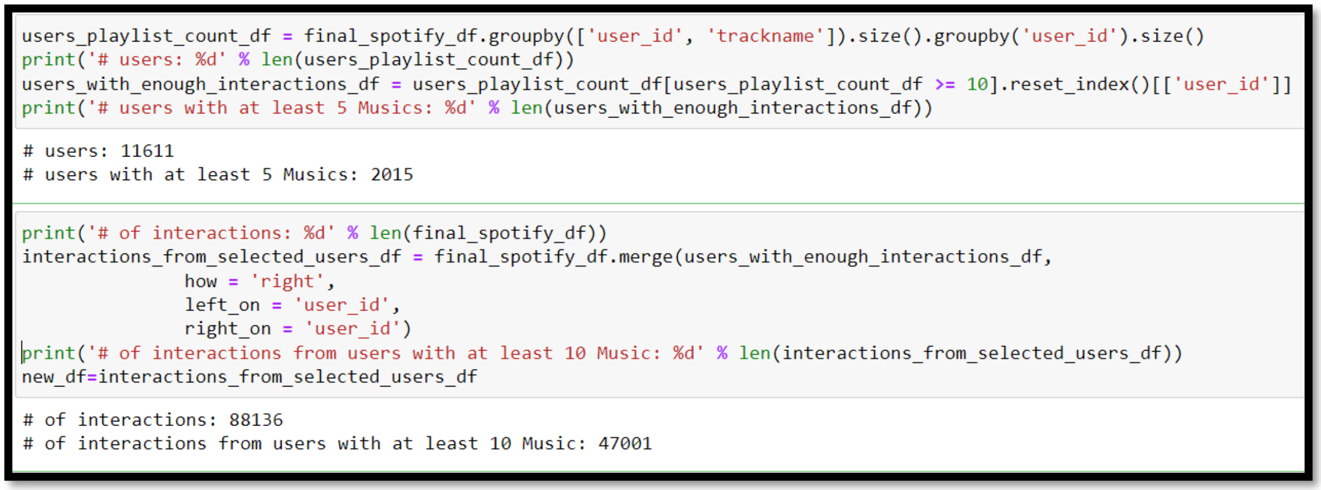
According to these rules, there are important relationships between specific songs and artists. For instance, it is quite likely that a customer will buy more songs by Radiohead if they buy the song "Creep" from the band. Businesses may use this information to target their marketing campaigns or provide customers with recommendations.

In addition to the top 5 regulations, the output contains numerous other rules that can be of interest to businesses. Bohemian Rhapsody and We Will Rock You by Queen are frequently bought together, according to one rule: "If a transaction contains the song "Bohemian Rhapsody" by Queen, then it is 11.46 times more likely to also contain the song "We Will Rock You" by Queen." Businesses may use this information to create song collections or to make song recommendations to customers based on their previous music purchases.

provides businesses with insightful data about consumer behavior that has an impact on their sales. Making personalized recommendations and concentrating marketing efforts based on this data may improve customers' overall experiences.

8.2 **Recommendation Systems**

Before we start building the recommendation system, we have to solve the user cold start problem. By filtering interactions from selected users who have met the minimum criteria of at least 10 music interactions, the code focuses on users who have demonstrated a certain level of engagement with the system. This approach helps mitigate the challenge posed by new users or users with limited interaction history. By working with a subset of data that comprises users who have shown sufficient engagement, the recommendation system can provide more accurate and meaningful recommendations tailored to their preferences and behaviors.



```
users_playlist_count_df = final_spotify_df.groupby(['user_id', 'trackname']).size().groupby('user_id').size()
print('# users: %d' % len(users_playlist_count_df))
users_with_enough_interactions_df = users_playlist_count_df[users_playlist_count_df >= 10].reset_index()[['user_id']]
print('# users with at least 5 Musics: %d' % len(users_with_enough_interactions_df))
```

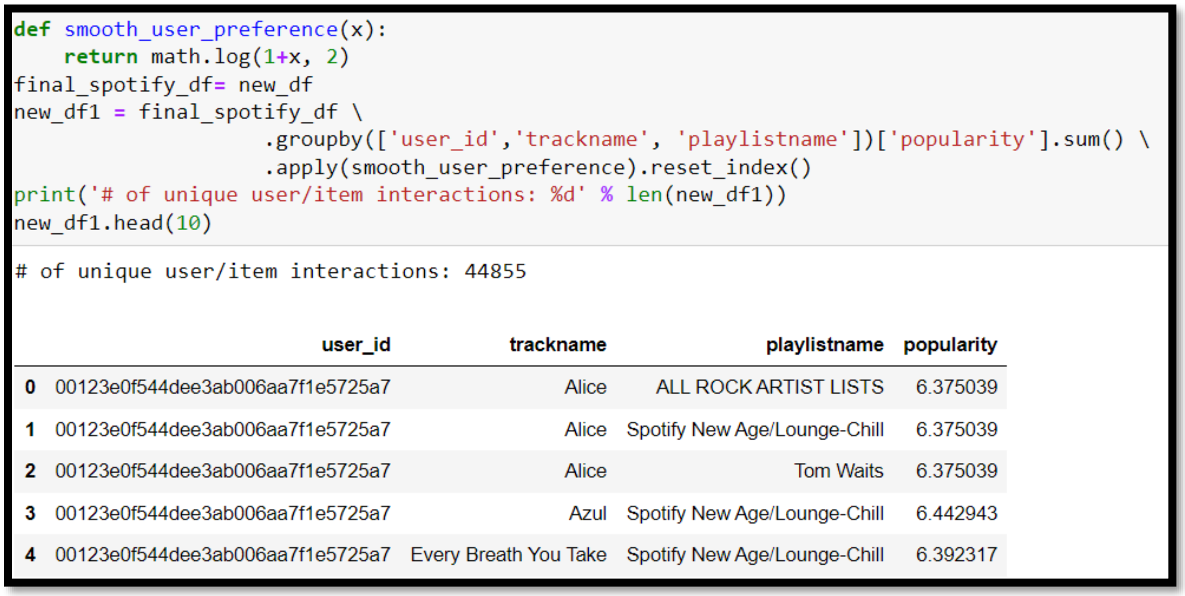
```
# users: 11611
# users with at least 5 Musics: 2015
```

```
print('# of interactions: %d' % len(final_spotify_df))
interactions_from_selected_users_df = final_spotify_df.merge(users_with_enough_interactions_df,
               how = 'right',
               left_on = 'user_id',
               right_on = 'user_id')
print('# of interactions from users with at least 10 Music: %d' % len(interactions_from_selected_users_df))
new_df=interactions_from_selected_users_df
```

```
# of interactions: 88136
# of interactions from users with at least 10 Music: 47001
```

Figure 12: User Cold start.

After that, we used the smooth_user_preference(x) function which is a mathematical transformation used to adjust the user preferences for music. It takes an input value x and applies a logarithmic transformation to it using base 2. The purpose of this transformation is to reduce values and reduce the impact of extreme preferences while retaining the relative order of preferences.
In the given code snippet, the function is applied to the popularity column of a DataFrame new_df after grouping the data by 'user_id', 'trackname', and 'playlistname'. This operation is used to compute the sum of ratings for each unique combination of user_id, trackname, and playlistname. This function aims to transform and aggregate user preferences for music, providing a more reliable and balanced measure of overall preference for different combinations.



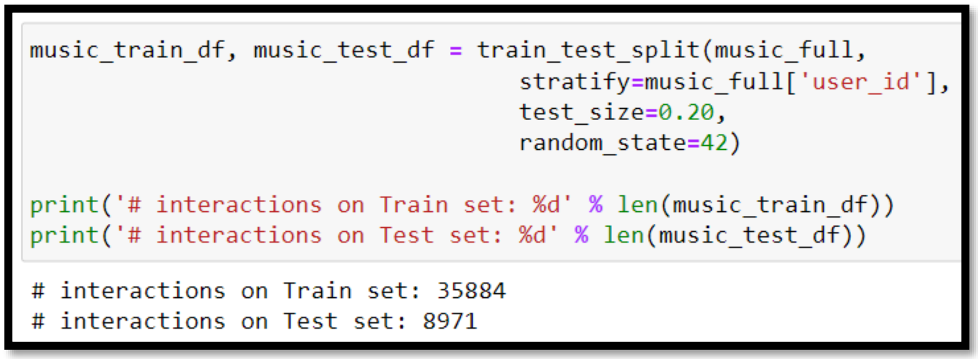
```
def smooth_user_preference(x):
    return math.log(1+x, 2)
final_spotify_df= new_df
new_df1 = final_spotify_df \
                     .groupby(['user_id','trackname', 'playlistname'])['popularity'].sum() \
                     .apply(smooth_user_preference).reset_index()
print('# of unique user/item interactions: %d' % len(new_df1))
new_df1.head(10)
```

```
# of unique user/item interactions: 44855
```

	user_id	trackname	playlistname	popularity
0	00123e0f544dee3ab006aa7f1e5725a7	Alice	ALL ROCK ARTIST LISTS	6.375039
1	00123e0f544dee3ab006aa7f1e5725a7	Alice	Spotify New Age/Lounge-Chill	6.375039
2	00123e0f544dee3ab006aa7f1e5725a7	Alice	Tom Waits	6.375039
3	00123e0f544dee3ab006aa7f1e5725a7	Azul	Spotify New Age/Lounge-Chill	6.442943
4	00123e0f544dee3ab006aa7f1e5725a7	Every Breath You Take	Spotify New Age/Lounge-Chill	6.392317

Figure 13: Smoothing the dataset.

After smoothing the data, we remain with 44855 unique interactions which contain the trackname, playlistname, user_id, and popularity.

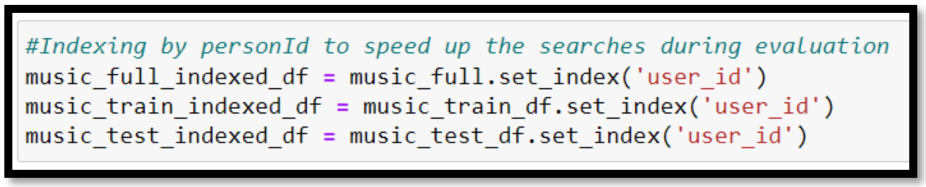


```
music_train_df, music_test_df = train_test_split(music_full,
                                   stratify=music_full['user_id'],
                                   test_size=0.20,
                                   random_state=42)

print('# interactions on Train set: %d' % len(music_train_df))
print('# interactions on Test set: %d' % len(music_test_df))
```

```
# interactions on Train set: 35884
# interactions on Test set: 8971
```

Figure 14: Splitting the dataset.

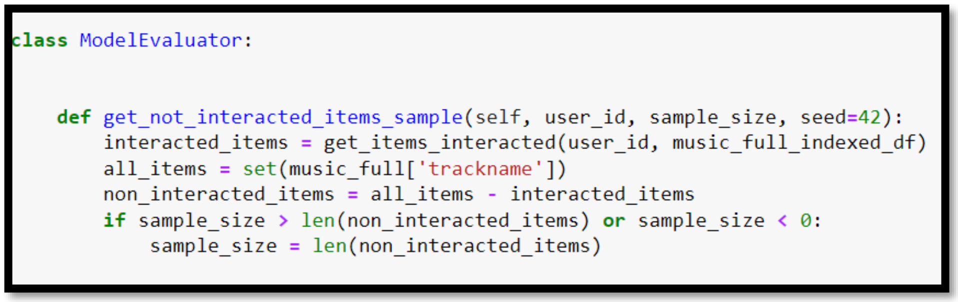
Next, we have to create a separate dataset for training and testing machine learning models or performing experiments. The training set is used to train the models, while the test set is used to evaluate the performance of the trained models on unseen data. By splitting the data, it helps to assess how well the models generalize to new, unseen movies and can provide insights into their predictive capabilities.



```
#Indexing by personId to speed up the searches during evaluation
music_full_indexed_df = music_full.set_index('user_id')
music_train_indexed_df = music_train_df.set_index('user_id')
music_test_indexed_df = music_test_df.set_index('user_id')
```

Figure 15: Indexing the dataset.

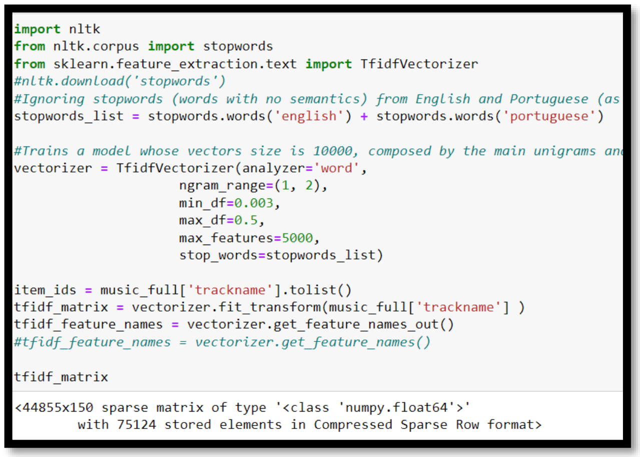
After that, we performed indexing on the DataFrames to speed up search operations during evaluation. It sets the 'Director' column as the index for the music_full This helps optimize the data access and retrieval process by taking advantage of indexing. By indexing the DataFrames, it becomes faster and more efficient to retrieve information related to specific directors, which can be useful for further analysis or recommendation systems that rely on director-based interactions or preferences.



```
class ModelEvaluator:


    def get_not_interacted_items_sample(self, user_id, sample_size, seed=42):
        interacted_items = get_items_interacted(user_id, music_full_indexed_df)
        all_items = set(music_full['trackname'])
        non_interacted_items = all_items - interacted_items
        if sample_size > len(non_interacted_items) or sample_size < 0:
            sample_size = len(non_interacted_items)
```

Figure 16: Evaluation class.

