UNIVERSITY OF CRAIOVA
FACULTY OF AUTOMATION, COMPUTERS AND ELECTRONICS

DEPARTMENT OF COMPUTERS AND INFORMATION TECHNOLOGY

DEGREE PROJECT

Ionuț – Dragoș Neremzoiu

SCIENTIFIC COORDINATOR

Prof. Univ. Dr. Ing. Costin Bădică

JULY 2022

CRAIOVA

UNIVERSITY OF CRAIOVA
FACULTY OF AUTOMATION, COMPUTERS AND ELECTRONICS

DEPARTMENT OF COMPUTERS AND INFORMATION TECHNOLOGY

Music Recommender System

Ionuț – Dragoș Neremzoiu

SCIENTIFIC COORDINATOR

Prof. Univ. Dr. Ing. Costin Bădică

JULY 2022

CRAIOVA

„Whatever the mind of man can conceive and believe, it can achieve."

Napoleon Hill

DECLARAȚIE DE ORIGINALITATE

Subsemnatul Neremzoiu Ionuț-Dragoş, student la specializarea Calculatoare (în limba engleză) din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoştinţă de cele prezentate mai jos şi că îmi asum, în acest context, originalitatea proiectului meu de licenţă:

- cu titlul Music Recommender System,
- coordonată de Prof. Univ. Dr. Ing. Costin Bădică,
- prezentată în sesiunea IULIE 2022

La elaborarea proiectului de licenţă, se consideră plagiat una dintre următoarele acţiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele şi referinţa precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obţinute sau a unor aplicaţii realizate de alţi autori fără menţionarea corectă a acestor surse,
- însuşirea totală sau parţială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situaţii neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe şi indicarea referinţei într-o listă corespunzătoare la sfărşitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii şi corespunzător în lista de referinţe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referinţelor poate fi omisă dacă se folosesc informaţii sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,   Semnătura candidatului,

23.06.2022   [illegible]

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Calculatoare şi Tehnologia Informaţiei

Aprobat la data de
26.06.2022
Şef de departament,
Şef lucrări dr. ing.
Nicolae Iulian
ENESCU

PROIECTUL DE DIPLOMĂ

Numele și prenumele studentului:	Neremzoiu Ionuț-Dragoș
Enunțul temei:	*Music Recommender System/ Experimental Music Recommender System which is meant to simulate a scenario in which the user listens to a song and recommedations based on that song pop up.*
Datele de pornire:	2 different Spotify dataset of songs with both common and different data columns
Conținutul proiectului:	1. Introduction 2. Recommendation algorithms 3. Design and implementation of the music recommender system using content-based approaches 4. Implementation and evaluation of the experimental system 5. Conclusions and suggestions on improvements
Material grafic obligatoriu:	Prezentare Power-Point, grafice, figuri, screenshot-uri, cod sursă
Consultații:	*Periodice*
Conducătorul științific (titlul, nume și prenume, semnătura):	Prof. Univ. Dr. Ing. Costin Bădică
Data eliberării temei:	15.10.2021
Termenul estimat de predare a proiectului:	23.06.2022
Data predării proiectului de către student și semnătura acestuia:	23.06.2022 [illegible]

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Calculatoare şi Tehnologia Informaţiei

REFERATUL CONDUCĂTORULUI ŞTIINȚIFIC

Numele şi prenumele candidatului:	Neremzoiu Ionuț-Dragoș
Specializarea:	Calculatoare şi Tehnologia Informaţiei (în lb. engleză)
Titlul proiectului:	Music Recommender System
Locaţia în care s-a realizat practica de documentare (se bifează una sau mai multe din opţiunile din dreapta):	În facultate X În producţie ☐ În cercetare ☐ Altă locaţie:

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine X
Tipul proiectului		Cercetare X	Proiectare ☐	Realizare practică ☐	Altul
Aparatul matematic utilizat		Simplu ☐	Mediu ☐	Complex X	Absent ☐
Utilitate		Contract de cercetare ☐	Cercetare internă X	Utilare ☐	Altul
Redactarea lucrării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine X
Partea grafică, desene		Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună X
Realizarea practică	Contribuţia autorului	Insuficientă ☐	Satisfăcătoare ☐	Mare ☐	Foarte mare X
	Complexitatea temei	Simplă ☐	Medie ☐	Mare ☐	Complexă X
	Analiza cerinţelor	Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine X
	Arhitectura	Simplă ☐	Medie ☐	Mare ☐	Complexă X
	Întocmirea specificaţiilor funcţionale	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună X

	Implementarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună X
	Testarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună X
	Funcționarea	Da X	Parțială ☐	Nu ☐	
Rezultate experimentale		Experiment propriu X		Preluare din bibliografie ☐	
Bibliografie		Cărți x	Reviste x	Articole x	Referințe web x
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI	RESPINGEREA PROIECTULUI
X	☐

Data,

26.06.2022

Semnătura conducătorului științific,

[illegible]

PROJECT SUMMARY

The purpose of this project was to develop an experimental music recommender system which can make the most relevant possible recommendations. For this project I managed to collect as much data as possible about songs, however I didn’t manage to find some helpful data regarding users’ tastes and interactions with songs, such that I could properly create ultimately a recommender system based on a hybrid approach. Therefore, the recommendations are content-based only, since the song similarities are determined on audio characteristics, mood values, lyrics similarities and popularity ratings.

Termenii cheie: PCA, content-based, recommender system, k-NN, K-Means, TF-IDF, Machine Learning, Min-Max scaling, Cosine similarity.

ACKNOWLEDGEMENTS

I would like to thank my coordonating professor, Prof. PhD. Eng. Costin Bădică, for suggesting me the research topic that I ended up choosing, for guiding me throughout the process whenever I encountered difficulties on the project and requested help, for the patience and advice he offered.

I would also like to thank the people who adviced me on collaborating with my coordinating professor on my Bachelor’s Degree project.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Context

Since the rise of Amazon, Netflix, Youtube and many other such web services, recommender systems have increasingly become a part of our daily lives. From e-commerce platforms, which suggest to buyers articles that could interest them, to online advertisements, which suggest to users relevant products, matching their preferences, recommender systems are nowadays unavoidable in our daily online journeys.

Therefore, there is all the more reason not only for any software developer, but also for any end-user, which spends his or her time browsing online, to have basic general knowledge about how such systems work. By getting to know how a recommender system works, one can have a general idea of its limitations and may make use of its underlying algorithm in order to be provided with the best and most relevenat recommendations in accordance with one's preferences.

1.2 Motivation

As far as I am concerned, there are three main aspects that determined me to pursue my own research on this field.

To begin with, one of my day-to-day activities is listening to music, usually on Youtube, while also studying. At some point, I noticed how Youtube, for instance, provided me with its custom-made playlists, which consisted mainly of songs that I played more than several times and over relatively short time spans. This type of systematic behaviour made me curious about how Youtube was able to determine a pattern of what I have been listening over specific time spans and eventually I found out about the fact that Youtube has various underlying recommendation algorithms.

Secondly, the moment I found out that I can have this topic as research project for my Bachelor's Degree, even though I had no prior further knowledge or experience in developing a recommendation system, I perceived this as an opportunity in aquiring knowledge on this topic and learning how to develop my own recommandation system.