Next, we created a function to evaluate the performance of a recommendation model in terms of hit rate and recall for different user_id. It assesses how well the model can recommend items that a director has interacted with and ranks them among the top N recommendations.



```
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
#nltk.download('stopwords')
#Ignoring stopwords (words with no semantics) from English and Portuguese (as
stopwords_list = stopwords.words('english') + stopwords.words('portuguese')

#Trains a model whose vectors size is 10000, composed by the main unigrams an
vectorizer = TfidfVectorizer(analyzer='word',
                     ngram_range=(1, 2),
                     min_df=0.003,
                     max_df=0.5,
                     max_features=5000,
                     stop_words=stopwords_list)

item_ids = music_full['trackname'].tolist()
tfidf_matrix = vectorizer.fit_transform(music_full['trackname'] )
tfidf_feature_names = vectorizer.get_feature_names_out()
#tfidf_feature_names = vectorizer.get_feature_names()

tfidf_matrix
```

```
<44855x150 sparse matrix of type '<class 'numpy.float64'>'
	with 75124 stored elements in Compressed Sparse Row format>
```

Figure 17: TF-IDF transform matrix.

The next step is to transform the track names into numerical representations using TF-IDF. This allows machine learning models to work with text data by representing it as a matrix of numerical features. The resulting matrix can be used for tasks such as text classification, clustering, or recommendation systems. The output shows the shape of the tfidf_matrix, indicating it is a sparse matrix with 44855 rows (movies) and 150 columns (features), and it contains 75124 stored elements in a compressed sparse row format.

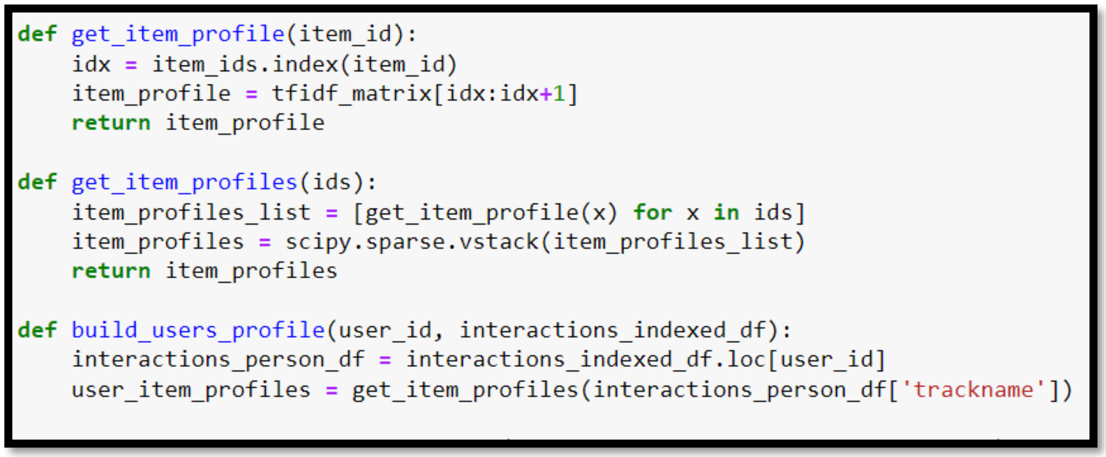

```python
def get_item_profile(item_id):
    idx = item_ids.index(item_id)
    item_profile = tfidf_matrix[idx:idx+1]
    return item_profile

def get_item_profiles(ids):
    item_profiles_list = [get_item_profile(x) for x in ids]
    item_profiles = scipy.sparse.vstack(item_profiles_list)
    return item_profiles

def build_users_profile(user_id, interactions_indexed_df):
    interactions_person_df = interactions_indexed_df.loc[user_id]
    user_item_profiles = get_item_profiles(interactions_person_df['trackname'])
```

Figure 18: Building user profile.

The next step is to create user profiles and item profiles for building a recommendation system. User profiles capture the preferences and interests of each users based on their interactions with track name. Item profiles represent the characteristics of each music based on its track name information. These profiles serve as input for recommendation algorithms that can match users with relevant items based on their preferences and similarities between items.
By building user profiles and item profiles, the recommendation system can generate personalized recommendations for userss by identifying tracks that align with their preferences and exhibit similar characteristics to the tracks they have previously interacted with. After performing this finctions we got 2015 users and items profile.

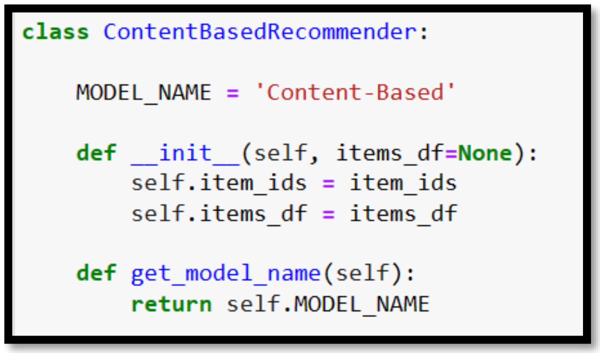
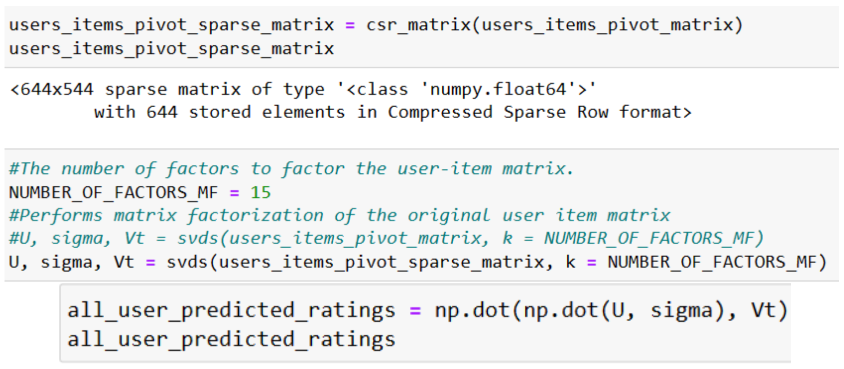

```python
class ContentBasedRecommender:

    MODEL_NAME = 'Content-Based'

    def __init__(self, items_df=None):
        self.item_ids = item_ids
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME
```

Figure 19: Content base recommender system.

The first used model for the recommender system in this project is a content-based recommender system. Content-based recommendation systems are a type of recommendation system that generate personalized recommendations based on the characteristics or "content" of the items being recommended. In content-based systems, the recommendations are made by comparing the features or attributes of items with the user's preferences or past interactions.
The purpose of the ContentBasedRecommender class is to provide personalized item recommendations based on the similarity between a user's profile and item profiles. It calculates the cosine similarity between the user's profile and all item profiles using TF-IDF representation. The most similar items are then recommended to the user, excluding any items they have already interacted with.



```
users_items_pivot_sparse_matrix = csr_matrix(users_items_pivot_matrix)
users_items_pivot_sparse_matrix
```

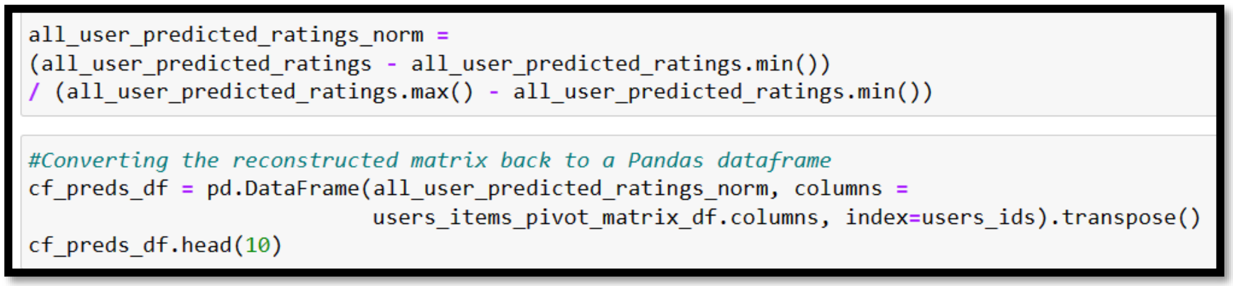
```
<644x544 sparse matrix of type '<class 'numpy.float64'>'
	with 644 stored elements in Compressed Sparse Row format>
```

```
#The number of factors to factor the user-item matrix.
NUMBER_OF_FACTORS_MF = 15
#Performs matrix factorization of the original user item matrix
#U, sigma, Vt = svds(users_items_pivot_matrix, k = NUMBER_OF_FACTORS_MF)
U, sigma, Vt = svds(users_items_pivot_sparse_matrix, k = NUMBER_OF_FACTORS_MF)
```

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt)
all_user_predicted_ratings
```

Figure 20: Transform the shape of the dataset.

Next, we perform matrix factorization to decompose the user-item matrix into lower-dimensional representations that capture the latent factors or features underlying the user-item interactions. This allows for dimensionality reduction and the identification of hidden patterns or similarities in the data. The resulting factor matrices can be used for making recommendations, as they capture the underlying user preferences and item characteristics.

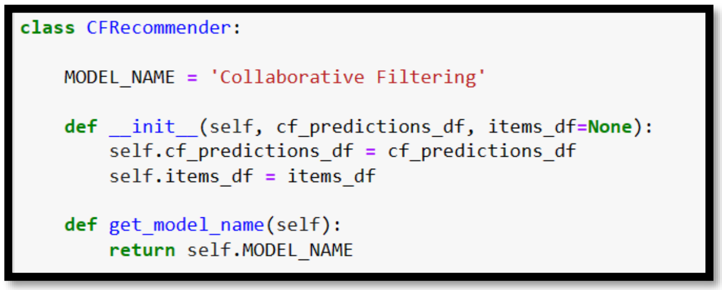


```
all_user_predicted_ratings_norm =
(all_user_predicted_ratings - all_user_predicted_ratings.min())
/ (all_user_predicted_ratings.max() - all_user_predicted_ratings.min())
```

```
#Converting the reconstructed matrix back to a Pandas dataframe
cf_preds_df = pd.DataFrame(all_user_predicted_ratings_norm, columns =
                           users_items_pivot_matrix_df.columns, index=users_ids).transpose()
cf_preds_df.head(10)
```

Figure 21: Normalize the dataset.

The last step of pre-processing the data for the collaborative recommender system is to normalize the ratings to ensure that the ratings are on a consistent scale and comparable across users and items. This normalization allows for meaningful comparison and ranking of items based on their predicted ratings.

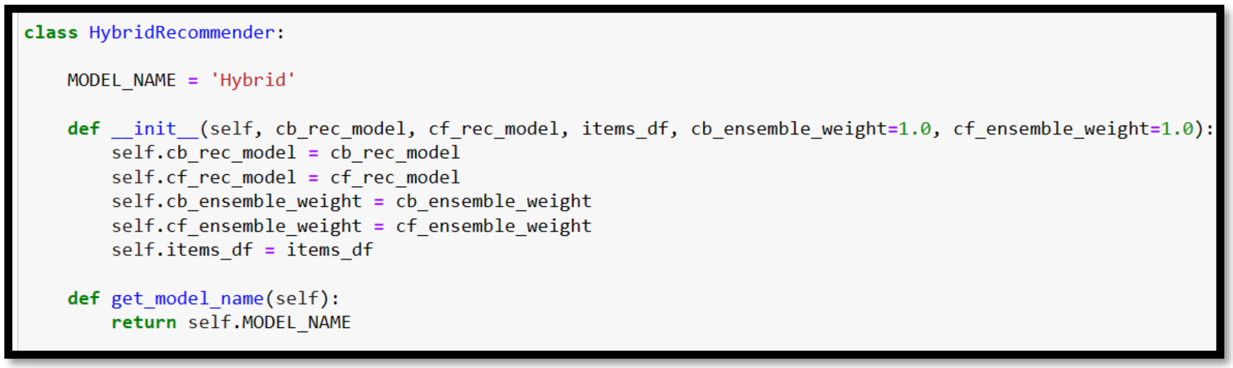


```
class CFRecommender:

    MODEL_NAME = 'Collaborative Filtering'

    def __init__(self, cf_predictions_df, items_df=None):
        self.cf_predictions_df = cf_predictions_df
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME
```

Figure 22: Building the collaborative recommender system.

Collaborative filtering is a recommendation technique that predicts a user's interests based on the preferences and behaviour of similar users. It assumes that users who have similar preferences in the past will have similar preferences in the future. Collaborative filtering does not require explicit item features or user profiles; instead, it relies on analyzing the patterns of user-item interactions.
We used the collaborative recommendation system via the CFRecommender class which

encapsulates the collaborative filtering model and provides a method called recommend_items to generate item recommendations. By instantiating an object of the CFRecommender class with the collaborative filtering predictions (cf_preds_df) and the item information (music_full), the model can generate personalized recommendations for specific users based on their predicted preferences derived from collaborative patterns of similar users.