At last, but not least, this field has strong connections with machine learning, which has been become a research field for the first time since the 40s, when it was first conceived from the mathematical modeling of neural networks in an attempt to mathematically map out thought processes and decision making in human cognition. Therefore, recommender systems along with machine learning provide various research papers, articles or blogs to acquire knowledge from and at the same time countless of possibilities to further reach new findings on this field that can have a great impact on our everyday lives.

1.3 Document Overview

It consists of 5 main chapters:

1) **Introduction** : In this chapter it is described the **context** in which the research project involves around, the **motivation** that lay behind my decision to choose this topic for my research project and, finally, the **project overview** in which the project content is summarized.

2) **Recommendation algorithms**: In this chapter it is briefly discussed **what recommandation algorithms are**, it is presented a **classification of recommendation methods** and, finally, an **evaluation of recommendation algorithms**.

3) **Design and implementation of a recommendation algorithm for music songs using *content-based* approach**: This chapter comprises the following sections :

   - Block diagram of the experimental system : This diagram is meant to ilustrate the workflow of the main processes of the system (*Data preparation*, *Preprocessing*, *Recommendation* and *Evaluation*).
   - Data sources : In this section it is specified the origin of the collections of music data and statistical visualizations on the collected data.
   - Dataset preparation : This section is divided in *Data Loading*, *Data filtering and cleaning, Merging* and *Lyrics Collection*.
   - Dataset used : In this section it is mainly focused on describing thoroughly the kind of data we are dealing with and how big it is.
   - Preprocessing algorithm: In this section the *normalization*, *dimensionality reduction* and *clustering* processes are described.
   - Recommendation algorithm : This section is meant to showcase how k-NN (a popular supervised machine-learnig algorithm) is used along with Tf-Idf (a popular algorithm used in Information Retrieval field) in order to produce a specified number of recommendations.

4) Implementation and evaluation of the experimental system : This chapter comprises the following sections :

   - Libraries and platforms used in the process of the recommender system development
   - Modules used for data visualisation and showcasing experimental results
   - Experimental results and evaluation of the experimental system

5) Conclusions and future works : This is the final chapter in which I highlight the conclusions that I reached after completing my research project and the improvements that could bring an impactful effect on the performance of the recommender system.

2 RECOMMENDATION ALGORITHMS

2.1 What are recommendation algorithms?

,,In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (the items can be movies to watch, text to read, products to buy or anything else depending on the industries).'' [Roc19]

The way such systems achieve recommending the appropiate relevant items to users depends heavily on multiple factors such as the dimensionality or dimension of the data involved and the approach that the system takes in making the recommandations. For instance, nowadays many companies make use of Big Data in order to produce recommendations as relevant as possible. Such recommendations can have a huge impact on users' decision in continuing or not continuing to purchase products or services of a specific company, which simultaneously can make a significant difference in the growth revenue of that company.

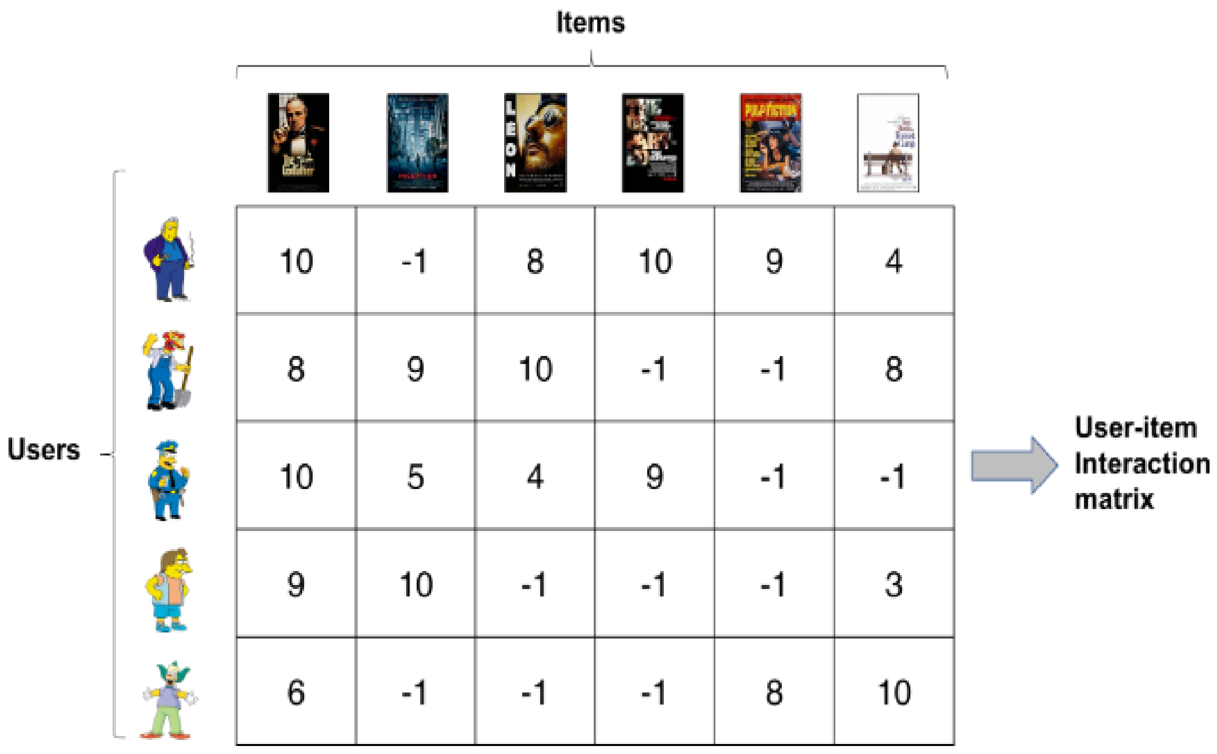
This is why, among a variety of recommendaiton algorithms, the data scientists need to choose the best one according to the limitations and requirement of a business.

2.2 Classification of recommendation algorithms

When it comes to classifying recommendation algorithms, there are two major paradigms of recommender systems : *collaborative* and *content* based methods. Both pardagims have their own weaknesses and strengths, so it’s up to the system arhitects and data scientists to decide on their recommendation strategy, considering the kind of data available at their disposal, the evaluation of various recommendation algorithms and the limitations each approach has.

2.2.1 Collaborative filtering methods

Collaborative methods [Roc19] for recommender systems are methods that rely solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called “user-item interactions matrix”, as illustrated in **Figure 1**.



10	-1	8	10	9	4
8	9	10	-1	-1	8
10	5	4	9	-1	-1
9	10	-1	-1	-1	3
6	-1	-1	-1	8	10

Figure 1 Example of user-item interaction matrix [BK20]

The main idea that rules collaborative methods is that these past user-item interactions are enough to detect similar users and/or similar items and make predictions based on these proximities.

The collaborative filtering algorithms can be further divided into two sub-categories that are generally called *memory-based* and *model-based* approaches. Memory-based approaches directly work with values of recorded interactions (assuming there is no model involved) and are essentially based on nearest neighbours search (for instance, find the closest users from a user of interest and suggest the most popular items among these neighbours). In contrast to *memory-based* approaches, *model-based* approaches assume an underlying “generative” model that explains the user-item interactions and tries to discover it to a greater extent in order to make new predictions.

The main advantage of collaborative approaches is that they require no information about users or items and, therefore, can be used in many scenarios. Moreover, the more users interact with items the more accurate new recommendations become. As a result, for a fixed set of users and items, new interactions recorded over time bring new information and make the system more and more effective.

However, since it only considers past interactions to make recommendations, the collaborative filtering strategy suffers from “the cold start problem”. This particular problem refers to the fact that it is impossible to recommend anything to new users or to recommend a new item to any users and many users or items have too few interactions to be properly handled. This drawback can be addressed in various ways:

- Random strategy (recommending random items to new users or new items to random users)
- Maximum expectation strategy (recommending popular items to new users or new items to most active users)
- Exploratory strategy (recommending a set of various items to new users or a new item to a set of various users)
- Using a non-collaborative method for the early life of the user or the item.

2.2.2 Content-based methods

Unlike collaborative methods that rely on the user-item interactions, content-based approaches use additional information about users and/or items. For instance, considering the presented context of a music recommender system, this additional information can be duration, genre, release year or any typical characteristic (acousticness, loudness, instrumentalness, etc.) for songs as well as the age, gender, geolocation or any other personal user information.

The idea of such methods is to try to build a model based on the available “features” that explains or attempts to explain the observed user-item interactions. Coming back to our definite context of music recommender system, the goal could be to model the fact that usually young people tend to listen to pop-genre music more than other genres, whereas more elderly people listen, for instance, to jazz and classical music more than other genres. Once we manage to get such model, then making music recommendations becomes much easier. The user profile (age, gender, geolocation, etc.) is analyzed and based on the user profile information relevant songs can be suggested. Another approach would be to interpret music characteristics of songs in order to build a model. For instance, usually any *Metallica* song is similar to any *Iron Maiden* song, since both can be classified as rock music. Obviously, the metric by which can compute a similarity has endless possibilities ranging from song duration, acousticness, loudness, instrumentalness, etc. Bottom line is, as presented in the previous example, an item-based profile can be built instead and based on that profile recommendations can be generated. The general behaviour of such recommender systems can be illustrated in **Figure 2** below.

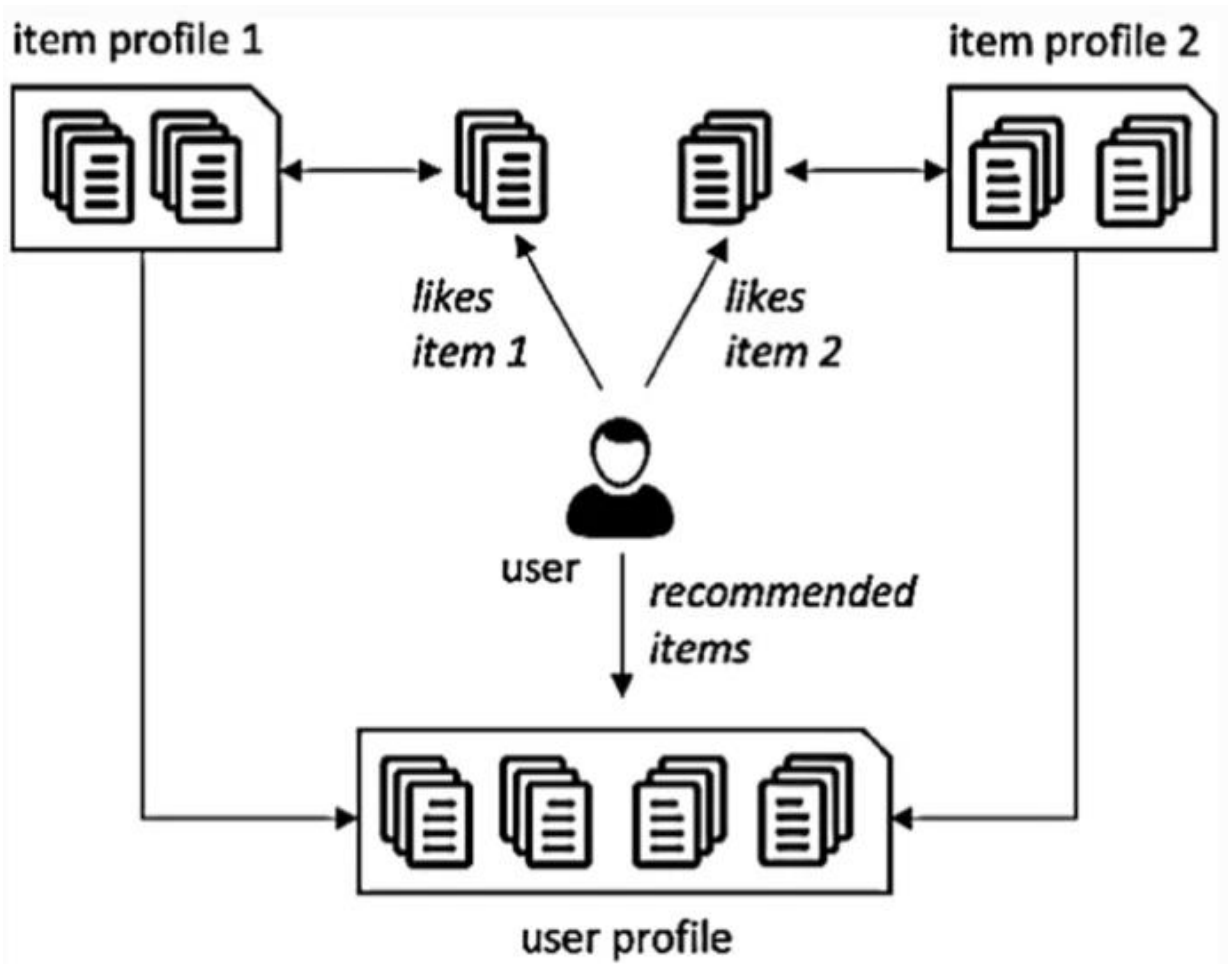


Figure 2. Illustration of a content-based recommender system [DR22]

Unlike *collaborative filtering* approaches, content-based methods suffer far less from the “cold start” problem: new users or items can be described by their characteristics (content) and thus relevant suggestions can be done for these new entities. Only new users or items with previously unseen features will logically suffer from this drawback, but once the system gets accustomed with these features, it is very unlikely to happen.

However, there are still some major disadvantages with these methods. Feature representation of items is hand-engineered to some extent, which requires a lot of domain knowledge and the model can only make recommendations based on the existing interest of a user. In other words, the model has limited ability to expand on the user's existing interests.

In content-based methods, the recommendation problem is casted into either a classification problem (predicting if a user "likes" or not an item) or into a regression problem (predicting the rating given by a user to an item). In both cases, a model based on the user and/or the item features will be built and used.