```
class HybridRecommender:

    MODEL_NAME = 'Hybrid'

    def __init__(self, cb_rec_model, cf_rec_model, items_df, cb_ensemble_weight=1.0, cf_ensemble_weight=1.0):
        self.cb_rec_model = cb_rec_model
        self.cf_rec_model = cf_rec_model
        self.cb_ensemble_weight = cb_ensemble_weight
        self.cf_ensemble_weight = cf_ensemble_weight
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME
```

Figure 23: Building the Hybrid recommender system.

The third model, a hybrid recommendation system combines the predictive power of content-based and collaborative filtering models to provide more accurate and diverse recommendations. It leverages the strengths of each approach and allows for customization through weight assignment. By combining these techniques, the hybrid approach aims to overcome limitations and provide improved recommendations that better align with the user's preferences and interests.
The hybrid recommendation system combines multiple recommendation techniques to provide more accurate and diverse recommendations to users. In the above code, the HybridRecommender class combines content-based filtering (CB) and collaborative filtering (CF) techniques to generate recommendations.

8.2.1 Recommender system analysis

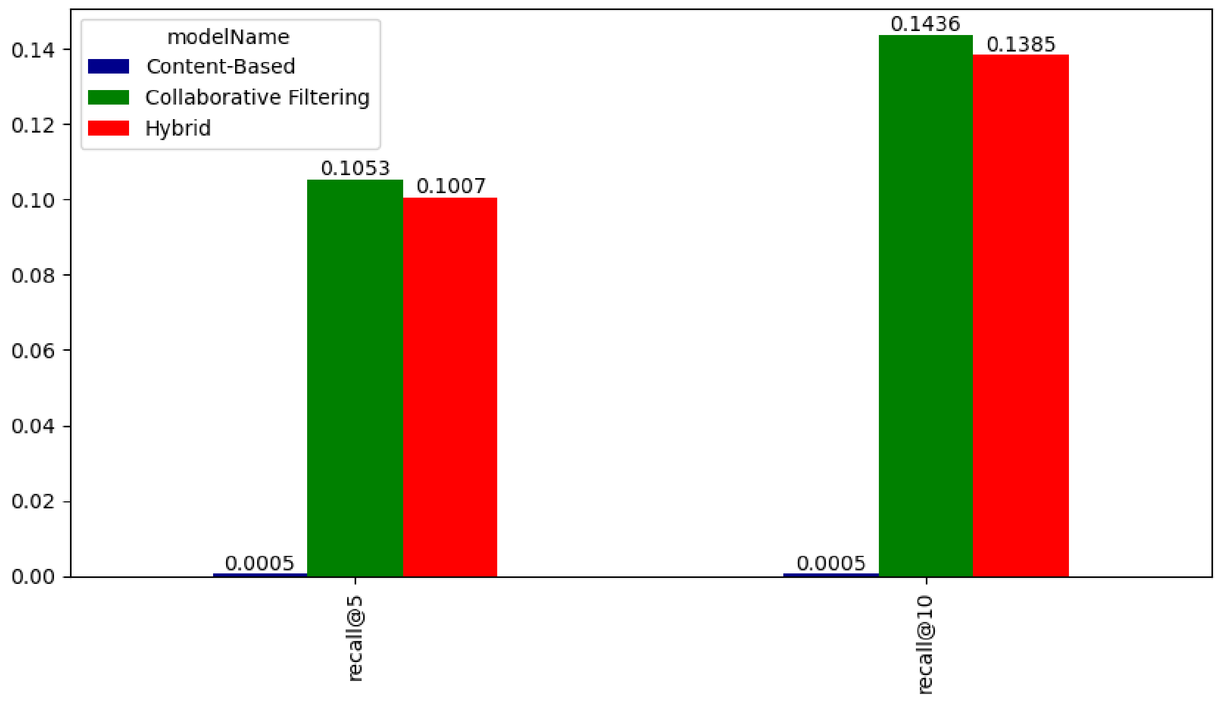
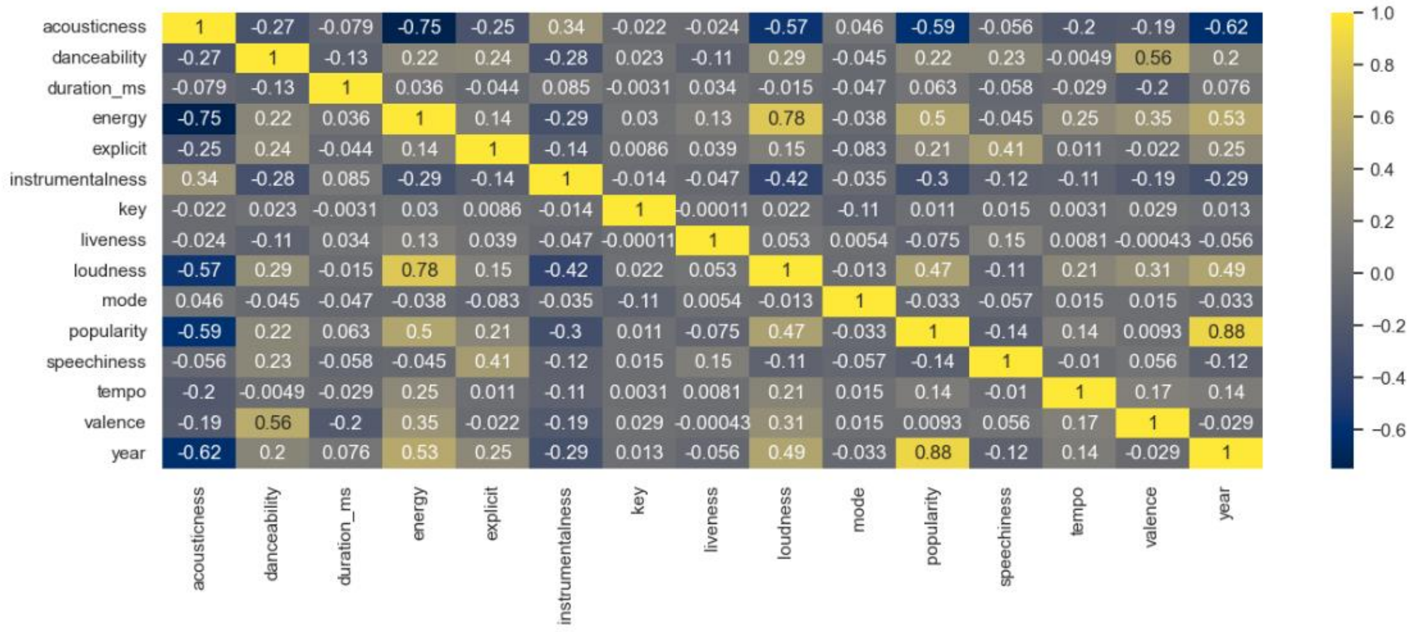


Figure 24: Recommender system result.

The recall@5 and recall@10 values for different models are presented and analyzed. The following models were evaluated: Content-Based, Collaborative Filtering, and Hybrid.
Among the evaluated models, Collaborative Filtering demonstrated superior performance, outperforming the other models in terms of recall at both cutoffs. It achieved a recall@5 of 0.105346 and a recall@10 of 0.143631. These results indicate that the Collaborative Filtering model was able to recommend relevant music items to users with higher accuracy compared to the other models. The Collaborative Filtering approach leverages user-item interactions and identifies similar users to provide recommendations, allowing for personalized and tailored suggestions based on user behavior and preferences.
The Hybrid model, which combines Content-Based and Collaborative Filtering techniques, also showcased promising performance. It achieved a recall@5 of 0.100650 and a recall@10 of 0.138454. The Hybrid model benefits from the strengths of both Content-Based and Collaborative Filtering approaches, utilizing content similarity and user behavior to provide more accurate recommendations. These results indicate that the combination of these two approaches can yield improved performance in music recommendation systems.
However, the Content-Based model exhibited relatively lower recall values compared to Collaborative Filtering and Hybrid models. It achieved a recall@5 of 0.000482 and a recall@10 of 0.000482. The lower accuracy of the Content-Based model can be attributed to the large number of different track names and their different characteristics. This approach may face limitations when dealing with complex user preferences that are not fully captured by item characteristics alone. Consequently, the Content-Based model may struggle to capture the diverse and nuanced preferences of users, resulting in lower accuracy in the recommendations provided.

8.3 Regression Analysis

Here, multiple regression analysis, KNNR, and Support Vector regression (SVR) are conducted. After extracting the required part of the dataset and performing the data preprocessing, the first step was the correlation analysis.



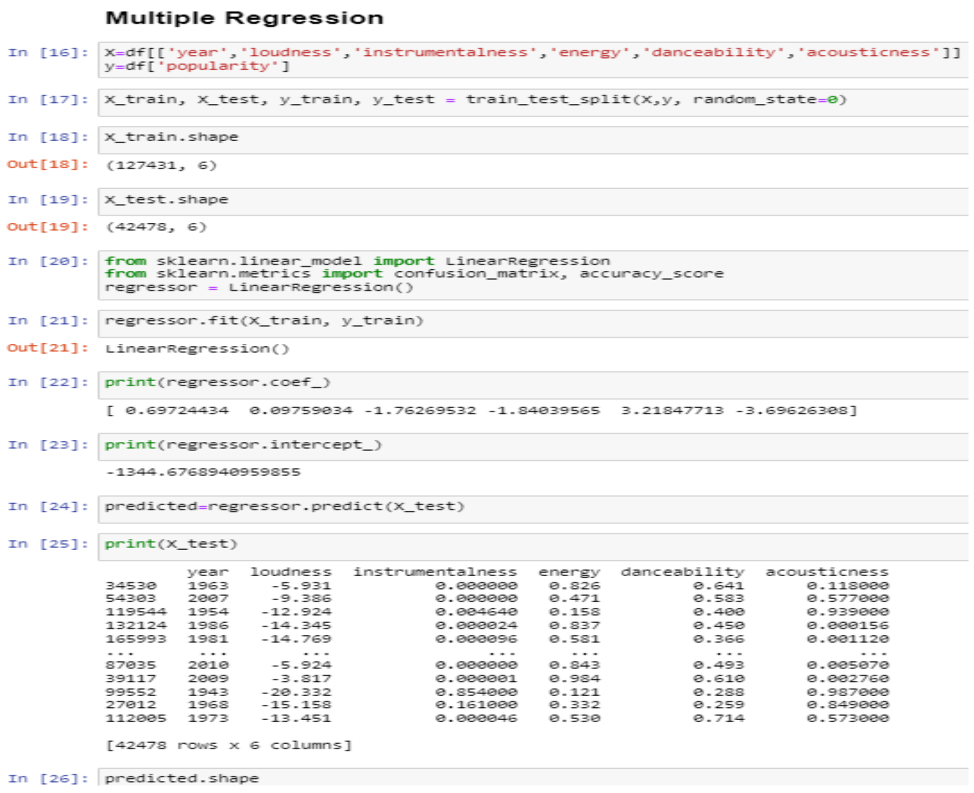
	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	valence	year
acousticness	1	-0.27	-0.079	-0.75	-0.25	0.34	-0.022	-0.024	-0.57	0.046	-0.59	-0.056	-0.2	-0.19	-0.62
danceability	-0.27	1	-0.13	0.22	0.24	-0.28	0.023	-0.11	0.29	-0.045	0.22	0.23	-0.0049	0.56	0.2
duration_ms	-0.079	-0.13	1	0.036	-0.044	0.085	-0.0031	0.034	-0.015	-0.047	0.063	-0.058	-0.029	-0.2	0.076
energy	-0.75	0.22	0.036	1	0.14	-0.29	0.03	0.13	0.78	-0.038	0.5	-0.045	0.25	0.35	0.53
explicit	-0.25	0.24	-0.044	0.14	1	-0.14	0.0086	0.039	0.15	-0.083	0.21	0.41	0.011	-0.022	0.25
instrumentalness	0.34	-0.28	0.085	-0.29	-0.14	1	-0.014	-0.047	-0.42	-0.035	-0.3	-0.12	-0.11	-0.19	-0.29
key	-0.022	0.023	-0.0031	0.03	0.0086	-0.014	1	-0.00011	0.022	-0.11	0.011	0.015	0.0031	0.029	0.013
liveness	-0.024	-0.11	0.034	0.13	0.039	-0.047	-0.00011	1	0.053	0.0054	-0.075	0.15	0.0081	-0.00043	-0.056
loudness	-0.57	0.29	-0.015	0.78	0.15	-0.42	0.022	0.053	1	-0.013	0.47	-0.11	0.21	0.31	0.49
mode	0.046	-0.045	-0.047	-0.038	-0.083	-0.035	-0.11	0.0054	-0.013	1	-0.033	-0.057	0.015	0.015	-0.033
popularity	-0.59	0.22	0.063	0.5	0.21	-0.3	0.011	-0.075	0.47	-0.033	1	-0.14	0.14	0.0093	0.88
speechiness	-0.056	0.23	-0.058	-0.045	0.41	-0.12	0.015	0.15	-0.11	-0.057	-0.14	1	-0.01	0.056	-0.12
tempo	-0.2	-0.0049	-0.029	0.25	0.011	-0.11	0.0031	0.0081	0.21	0.015	0.14	-0.01	1	0.17	0.14
valence	-0.19	0.56	-0.2	0.35	-0.022	-0.19	0.029	-0.00043	0.31	0.015	0.0093	0.056	0.17	1	-0.029
year	-0.62	0.2	0.076	0.53	0.25	-0.29	0.013	-0.056	0.49	-0.033	0.88	-0.12	0.14	-0.029	1

Figure 25: Correlation Analysis

Here the correlation is performed on fourteen of the nineteen columns within the dataset, and our focus is on the following columns with float values with the popularity of the song as our target: the year the song was released, the loudness, instrumentalness, energy, danceability, and acousticness of the song.
The correlation values are: for the year the song was released, it's 0.88; the loudness is 0.47; instrumentalness is -0.3; energy is 0.5; and the danceability and acousticness of the song are 0.22 and -0.59, respectively. This tells us that the year the song was released has a high positive correlation, while acousticness has a high negative correlation with the popularity of the song. This means that the popularity of the song does depend on the year it was released as well as the fact that when the song has more acousticness, it is less likely to be popular. Columns such as energy and loudness have almost the same value and the same correlation with popularity, which can be identified as a normal correlation.

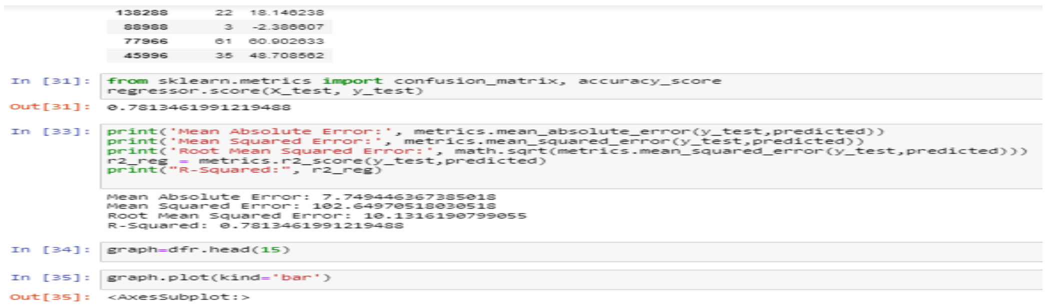
8.3.1 KNNR vs Multiple Linear Regression vs SVR

After extracting the required part of the dataset and performing the data preprocessing. I used these features as input features and popularity as a target feature.
In figure 25 shows how, I trained the multiple regression model.



Multiple Regression

```
In [16]: X=df[['year','loudness','instrumentalness','energy','danceability','acousticness']]
         y=df['popularity']

In [17]: X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=0)

In [18]: X_train.shape
Out[18]: (127431, 6)

In [19]: X_test.shape
Out[19]: (42478, 6)

In [20]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import confusion_matrix, accuracy_score
         regressor = LinearRegression()

In [21]: regressor.fit(X_train, y_train)
Out[21]: LinearRegression()

In [22]: print(regressor.coef_)
         [ 0.69724434  0.09759034 -1.76269532 -1.84039565  3.21847713 -3.69626308]

In [23]: print(regressor.intercept_)
         -1344.6768940959855

In [24]: predicted=regressor.predict(X_test)

In [25]: print(X_test)
                 year  loudness  instrumentalness  energy  danceability  acousticness
         34530   1963    -5.931          0.000000   0.826         0.641      0.118000
         54303   2007    -9.386          0.000000   0.471         0.583      0.577000
         119544  1954   -12.924          0.004640   0.158         0.400      0.939000
         132124  1986   -14.345          0.000024   0.837         0.450      0.000156
         165993  1981   -14.769          0.000096   0.581         0.366      0.001120
         ...      ...       ...               ...     ...           ...           ...
         87035   2010    -5.924          0.000000   0.843         0.493      0.005070
         39117   2009    -3.817          0.000001   0.984         0.610      0.002760
         99552   1943   -20.332          0.854000   0.121         0.288      0.987000
         27012   1968   -15.158          0.161000   0.332         0.259      0.849000
         112005  1973   -13.451          0.000046   0.530         0.714      0.573000

         [42478 rows x 6 columns]

In [26]: predicted.shape
```



```
          138288     22   18.146238
           88988      3   -2.386607
           77966     61   60.902633
           45996     35   48.708562

In [31]: from sklearn.metrics import confusion_matrix, accuracy_score
         regressor.score(X_test, y_test)
Out[31]: 0.7813461991219488

In [33]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,predicted))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test,predicted))
         print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test,predicted)))
         r2_reg = metrics.r2_score(y_test,predicted)
         print("R-Squared:", r2_reg)

         Mean Absolute Error: 7.749446367385018
         Mean Squared Error: 102.64970518030518
         Root Mean Squared Error: 10.1316190799055
         R-Squared: 0.7813461991219488

In [34]: graph=dfr.head(15)

In [35]: graph.plot(kind='bar')
Out[35]: <AxesSubplot:>
```

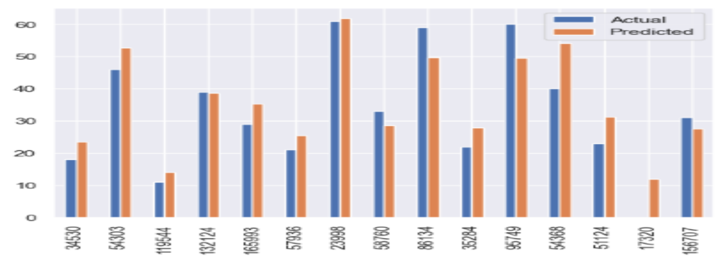
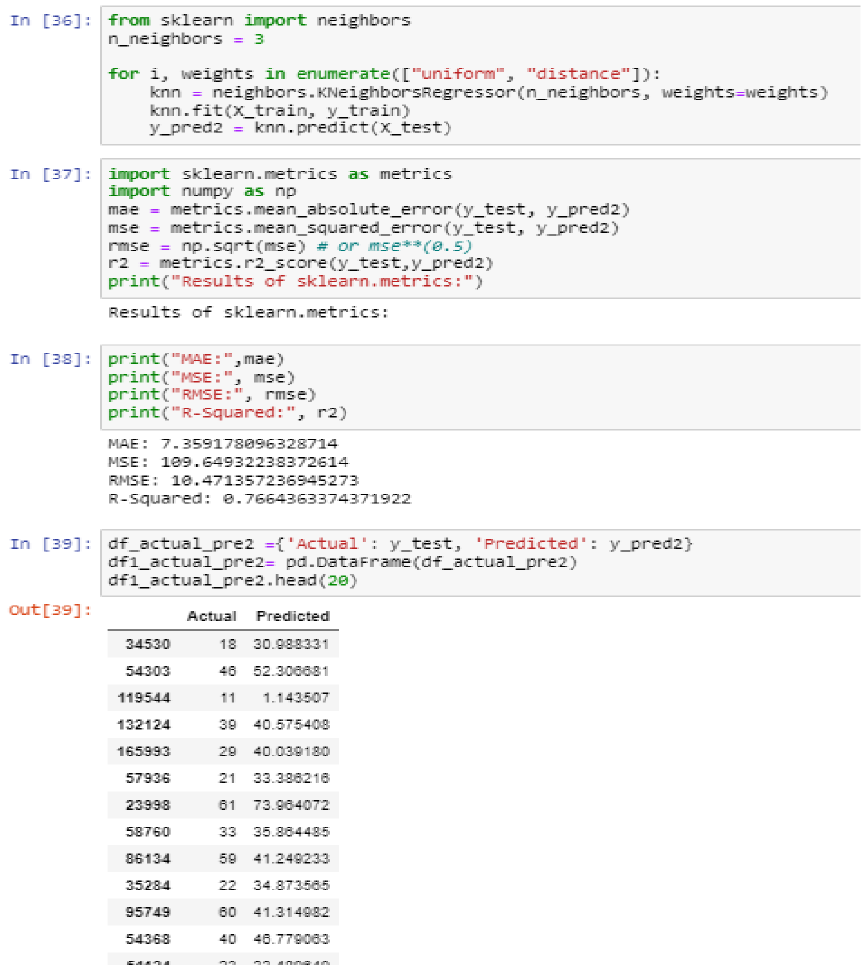


Figure 26: multiple regression model code

After multiple regression model I also built a KNNR and SVR models.



```
In [36]: from sklearn import neighbors
         n_neighbors = 3

         for i, weights in enumerate(["uniform", "distance"]):
             knn = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
             knn.fit(X_train, y_train)
             y_pred2 = knn.predict(X_test)
```

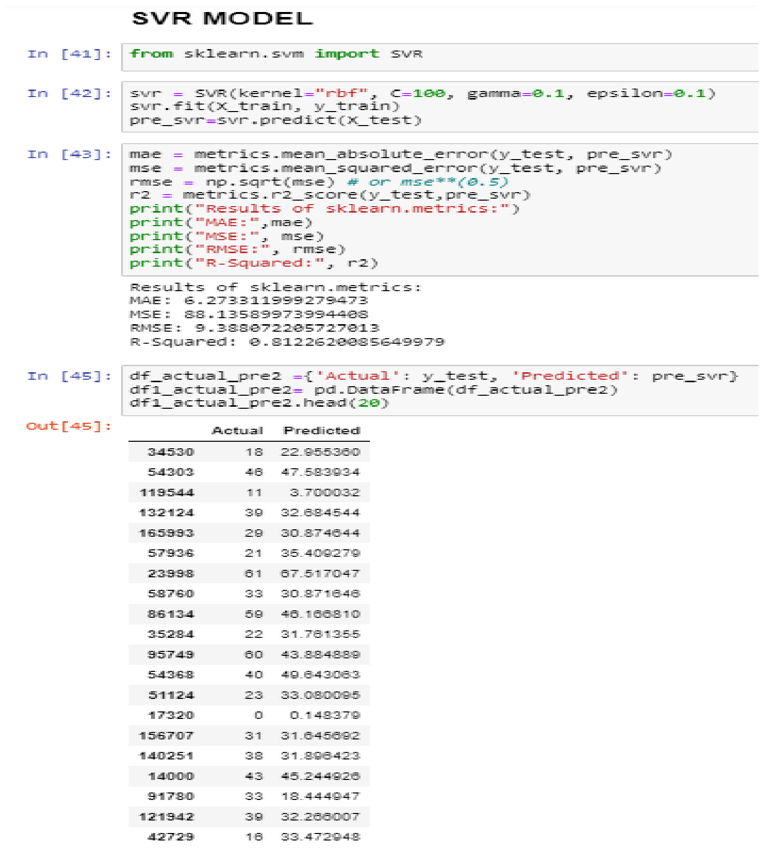
```
In [37]: import sklearn.metrics as metrics
         import numpy as np
         mae = metrics.mean_absolute_error(y_test, y_pred2)
         mse = metrics.mean_squared_error(y_test, y_pred2)
         rmse = np.sqrt(mse) # or mse**(0.5)
         r2 = metrics.r2_score(y_test,y_pred2)
         print("Results of sklearn.metrics:")

         Results of sklearn.metrics:
```

```
In [38]: print("MAE:",mae)
         print("MSE:", mse)
         print("RMSE:", rmse)
         print("R-Squared:", r2)

         MAE: 7.359178096328714
         MSE: 109.64932238372614
         RMSE: 10.471357236945273
         R-Squared: 0.7664363374371922
```

```
In [39]: df_actual_pre2 ={'Actual': y_test, 'Predicted': y_pred2}
         df1_actual_pre2= pd.DataFrame(df_actual_pre2)
         df1_actual_pre2.head(20)
```

Out[39]:

	Actual	Predicted
34530	18	30.988331
54303	46	52.306681
119544	11	1.143507
132124	39	40.575408
165993	29	40.039180
57936	21	33.386216
23998	61	73.964072
58760	33	35.864485
86134	59	41.249233
35284	22	34.873565
95749	60	41.314982
54368	40	46.779063
[illegible]	[illegible]	[illegible]



SVR MODEL

```
In [41]: from sklearn.svm import SVR

In [42]: svr = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
         svr.fit(X_train, y_train)
         pre_svr=svr.predict(X_test)

In [43]: mae = metrics.mean_absolute_error(y_test, pre_svr)
         mse = metrics.mean_squared_error(y_test, pre_svr)
         rmse = np.sqrt(mse) # or mse**(0.5)
         r2 = metrics.r2_score(y_test,pre_svr)
         print("Results of sklearn.metrics:")
         print("MAE:",mae)
         print("MSE:", mse)
         print("RMSE:", rmse)
         print("R-Squared:", r2)

         Results of sklearn.metrics:
         MAE: 6.273311999279473
         MSE: 88.13589973994408
         RMSE: 9.388072205727013
         R-Squared: 0.8122620085649979

In [45]: df_actual_pre2 ={'Actual': y_test, 'Predicted': pre_svr}
         df1_actual_pre2= pd.DataFrame(df_actual_pre2)
         df1_actual_pre2.head(20)