If the classification (or regression) relies on user features, it is said that the approach is item-centered. In other words, modelling, optimizations and computations can be done "by item". When building such model based on user features, what we are actually trying to do is to answer either the question "what is the probability for each user to like this item?" (for classification) or "what is the rate given by each user to this item?" (for regression). The model associated to each item is naturally trained on data related to this item and it leads, in general, to pretty robust models as a lot of users have interacted with the item. However, the interactions considered for the model to learn come from every user and even if these users have similar characteristics (features), their preferences can be different. This means that even if this method is more robust, it can be considered as being less personalized (more biased) than the user-centered method.

If we are working with item features, the method is then considered user-centered. In other words, the modelling, optimizations and computations can be done "by user". When building such model on item features, what we are actually trying to do is to answer the question "what is the probability for this user to like each item?" (for classification) or "what is the rate given by this user to each item" (for regression). The model can then be attached to each user that is trained on its data: the model obtained is, therefore, more personalized than its item-centered counterpart as it only takes into account interactions from the considered user. However, most of the time a user has interacted with relatively few items and, therefore, the model obtained is far less robust than an item-centered one.

In order to have a more realistic visualization, **Figure 3** can be consulted on how both item-centered and user-centered strategies work.

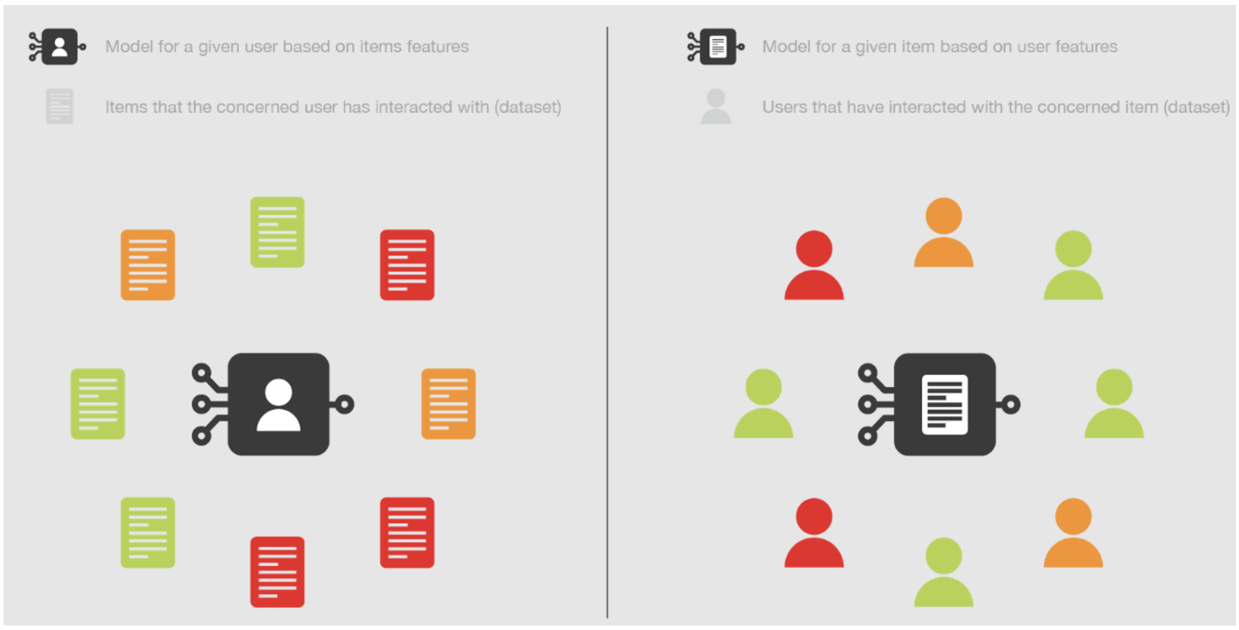

Model for a given user based on items features
Items that the concerned user has interacted with (dataset)
Model for a given item based on user features
Users that have interacted with the concerned item (dataset)

Figure 3. Illustration of the difference between item-centered and user-centered content-based methods [Roc19]

From a practical point of view, it should be underlined that, most of the time, it is much more difficult to ask a new user for some infromation (usually users don't want to answer too many questions) than to ask information about a new item (users buying items in a chart for example can be translated into information of those items which can become recommendation to potential new users). It can also be noticed that, depending on the complexity of the relation to the individual, the potential model can be more or less complex, ranging from basic models (logistic/linear regression for classification/regression) to deep neural networks. Apart from that, content-based methods can also be either user or item centered : both user and item information can be used for modelling process, for instance by stacking two feature vectors and make them undergo a neural network arhitecure.

2.3 Evaluation of recommendation algorithms

As for any machine learning algortihm, we need to be able to evaluate the performances of our recommender systems in order to decide which algorithm fits the best our situation. Evaluation methods for recommender systems can be mainly be divided in two categories: evaluation based on well defined **metrics** and evaluation mainly based on ***human judgement*** and ***satisfaction*** **estimation**.

2.3.1 Metrics based evaluation

If our recommender system is based on a model that outputs numeric values such as ratings predictions or matching probabilities, the quality of these outputs can be assessed in a very classical manner using an error measurement metric such as, for instance, **mean square error** (**MSE**). In this case, the model is trained only on a part of the available interactions and is tested on the remaining ones.

Even so, if the recommender system is based on a model that predicts numeric values, these values can be binarized with a classical thresholding approach (values above the threshold are positive and values below are negative) and this way the model can be evaluated in a more “classification-like way”. As such, as the dataset of user-item past interactions is also binary (or can be binarized by thresholding), the accuracy (as well as the precision and the recall) of the binarized outputs of the model can be evaluated on a test dataset of interactions not used for training.

At last, but not least, if a recommender system, which is not based on numeric values, is taken into consideration and it only returns a list of recommendations (such as user-user or item-item that are based on a k-NN approach), a precision metric can still be defined by estimating the proportion of recommended items that really suit our user. To estimate this precision, recommended items that our user has not interacted with cannot be taken into account and only items from the test dataset for which user feedback exists should be considered.

2.3.2 Human based evaluation

When designing a recommender system, we shouldn’t only be interested in how to obtain a model that produces recommendations that we are very sure about, but we should also be interested in some other properties such as diversity and explainability of recommendations.