Out[45]:
                Actual  Predicted
         34530      18  22.955360
         54303      48  47.583934
        119544      11   3.700032
        132124      39  32.684544
        165993      29  30.874644
         57936      21  35.409279
         23998      61  67.517047
         58760      33  30.871646
         86134      59  48.166810
         35284      22  31.761355
         95749      60  43.884889
         54368      40  49.643063
         51124      23  33.080095
         17320       0   0.148379
        156707      31  31.645692
        140251      38  31.896423
         14000      43  45.244926
         91780      33  18.444947
        121942      39  32.266007
         42729      16  33.472948
```

Figure 27: Model training for SVR and KNNR

Following model training, three models are compared.

Table 3: Regression models comparison

Model	MAE	RMSE	MSE	R-squared
Multiple linear regression	7.749	10.13	102.6	0.78
KNNR regressor	7.35	10.47	109.69	0.766
SVR	6.27	9.38	88.13	0.812

Table 3 & figure 28 and 29 presents a comparative analysis of three distinct regression models: Multiple Linear Regression, K-Nearest Neighbors Regressor (KNNR), and Support Vector Regression (SVR). The evaluation metrics used in this comparison are Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Mean Square Error (MSE), and R-squared.

The Multiple Linear Regression model exhibits an MAE of 7.749, an RMSE of 10.13, an MSE of 102.6, and an R-squared value of 0.78. This suggests a moderate level of prediction accuracy, with a relatively high degree of variability in the model's predictions.

The KNNR model, on the other hand, demonstrates an MAE of 7.35, an RMSE of 10.47, an MSE of 109.69, and an R-squared value of 0.766. While the KNNR model has a slightly lower MAE than the Multiple Linear Regression model, it also has a higher RMSE and MSE, indicating a greater spread of residuals and a slightly lower proportion of the variance in the dependent variable that is predictable from the independent variables.

Lastly, the SVR model shows an MAE of 6.27, an RMSE of 9.38, an MSE of 88.13, and an R-squared value of 0.812. These values indicate that the SVR model has the lowest error rates and the highest R-squared value among the three models, suggesting that it provides the most accurate predictions and explains a larger proportion of the variance in the dependent variable.

In summary, while all three models demonstrate a certain degree of predictive accuracy, the SVR model appears to perform the best based on the evaluation metrics used in this comparison.

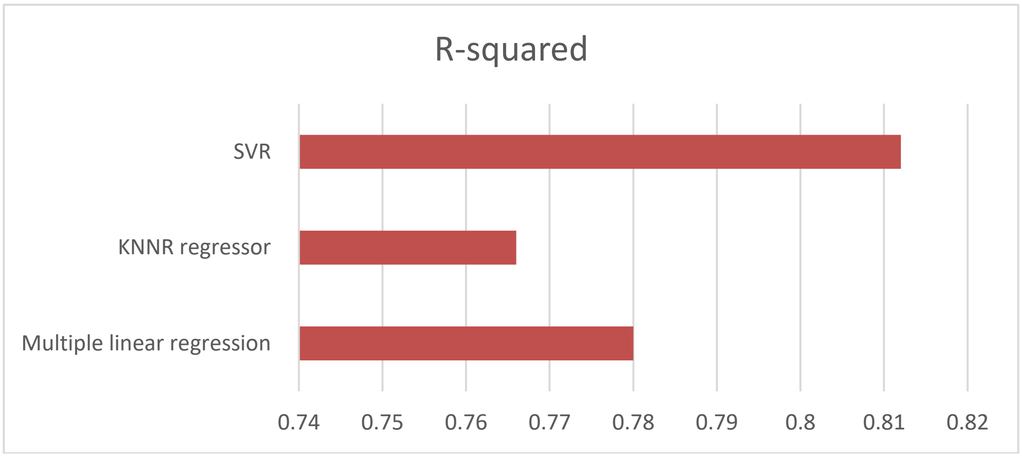


Figure 28: R-square for the three models

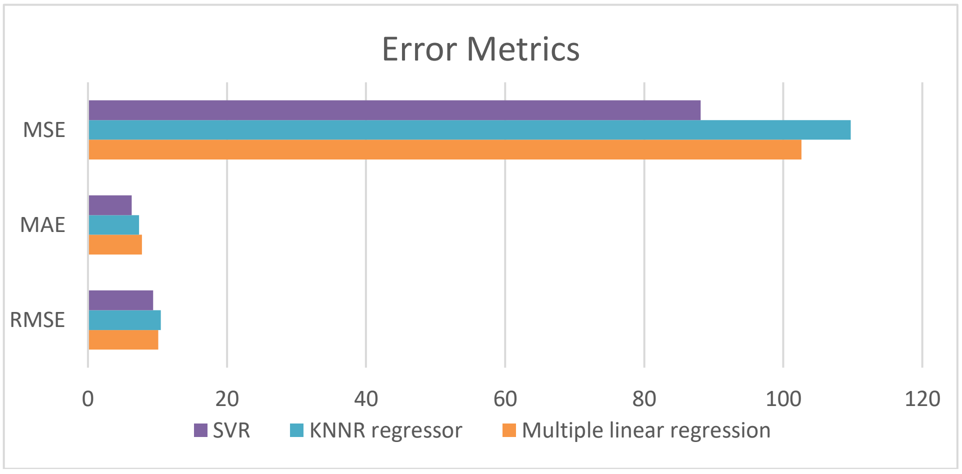


Figure 29: MSE, RMSE and MAE errors

9.0 Conclusion

In this study, we focused on the analysis of a music recommendation system and conducted experiments using various machine learning methods such as market basket analysis, recommender systems, and regression analysis. The scope of the project was limited to these three methods, and we utilized a comprehensive dataset containing over one million records of user-song preferences.

In terms of market basket analysis, we applied the Apriori algorithm and FP Growth to identify frequent itemsets and association rules within the dataset. This analysis provided valuable insights into co-occurrence patterns of music items, allowing businesses to understand customers' preferences and make informed decisions for product placement and recommendations.

For recommender systems, we implemented three models: Content-Based, Collaborative Filtering, and Hybrid. The evaluation of these models based on recall metrics revealed that Collaborative Filtering outperformed the other models, achieving higher recall@5 and recall@10 scores. Collaborative Filtering leverages user-item interactions to identify similar users and provide personalized recommendations. The Hybrid model, which combined Content-Based and Collaborative Filtering approaches, also showed promising performance, demonstrating the potential of leveraging multiple techniques for improved accuracy in music recommendation systems. However, the Content-Based model exhibited lower accuracy, mainly due to its reliance solely on item characteristics, which may not fully capture diverse user preferences.

Regression analysis was conducted to explore the relationship between song attributes and popularity. Multiple Linear Regression, KNNR, and SVR models were trained and compared. The SVR model emerged as the most accurate, with the lowest error rates and the highest R-squared value, indicating its ability to provide reliable predictions and explain a significant portion of the variance in song popularity.

Overall, this study shed light on the performance and limitations of different machine learning techniques within the context of a music recommendation system. The findings suggest that Collaborative Filtering and Hybrid approaches hold promise for accurate and personalized recommendations, while market basket analysis can provide valuable insights into customer behaviour. The regression analysis highlighted the impact of song attributes on popularity, allowing businesses to better understand the factors influencing music trends.

Moving forward, further research and refinement of recommendation algorithms, incorporating additional features and considering hybrid approaches, can lead to enhanced accuracy and personalization in music recommendation systems. Additionally, exploring other machine learning techniques and incorporating user feedback and contextual information could contribute to the continuous improvement and optimization of the recommendation system in the dynamic music industry.

References

[1] Nast, C. (2019) Why Big Data Has Been (Mostly) Good for Music, Wired. Available at: https://www.wired.com/story/big-data-music/

[2] Market Basket Analysis Available at: www.techtarget.com/searchcustomerexperience/definition/market-basket-analysis%3famp=1

[3] Analysing customer preferences for Astro using Market Basket Analysis / Nur Fatin Atirah Dahlan - UiTM Institutional Repository (2022). Available at: https://ir.uitm.edu.my/id/eprint/41584/

[4] Understanding Recommender Systems Available at: www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/

[5] Chen, H. and Chen, A. (2005) "A Music Recommendation System Based on Music and User Grouping", Journal of Intelligent Information Systems, 24(2-3), pp. 113-132. doi: 10.1007/s10844-005-0319-3.

[6] Salganik, Matthew J., Peter Sheridan Dodds, and Duncan J. Watts. ”Experimental study of inequality and unpredictability in an artificial cultural market.” science 311.5762 (2006): 854-856

[7] Jeff Saltz and Nick Hotz, “EVALUATING CRISP-DM FOR DATASCIENCE,” Aug. 2021. https://www.datascience-pm.com/wp-content/uploads/2021/08/CRISP-DM-for-Data-Science.pdf (accessed Jun. 07, 2023).

[8] Ekta Negi and Geetika Singla, “Spotify-Data 1921-2020,” 2020. https://www.kaggle.com/datasets/ektanegi/spotifydata-19212020 (accessed Jun. 07, 2023).

[9] Larxel and M. Pichl, “Spotify Playlists,” 2015. https://www.kaggle.com/datasets/andrewmvd/spotify-playlists (accessed Jun. 07, 2023).

[10] https://www.turing.com/kb/market-basket-analysis

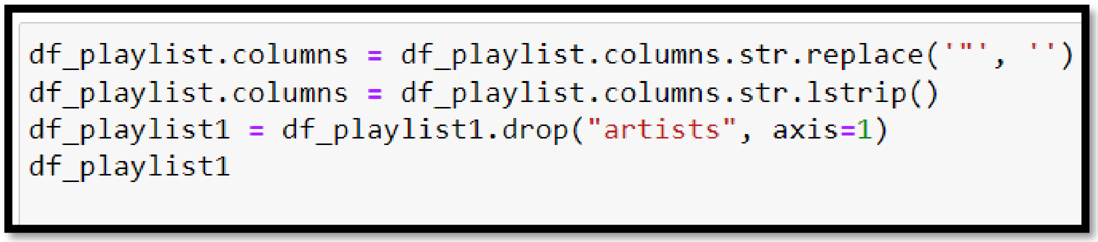
[11] https://www.turing.com/kb/market-basket-analysis

[12] https://www.softwaretestinghelp.com/fp-growth-algorithm-data-mining/

Appendix

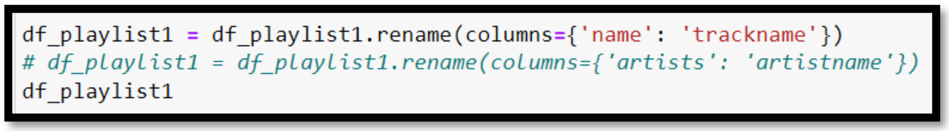
Source Code Screen Shots

Data pre-processing



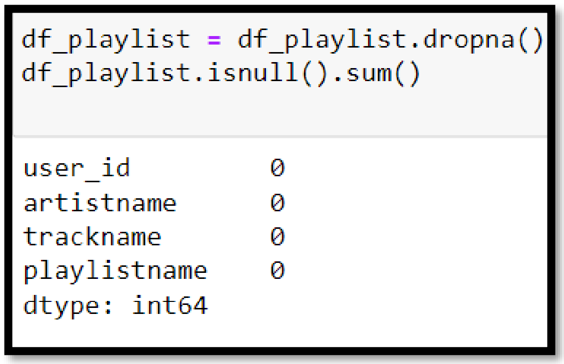
```
df_playlist.columns = df_playlist.columns.str.replace('"', '')
df_playlist.columns = df_playlist.columns.str.lstrip()
df_playlist1 = df_playlist1.drop("artists", axis=1)
df_playlist1
```

Clean the dataset



```
df_playlist1 = df_playlist1.rename(columns={'name': 'trackname'})
# df_playlist1 = df_playlist1.rename(columns={'artists': 'artistname'})
df_playlist1
```

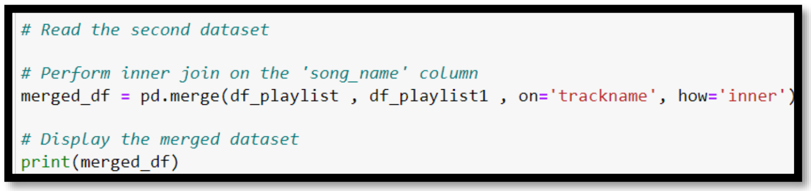
Rename the dataset to make it clearer and easier to merge the 2 datasets together.