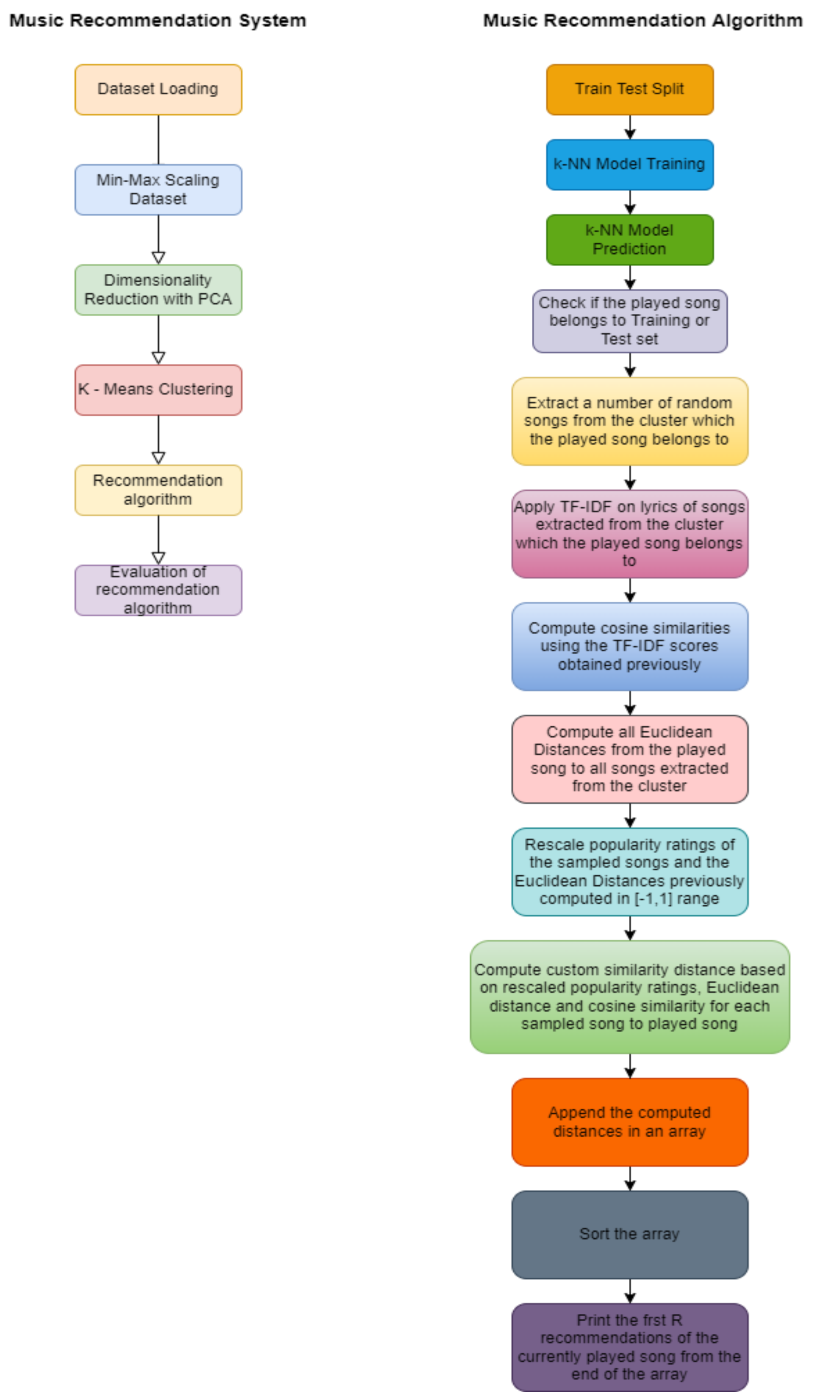
As previously mentioned in the collaborative section, it is absolutely needed to avoid having a user stuck in an “information confinement area”. The notion of “serendipity” is often used to express the tendency a model has or not to create such a confinement area (diversity of recommendations). Serendipity, which can be estimated by computing the distance between recommended items, should not be too low as it would create confinement areas, but should also not be too high as it would mean that we do not take enough into account our users’ interests when making recommendations. In other words, we are dealing with an “exploration vs exploitation” problem. Thus, in order to bring out diversity in the suggested choices, it is desirable to recommend items that suit both suit our user very well and are not too similar from each other. For instance, instead of suggesting a user “Star Wars” 1,2 and 3 movies, it seems better to only recommend “Star Wars 1” along with suggestions such as “Star Trek into darkness” and “Indiana Jones and the raiders of the lost ark”. The latter two may be seen by our system as having less chance to interest our user, but recommending 3 items that look too similar might not be a good option either.

Explainability is another key point of the success of recommendations algorithms. Indeed, it has been proven that if users don’t understand why they had been recommended a specific item, they tend to lose confidence into the recommender system. Therefore, if a model that is clearly explainable can be designed, when making recommendations, a little sentence can be added stating why an item has been recommended (“people who liked this item also liked this one”, “you liked this item, you may be interested by this one”, etc.).

At last, but not least, on top of the fact that diversity and explainability can be intrinsically difficult to evaluate, it can be noticed that it is also pretty difficult to assess the quality of a recommendation that doesn’t belong to the testing dataset : how can it be known if a new recommendation is relevant before actually recommending it to our user? For all these reasons, it can sometimes be tempting to test the model in “real conditions”. As the goal of the recommender system is to generate an action (watch a movie, buy a product, read an article etc.), its ability to generate the expected action can be indeed evaluated. For example, the system can be put in production, following an A/B testing approach or can be tested on a sample of users. Such testing processes require, however, having a certain level of confidence in the model.

3 DESIGN AND IMPLEMENTATION OF THE MUSIC RECOMMENDER SYSTEM USING CONTENT-BASED APPROACH

3.1 Block diagram of the experimental system



Music Recommendation System
Dataset Loading
Min-Max Scaling Dataset
Dimensionality Reduction with PCA
K - Means Clustering
Recommendation algorithm
Evaluation of recommendation algorithm
Music Recommendation Algorithm
Train Test Split
k-NN Model Training
k-NN Model Prediction
Check if the played song belongs to Training or Test set
Extract a number of random songs from the cluster which the played song belongs to
Apply TF-IDF on lyrics of songs extracted from the cluster which the played song belongs to
Compute cosine similarities using the TF-IDF scores obtained previously
Compute all Euclidean Distances from the played song to all songs extracted from the cluster
Rescale popularity ratings of the sampled songs and the Euclidean Distances previously computed in [-1,1] range
Compute custom similarity distance based on rescaled popularity ratings, Euclidean distance and cosine similarity for each sampled song to played song
Append the computed distances in an array
Sort the array
Print the frst R recommendations of the currently played song from the end of the array

Figure 4. Block diagram of both music recommendation system and recommendation algorithm

3.2 Data sources

The two datasets used for the music recommender system are Spotify data music collections scrapped together by two different contributors (potential developers) using *Spotipy* API. Both datasets have online public access : one dataset can be found on **Jovian.ai** platform [XZ21] and the other one can be found on a public-available Github repository [RD20].

,,**Jovian** is a community-driven learning platform for data science and machine learning, where apart from taking online courses, bulding real-world projects and interacting with a global community” [J], people can also publicly share their notebooks along with the datasets used.

,,**Spotipy** is a lightweight Python library for the Spotify Web API. With Spotipy you get full access to all of the music data provided by the Spotify platform’’ [PL]. In order to use the library, it is needed to generate a *Spotipy Client ID* and Spotipy *Client Secret* and set them as environment variables.

3.2.1 Data description

Since Spotify is one of the most widely used audio streaming platofrms, its database has a wide range of songs and is open-source through their Web API. The API provides public access to user related data such as: playlists, tracks that the user saves and also provides metadata about music artists, albums, tracks directly from the *Spotify Data Catalogue*.

I managed to scrap a total of 180,220 unique (by song ID after removing duplicates) tracks after join the two datasets previously mentioned with all the necessary metadata related to the tracks. The features provided by Spotify Data Catalogue through Spotify Web API are mainly audio characteristics such as acousticness, danceability, valence, loudness, etc., but also information such as year in which the songs were released or the name of the artists. As far as the common (both audio and non-audio) features provided by Spotify are concerned, they are listed in more detail as follows :

- **Name** : Indicates the name of the track
- **Artists** : Indicates a list of artist names separated by commas
- **Duration** : Represents the duration of a track in milliseconds
- **Preview** : Link to 30 seconds preview of the track
- **Key** : Represents the estimated overall key of the track. Integers map to pitches using standard ,,Pitch Class notation''[PC22] (E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on).
- **Time signature** : A notational convention used in Western musical notation to specify how many beats(pulses) are contained in each measure(bar), and which note value is equivalent to a beat. [TS22]
- **Acousticness** : A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence that the track is acoustic.
- **Danceability** : Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rythm stability, beat strength and overall regularity. A value of 0.0 is the least danceable and 1.0 is the most danceable.
- **Energy** : Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of itensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For instance, metal and rock will have high energy.

- **Instrumentalness** : Predicts whether a track contains no vocals. „Ooh” and „aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly „vocal”.The closer the instrumentalness value is to 1.0, the greater likelihood the track content contains no vocal content.
- **Liveness** : Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
- **Loudness** : The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks.
- **Speechiness** : Detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
- **Valence** : Describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
- **Tempo** : The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece, and derives directly from the average beat duration.
- **Mode** : Indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is encoded as 1 and minor as 0.
- **Label** : Indicates the category of a playlist (e.g. toplists, pop, EDM, etc.).
- **Explicit** : Indicates whether a song contains vulgar, obscene words or not.
- **Release date** : Release date of a track in either DD MM YYYY or YYYY.
- **Year** : Release year of a track.

- **Type** : Spotify not only has audio content, but also video content such as music videos, podcasts, documentaries, etc. However, in our case since we only deal with audio content, the *audio_features* value is default for all rows.

3.2.2 Data visualizations

When dealing with data science or machine learning-related problems, it is good practice to know, understand and to get info on your data you are dealing with as much as possible in order to find potential correlations, which could lead the best possible solutions. That's why I used different methods for data visualizations such as line plots, bar plots, pie charts, etc.

Firstly, in **Figure 5** I extracted and managed to visualize the top 10 most popular tracks, after merging both datasets. Since in this merged artist, there is some sort of duplicates, i.e., there are multiple rows which can have an identical artist and song name, but different audio characteristics (possibly due to the environment in which some songs were recorded), IDs and popularity ratings. That's when computing the final popularity rating of a song played by the same artist, it is applied a mean on all occurrences of the same song played by the same artist.

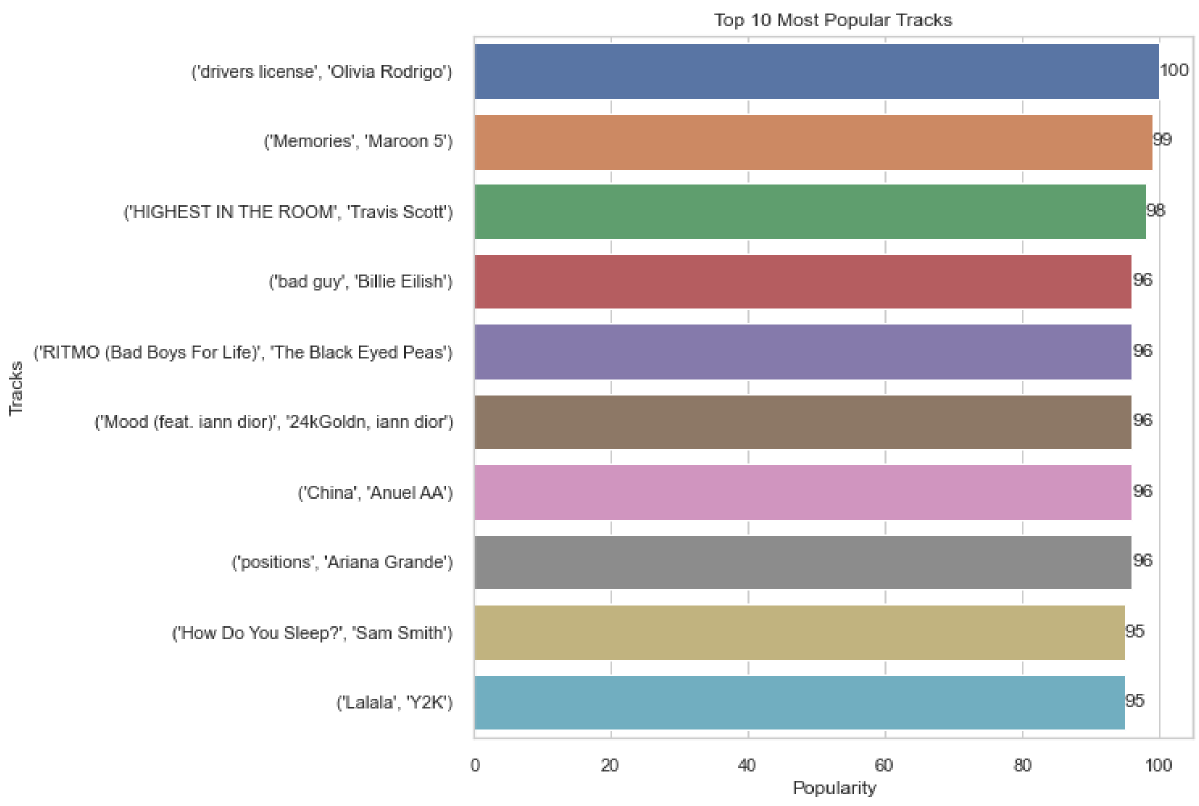


Figure 5. Bar plot of Top 10 Most Popular Tracks

In **Figure 6** I managed to extract and visualize the top 10 most popular artists, after merging both datasets. The popularity scale is computed by summing the popularity ratings of all songs of an artist or group of artists.