```
df_playlist = df_playlist.dropna()
df_playlist.isnull().sum()
```

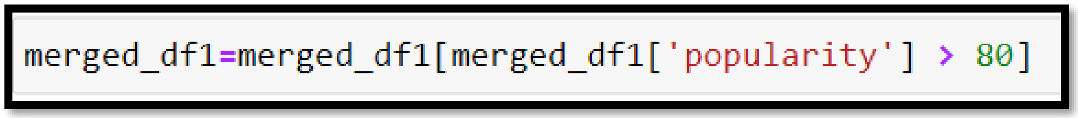
```
user_id         0
artistname      0
trackname       0
playlistname    0
dtype: int64
```

Removing the null values from the dataset



```
# Read the second dataset

# Perform inner join on the 'song_name' column
merged_df = pd.merge(df_playlist , df_playlist1 , on='trackname', how='inner')

# Display the merged dataset
print(merged_df)
```

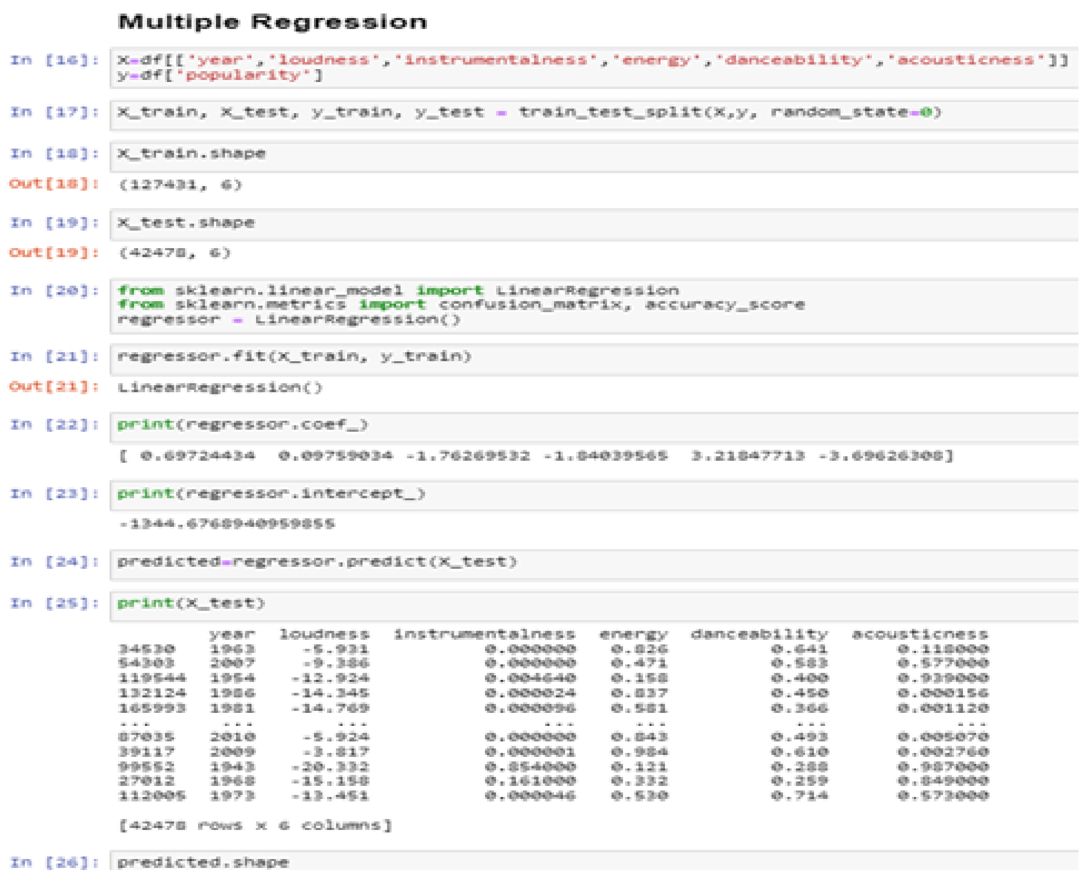
Merging the 2 datasets



```
merged_df1=merged_df1[merged_df1['popularity'] > 80]
```

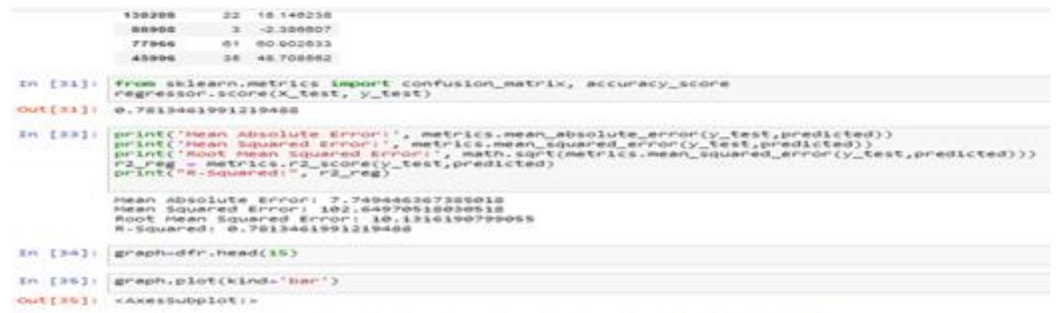
Filtering by the popularity and taking the top 20% songs.

Multiple Linear Regression, KNNR and SVR

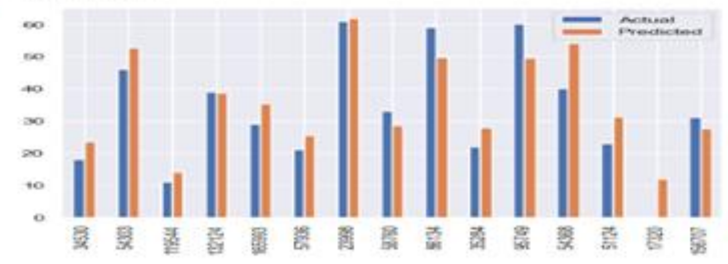


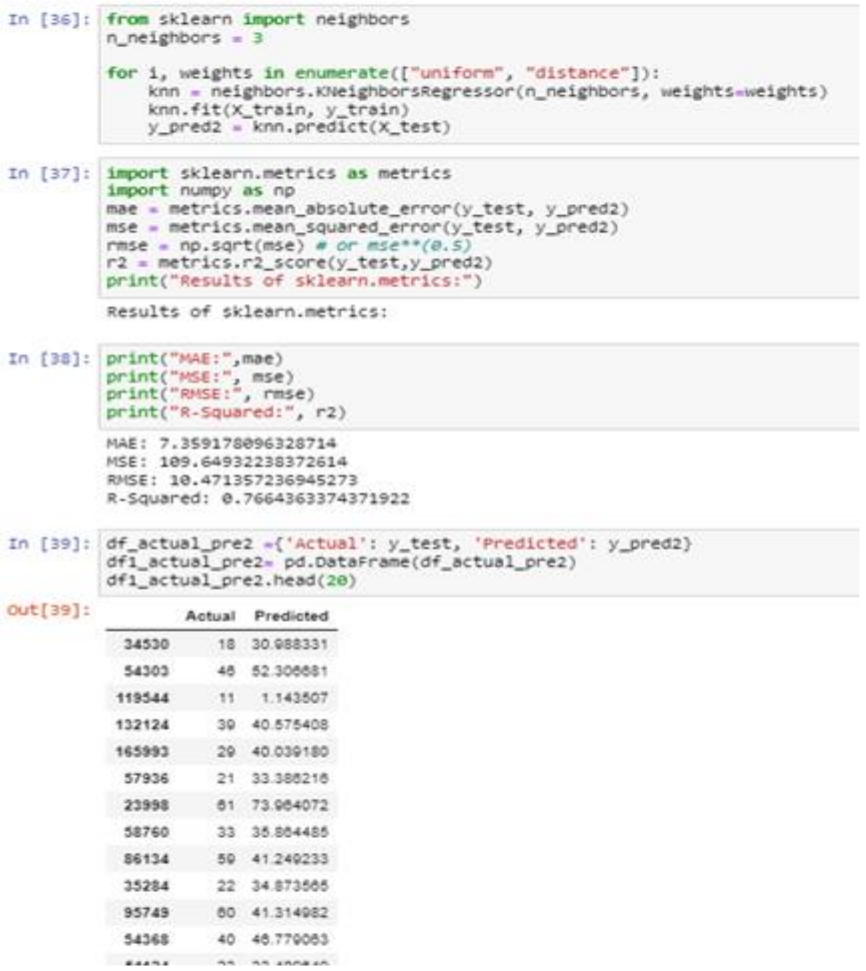
Multiple Regression

```
In [16]: X=df[['year','loudness','instrumentalness','energy','danceability','acousticness']]
         y=df['popularity']

In [17]: X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=0)

In [18]: X_train.shape

Out[18]: (127431, 6)

In [19]: X_test.shape

Out[19]: (42478, 6)

In [20]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import confusion_matrix, accuracy_score
         regressor = LinearRegression()

In [21]: regressor.fit(X_train, y_train)

Out[21]: LinearRegression()

In [22]: print(regressor.coef_)

         [ 0.69724434  0.09759034 -1.76269532 -1.84039565  3.21847713 -3.69626308]

In [23]: print(regressor.intercept_)

         -1344.6768940959855

In [24]: predicted=regressor.predict(X_test)

In [25]: print(X_test)

                 year  loudness  instrumentalness  energy  danceability  acousticness
         34530   1963    -5.931          0.000000   0.826         0.641      0.118000
         54303   2007    -9.386          0.000000   0.471         0.583      0.577000
         119544  1954   -12.924          0.004640   0.158         0.400      0.939000
         132124  1986   -14.345          0.000024   0.837         0.450      0.000156
         165993  1981   -14.769          0.000096   0.581         0.366      0.001120
         ...      ...       ...               ...     ...           ...           ...
         87035   2010    -5.924          0.000000   0.843         0.493      0.005070
         39117   2009    -3.817          0.000001   0.984         0.610      0.002760
         99552   1943   -20.332          0.854000   0.121         0.288      0.987000
         27012   1968   -15.158          0.161000   0.332         0.259      0.849000
         112005  1973   -13.451          0.000046   0.530         0.714      0.573000

         [42478 rows x 6 columns]

In [26]: predicted.shape
```



```
         130288     22   18.148258
         88988       3   -2.386807
         77966      61   60.902633
         45996      38   48.708862

In [31]: from sklearn.metrics import confusion_matrix, accuracy_score
         regressor.score(X_test, y_test)
Out[31]: 0.7813461991219488

In [33]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,predicted))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test,predicted))
         print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test,predicted)))
         r2_reg = metrics.r2_score(y_test,predicted)
         print("R-Squared:", r2_reg)

         Mean Absolute Error: 7.7494463673385018
         Mean Squared Error: 102.6497051803051 8
         Root Mean Squared Error: 10.131619079905 5
         R-Squared: 0.7813461991219488

In [34]: graph=dfr.head(15)

In [35]: graph.plot(kind='bar')
Out[35]: <AxesSubplot:>
```





```
In [36]: from sklearn import neighbors
         n_neighbors = 3

         for i, weights in enumerate(["uniform", "distance"]):
             knn = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
             knn.fit(X_train, y_train)
             y_pred2 = knn.predict(X_test)
```

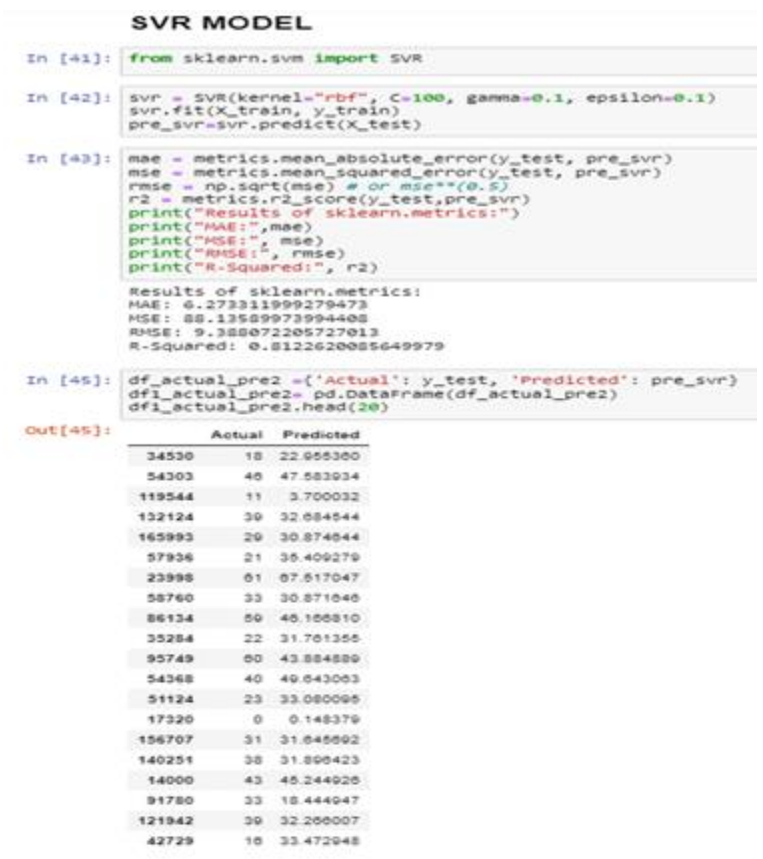
```
In [37]: import sklearn.metrics as metrics
         import numpy as np
         mae = metrics.mean_absolute_error(y_test, y_pred2)
         mse = metrics.mean_squared_error(y_test, y_pred2)
         rmse = np.sqrt(mse) # or mse**(0.5)
         r2 = metrics.r2_score(y_test,y_pred2)
         print("Results of sklearn.metrics:")

         Results of sklearn.metrics:
```

```
In [38]: print("MAE:",mae)
         print("MSE:", mse)
         print("RMSE:", rmse)
         print("R-Squared:", r2)

         MAE: 7.359178096328714
         MSE: 109.64932238372614
         RMSE: 10.471357236945273
         R-Squared: 0.7664363374371922
```