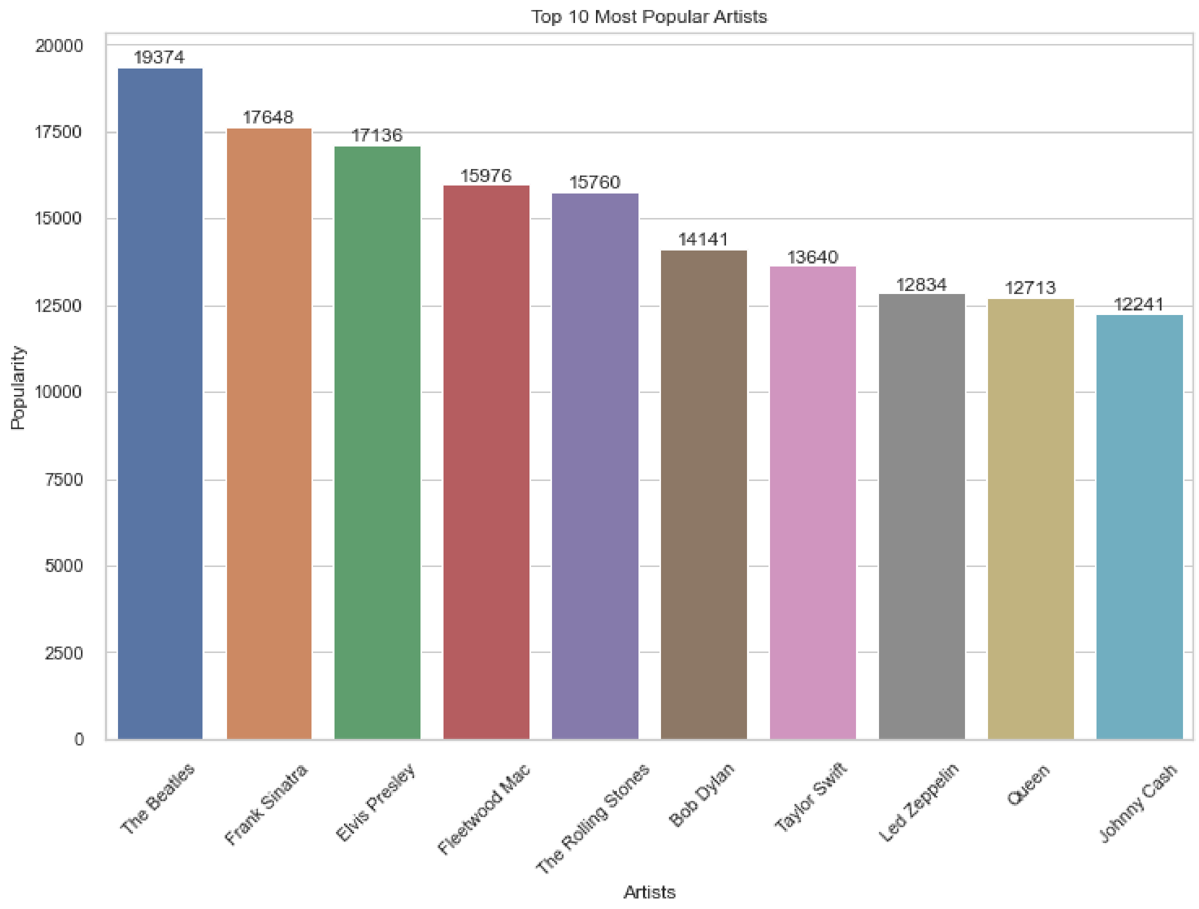


Figure 6. Bar plot of Top 10 Most Popular Artists

In **Figure 7** I managed to plot a line graph, which describes the number of songs of a specific year ranging from 1920 to 2021 only in the second dataset, since currently I have no specific data on the release year of each from the other dataset used.

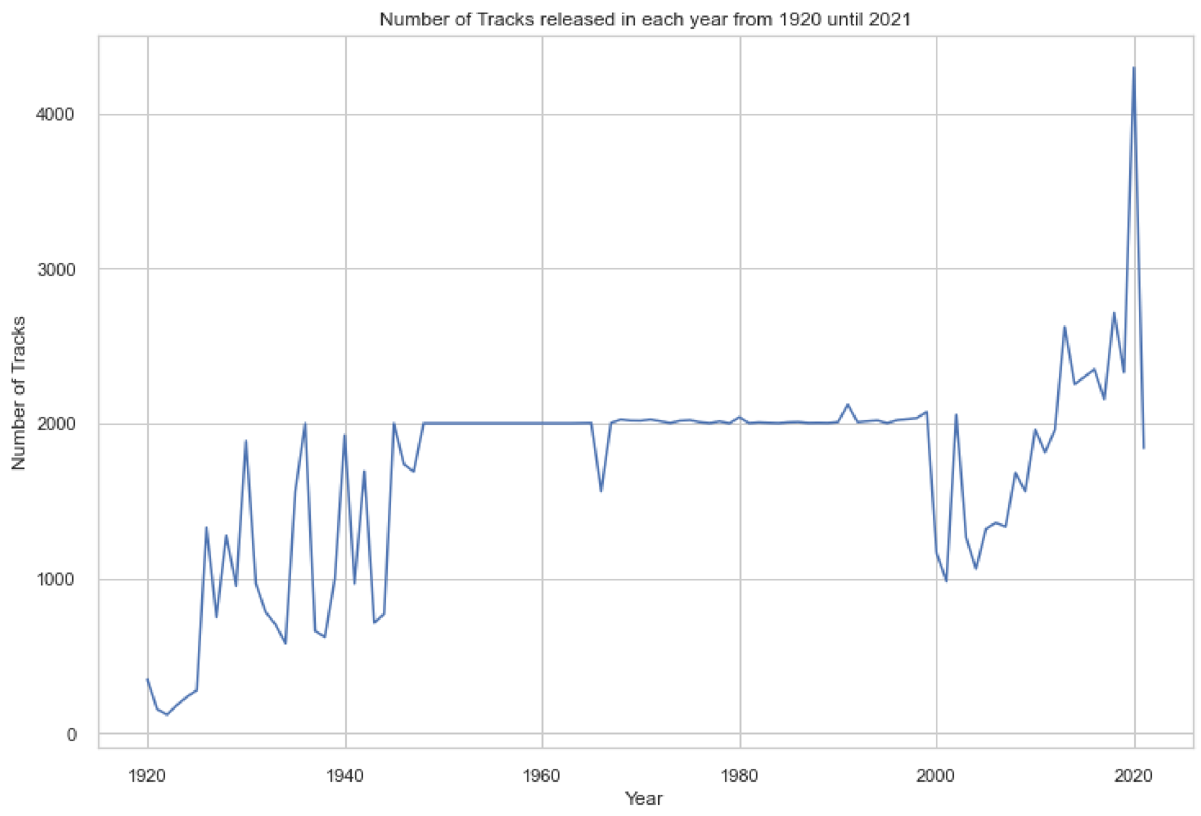


Figure 7. Number of Tracks released in each year from 1920 until 2021

In **Figure 8** I managed to plot a line graph, which describes the average evolution of all audio characteristics whose measurements range from 0.0 to 1.0 in each year from 1920 to 2021 from one of the datasets that has data on release year. So, the only audio characteristic plotted on this graph are *acousticness*, *danceability*, *energy*, *speechiness*, *liveness*, *valence* and *instrumentalness*. It can be summarized that:

- There is a huge decrease of *acousticness* in songs after the *60s*. This means that since then until nowadays songs are **far less acoustic**.
- After the *60s* there are, to some extent, slight increases in the *danceability* of songs. In other words, songs are getting **more and more suitable for dancing**.
- After somewhere between *40s* and *60s* (maybe, to be more precise, between *40s* and *50s*) there are overall **slight** and **more major, significant increases** in the energy of songs.
- As far as *speechiness* is concerned, all songs **either contain both music and some sort of speec**h (such as one typically used in rap) or **only contain music and no sort of speech**.
- As for possible live performed songs, since the *liveness* value ranges from slightly below 0.2 to slightly above 0.2 for all songs, there are **no songs performed live**.

- When it comes to musical positiveness by tracks, it can be concluded that by *valence* songs are probably **mostly little negative with few exceptions of little positive songs**.
- Lastly, when it comes to *instrumentalness*, apart from the *20s* to *40s* range, where there are **significant values which indicate multiple songs of significant less vocal content**, but overall **most songs contain vocal content**.