```
In [39]: df_actual_pre2 ={'Actual': y_test, 'Predicted': y_pred2}
         df1_actual_pre2= pd.DataFrame(df_actual_pre2)
         df1_actual_pre2.head(20)
```

Out[39]:

	Actual	Predicted
34530	18	30.988331
54303	48	52.306681
119544	11	1.143507
132124	39	40.575408
165993	29	40.039180
57936	21	33.388216
23998	61	73.964072
58760	33	35.864485
86134	59	41.249233
35284	22	34.873565
95749	60	41.314982
54368	40	46.779063
[illegible]	[illegible]	[illegible]



SVR MODEL

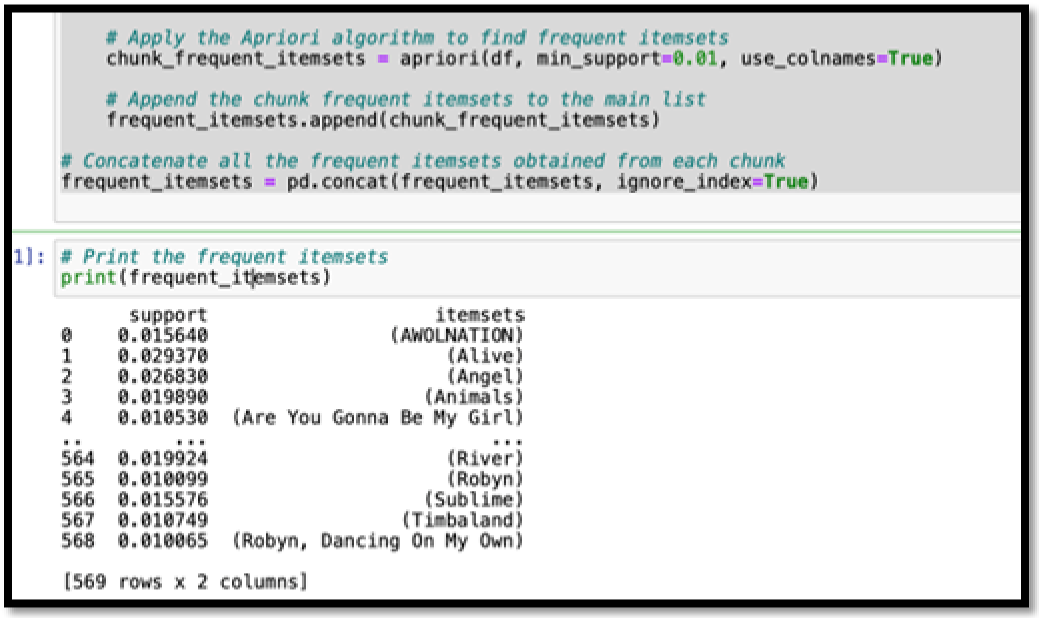
```
In [41]: from sklearn.svm import SVR
```

```
In [42]: svr = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
         svr.fit(X_train, y_train)
         pre_svr=svr.predict(X_test)
```

```
In [43]: mae = metrics.mean_absolute_error(y_test, pre_svr)
         mse = metrics.mean_squared_error(y_test, pre_svr)
         rmse = np.sqrt(mse) # or mse**(0.5)
         r2 = metrics.r2_score(y_test,pre_svr)
         print("Results of sklearn.metrics:")
         print("MAE:",mae)
         print("MSE:", mse)
         print("RMSE:", rmse)
         print("R-Squared:", r2)

         Results of sklearn.metrics:
         MAE: 6.273311999279473
         MSE: 88.13589973994408
         RMSE: 9.388072205727013
         R-Squared: 0.8122620085649979
```

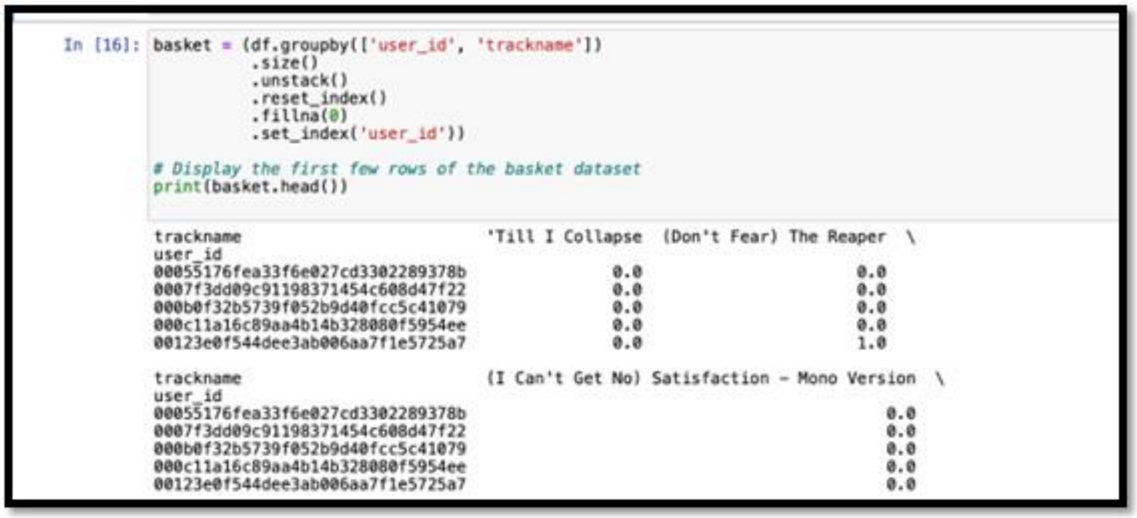
```
In [45]: df_actual_pre2 ={'Actual': y_test, 'Predicted': pre_svr}
         df1_actual_pre2= pd.DataFrame(df_actual_pre2)
         df1_actual_pre2.head(20)

Out[45]:
                 Actual   Predicted
          34530      18   22.955360
          54303      46   47.583934
         119544      11    3.700032
         132124      39   32.684544
         165993      29   30.874644
          57936      21   36.409279
          23998      61   67.617047
          58760      33   30.871646
          86134      59   46.166810
          35284      22   31.761356
          95749      60   43.884889
          54368      40   49.643063
          51124      23   33.080096
          17320       0    0.148379
         156707      31   31.645692
         140251      38   31.896423
          14000      43   46.244926
          91780      33   18.444947
         121942      39   32.266007
          42729      16   33.472948
```

Market basket analysis

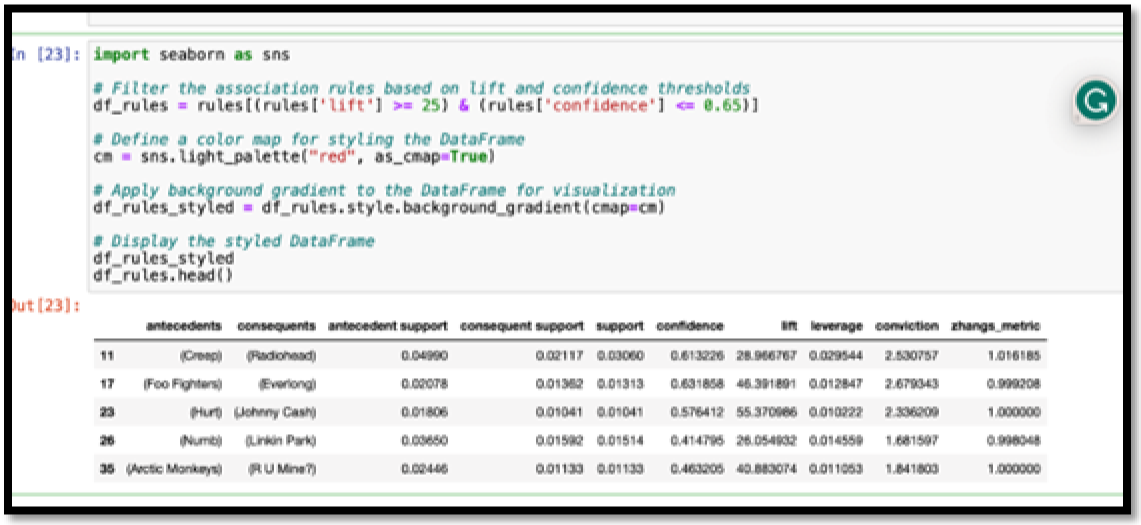


```
    # Apply the Apriori algorithm to find frequent itemsets
    chunk_frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

    # Append the chunk frequent itemsets to the main list
    frequent_itemsets.append(chunk_frequent_itemsets)

# Concatenate all the frequent itemsets obtained from each chunk
frequent_itemsets = pd.concat(frequent_itemsets, ignore_index=True)
```

```
1]: # Print the frequent itemsets
    print(frequent_itemsets)

         support                      itemsets
    0    0.015640                  (AWOLNATION)
    1    0.029370                       (Alive)
    2    0.026830                       (Angel)
    3    0.019890                     (Animals)
    4    0.010530     (Are You Gonna Be My Girl)
    ..        ...                           ...
    564  0.019924                       (River)
    565  0.010099                       (Robyn)
    566  0.015576                     (Sublime)
    567  0.010749                   (Timbaland)
    568  0.010065     (Robyn, Dancing On My Own)

    [569 rows x 2 columns]
```



```
In [16]: basket = (df.groupby(['user_id', 'trackname'])
                  .size()
                  .unstack()
                  .reset_index()
                  .fillna(0)
                  .set_index('user_id'))

         # Display the first few rows of the basket dataset
         print(basket.head())

         trackname                         'Till I Collapse  (Don't Fear) The Reaper  \
         user_id
         00055176fea33f6e027cd3302289378b               0.0                      0.0
         0007f3dd09c91198371454c608d47f22               0.0                      0.0
         000b0f32b5739f052b9d40fcc5c41079               0.0                      0.0
         000c11a16c89aa4b14b328080f5954ee               0.0                      0.0
         00123e0f544dee3ab006aa7f1e5725a7               0.0                      1.0

         trackname                         (I Can't Get No) Satisfaction - Mono Version  \
         user_id
         00055176fea33f6e027cd3302289378b                                           0.0
         0007f3dd09c91198371454c608d47f22                                           0.0
         000b0f32b5739f052b9d40fcc5c41079                                           0.0
         000c11a16c89aa4b14b328080f5954ee                                           0.0
         00123e0f544dee3ab006aa7f1e5725a7                                           0.0
```

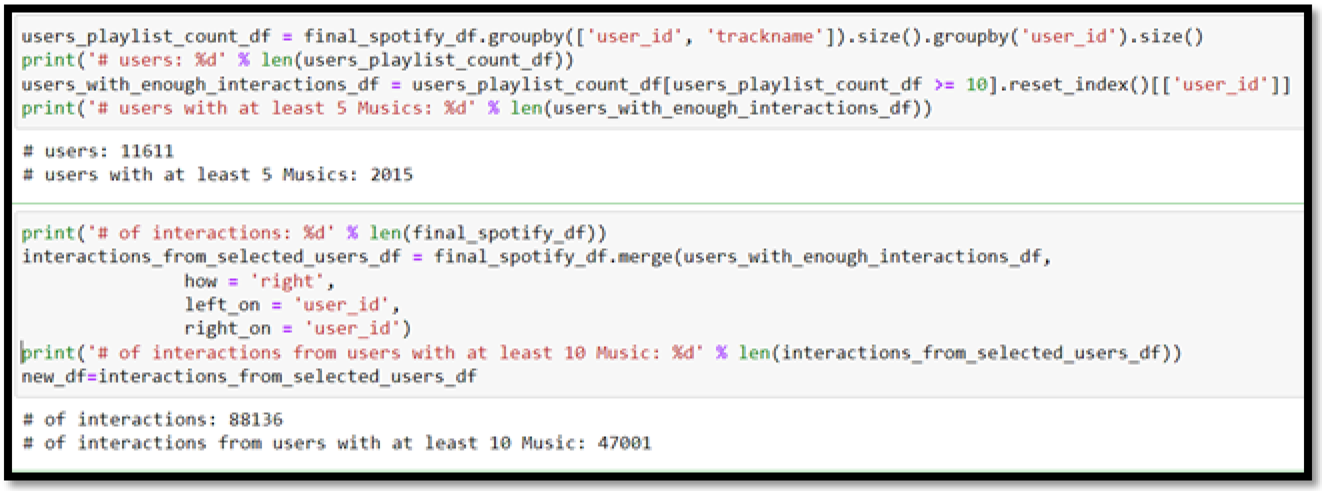


```
In [23]: import seaborn as sns

         # Filter the association rules based on lift and confidence thresholds
         df_rules = rules[(rules['lift'] >= 25) & (rules['confidence'] <= 0.65)]

         # Define a color map for styling the DataFrame
         cm = sns.light_palette("red", as_cmap=True)

         # Apply background gradient to the DataFrame for visualization
         df_rules_styled = df_rules.style.background_gradient(cmap=cm)

         # Display the styled DataFrame
         df_rules_styled
         df_rules.head()
```

Out[23]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
11	(Creep)	(Radiohead)	0.04990	0.02117	0.03060	0.613226	28.966767	0.029544	2.530757	1.016185
17	(Foo Fighters)	(Everlong)	0.02078	0.01362	0.01313	0.631858	46.391891	0.012847	2.679343	0.999208
23	(Hurt)	(Johnny Cash)	0.01806	0.01041	0.01041	0.576412	55.370986	0.010222	2.336209	1.000000
26	(Numb)	(Linkin Park)	0.03650	0.01592	0.01514	0.414795	26.054932	0.014559	1.681597	0.998048
35	(Arctic Monkeys)	(R U Mine?)	0.02446	0.01133	0.01133	0.463205	40.883074	0.011053	1.841803	1.000000

Recommendation system.

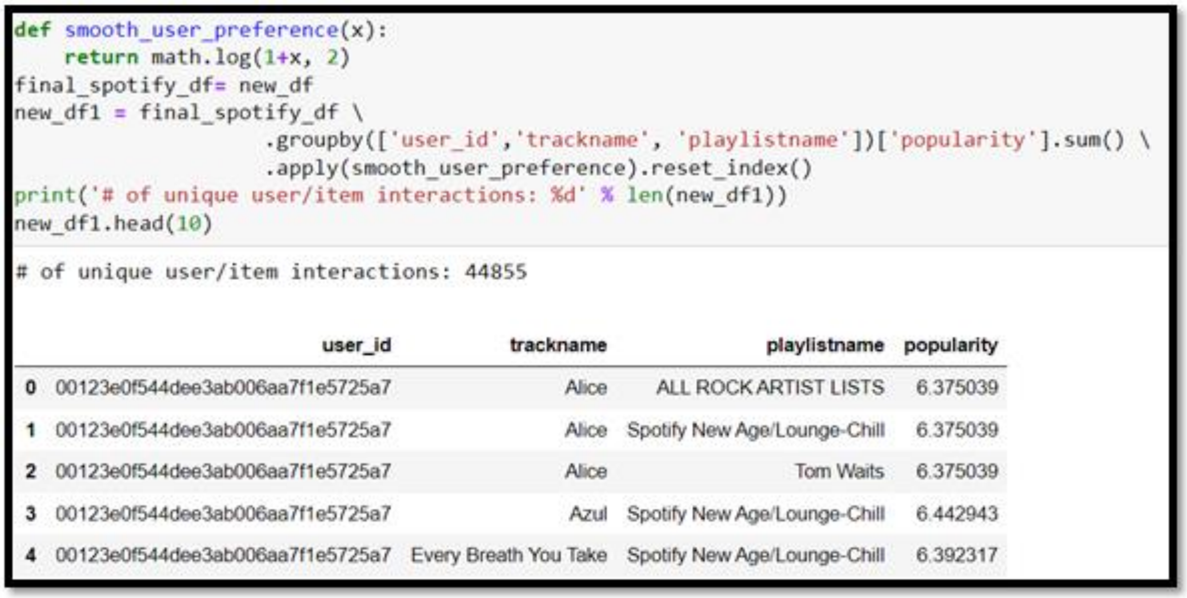


```
users_playlist_count_df = final_spotify_df.groupby(['user_id', 'trackname']).size().groupby('user_id').size()
print('# users: %d' % len(users_playlist_count_df))
users_with_enough_interactions_df = users_playlist_count_df[users_playlist_count_df >= 10].reset_index()[['user_id']]
print('# users with at least 5 Musics: %d' % len(users_with_enough_interactions_df))
```

```
# users: 11611
# users with at least 5 Musics: 2015
```

```
print('# of interactions: %d' % len(final_spotify_df))
interactions_from_selected_users_df = final_spotify_df.merge(users_with_enough_interactions_df,
               how = 'right',
               left_on = 'user_id',
               right_on = 'user_id')
print('# of interactions from users with at least 10 Music: %d' % len(interactions_from_selected_users_df))
new_df=interactions_from_selected_users_df
```

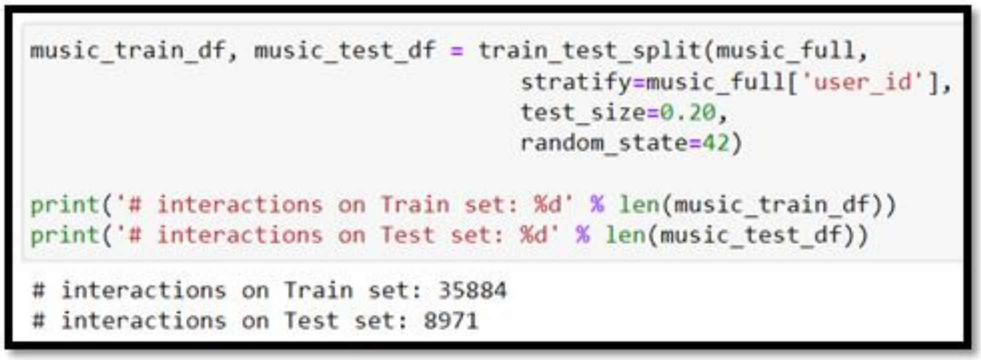
```
# of interactions: 88136
# of interactions from users with at least 10 Music: 47001
```



```
def smooth_user_preference(x):
    return math.log(1+x, 2)
final_spotify_df= new_df
new_df1 = final_spotify_df \
                      .groupby(['user_id','trackname', 'playlistname'])['popularity'].sum() \
                      .apply(smooth_user_preference).reset_index()
print('# of unique user/item interactions: %d' % len(new_df1))
new_df1.head(10)
```

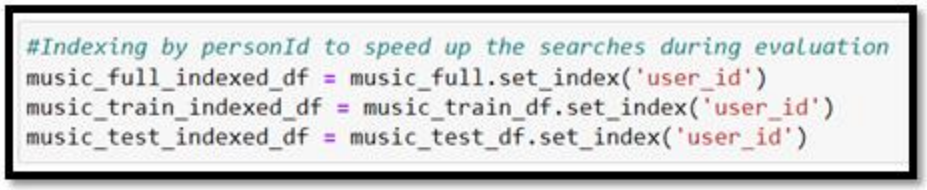
```
# of unique user/item interactions: 44855
```

	user_id	trackname	playlistname	popularity
0	00123e0f544dee3ab006aa7f1e5725a7	Alice	ALL ROCK ARTIST LISTS	6.375039
1	00123e0f544dee3ab006aa7f1e5725a7	Alice	Spotify New Age/Lounge-Chill	6.375039
2	00123e0f544dee3ab006aa7f1e5725a7	Alice	Tom Waits	6.375039
3	00123e0f544dee3ab006aa7f1e5725a7	Azul	Spotify New Age/Lounge-Chill	6.442943
4	00123e0f544dee3ab006aa7f1e5725a7	Every Breath You Take	Spotify New Age/Lounge-Chill	6.392317

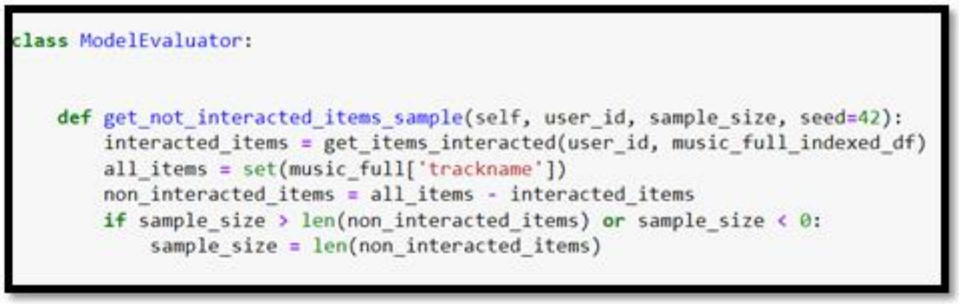


```
music_train_df, music_test_df = train_test_split(music_full,
                                   stratify=music_full['user_id'],
                                   test_size=0.20,
                                   random_state=42)

print('# interactions on Train set: %d' % len(music_train_df))
print('# interactions on Test set: %d' % len(music_test_df))
```

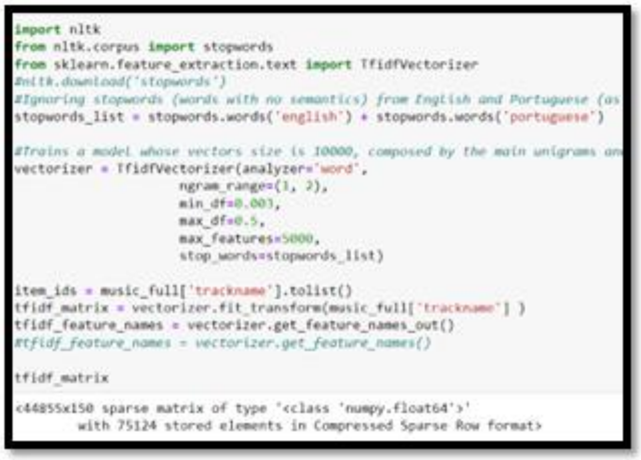
```
# interactions on Train set: 35884
# interactions on Test set: 8971
```



```
#Indexing by personId to speed up the searches during evaluation
music_full_indexed_df = music_full.set_index('user_id')
music_train_indexed_df = music_train_df.set_index('user_id')
music_test_indexed_df = music_test_df.set_index('user_id')
```

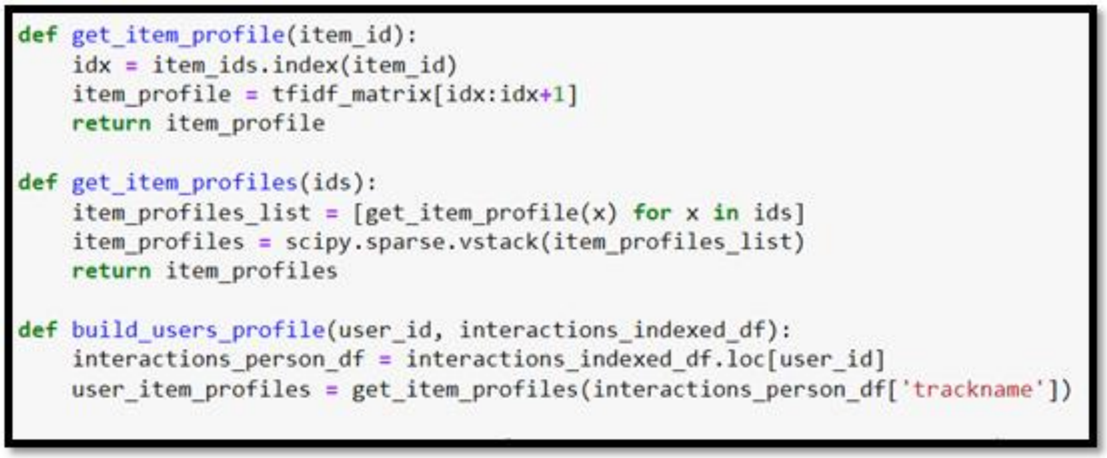


```python
class ModelEvaluator:


    def get_not_interacted_items_sample(self, user_id, sample_size, seed=42):
        interacted_items = get_items_interacted(user_id, music_full_indexed_df)
        all_items = set(music_full['trackname'])
        non_interacted_items = all_items - interacted_items
        if sample_size > len(non_interacted_items) or sample_size < 0:
            sample_size = len(non_interacted_items)
```

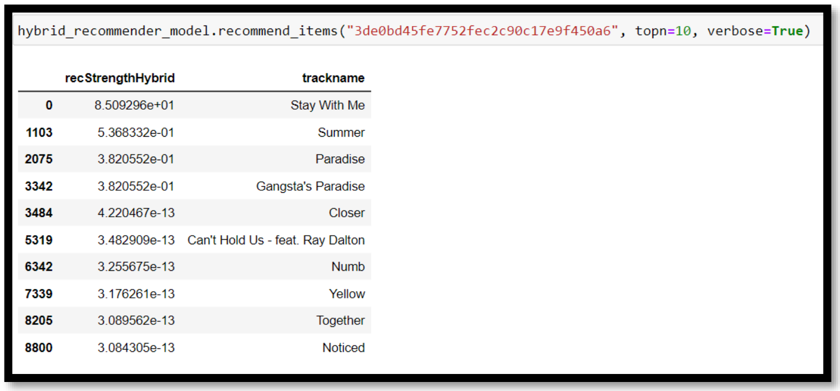


```python
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
#nltk.download('stopwords')
#Ignoring stopwords (words with no semantics) from English and Portuguese (as
stopwords_list = stopwords.words('english') + stopwords.words('portuguese')

#Trains a model whose vectors size is 10000, composed by the main unigrams and
vectorizer = TfidfVectorizer(analyzer='word',
                     ngram_range=(1, 2),
                     min_df=0.003,
                     max_df=0.5,
                     max_features=5000,
                     stop_words=stopwords_list)

item_ids = music_full['trackname'].tolist()
tfidf_matrix = vectorizer.fit_transform(music_full['trackname'] )
tfidf_feature_names = vectorizer.get_feature_names_out()
#tfidf_feature_names = vectorizer.get_feature_names()

tfidf_matrix
```

```
<44855x150 sparse matrix of type '<class 'numpy.float64'>'
	with 75124 stored elements in Compressed Sparse Row format>
```



```python
def get_item_profile(item_id):
    idx = item_ids.index(item_id)
    item_profile = tfidf_matrix[idx:idx+1]
    return item_profile

def get_item_profiles(ids):
    item_profiles_list = [get_item_profile(x) for x in ids]
    item_profiles = scipy.sparse.vstack(item_profiles_list)
    return item_profiles

def build_users_profile(user_id, interactions_indexed_df):
    interactions_person_df = interactions_indexed_df.loc[user_id]
    user_item_profiles = get_item_profiles(interactions_person_df['trackname'])
```



```python
hybrid_recommender_model.recommend_items("3de0bd45fe7752fec2c90c17e9f450a6", topn=10, verbose=True)
```

	recStrengthHybrid	trackname
0	8.509296e+01	Stay With Me
1103	5.368332e-01	Summer
2075	3.820552e-01	Paradise
3342	3.820552e-01	Gangsta's Paradise
3484	4.220467e-13	Closer
5319	3.482909e-13	Can't Hold Us - feat. Ray Dalton
6342	3.255675e-13	Numb
7339	3.176261e-13	Yellow
8205	3.089562e-13	Together
8800	3.084305e-13	Noticed