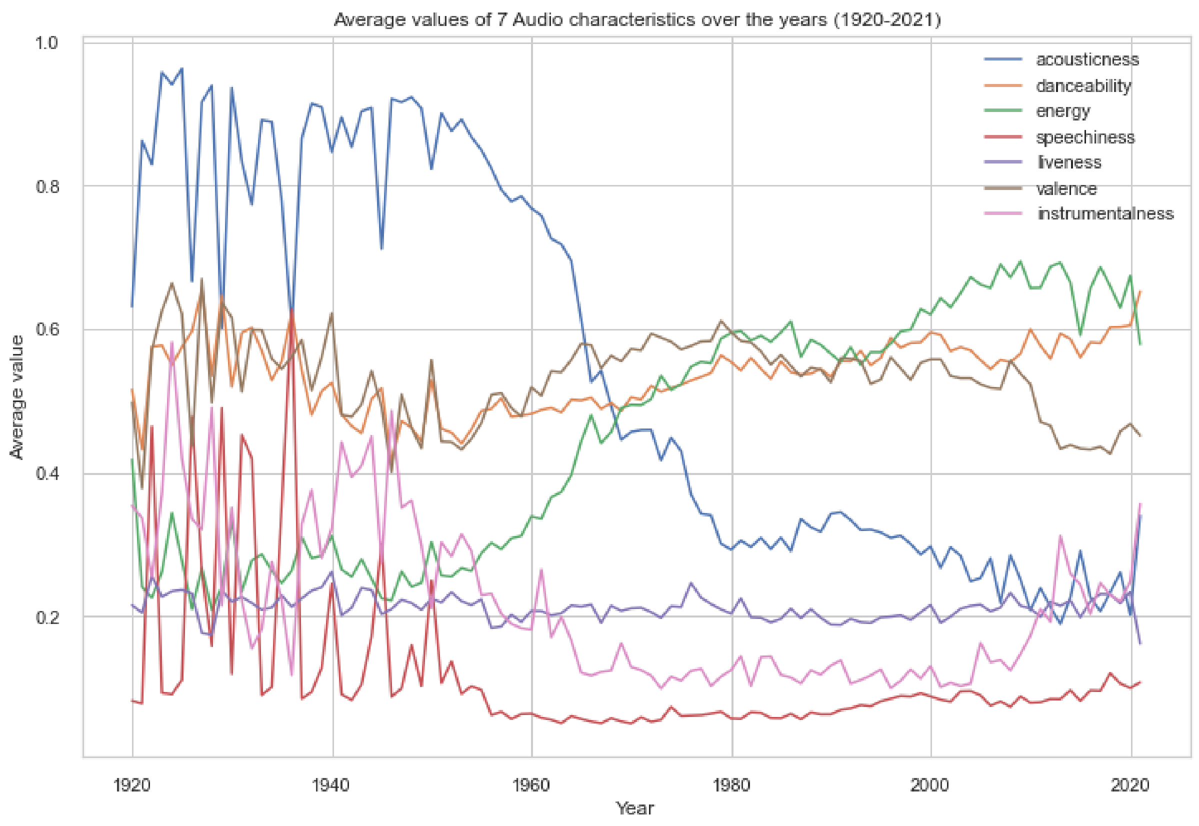


Figure 8. Average values of 7 audio characteristics in each year from 1920 until 2021

In **Figure 9** I plotted a graph which describes the average tempo of all songs in each year from 1920 until 2021. Considering only one of the datasets, the tempo values range from 0 to approximately 243.5 BPM (Beats Per Minute). Mainly from somewhere between 40s to 50s, songs have greater and greater tempo values.

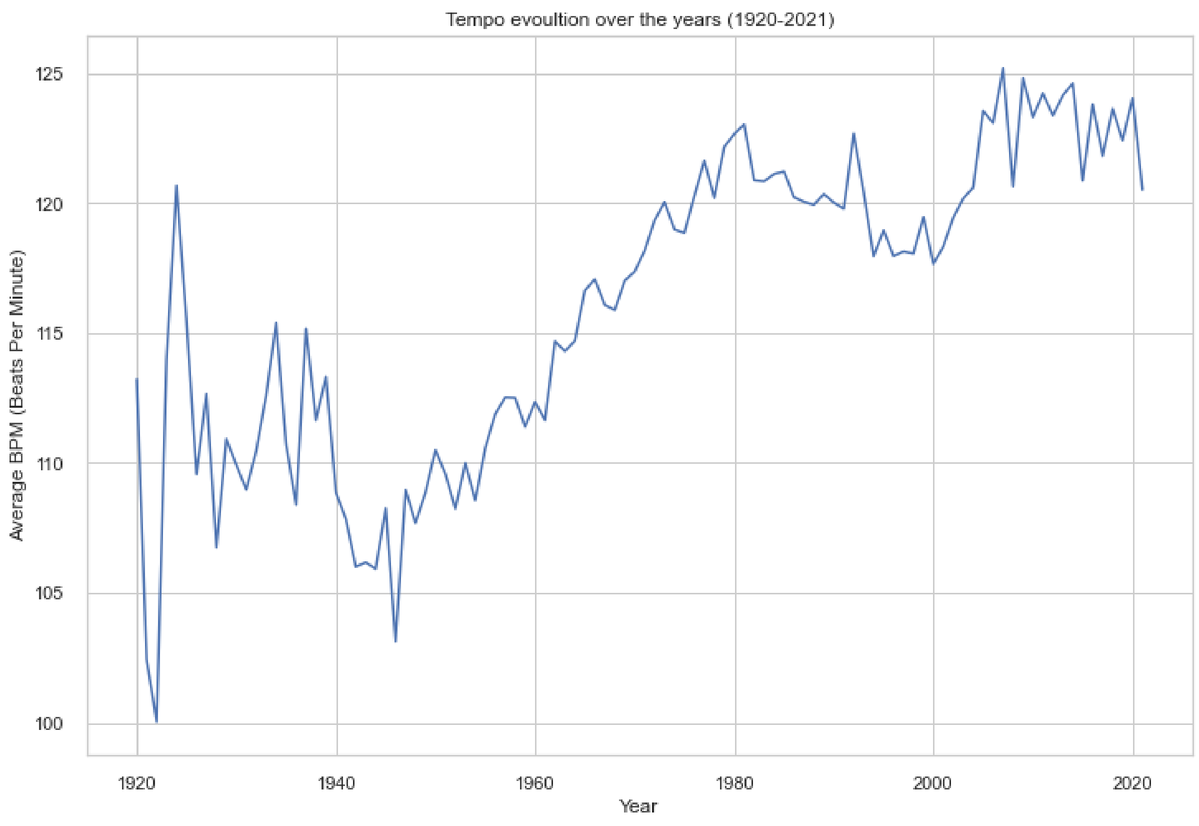


Figure 9. Average tempo of all songs in each year from 1920 until 2021

In **Figure 10** I plotted a graph which describes the average sound measurement of loudness in all songs in each year from 1920 until 2021. Considering only one of the datasets, the values range from **-**60 to 3.855 db. From this graph, it can be stated that mainly songs after *60s* are **more or less constantly getting louder**.

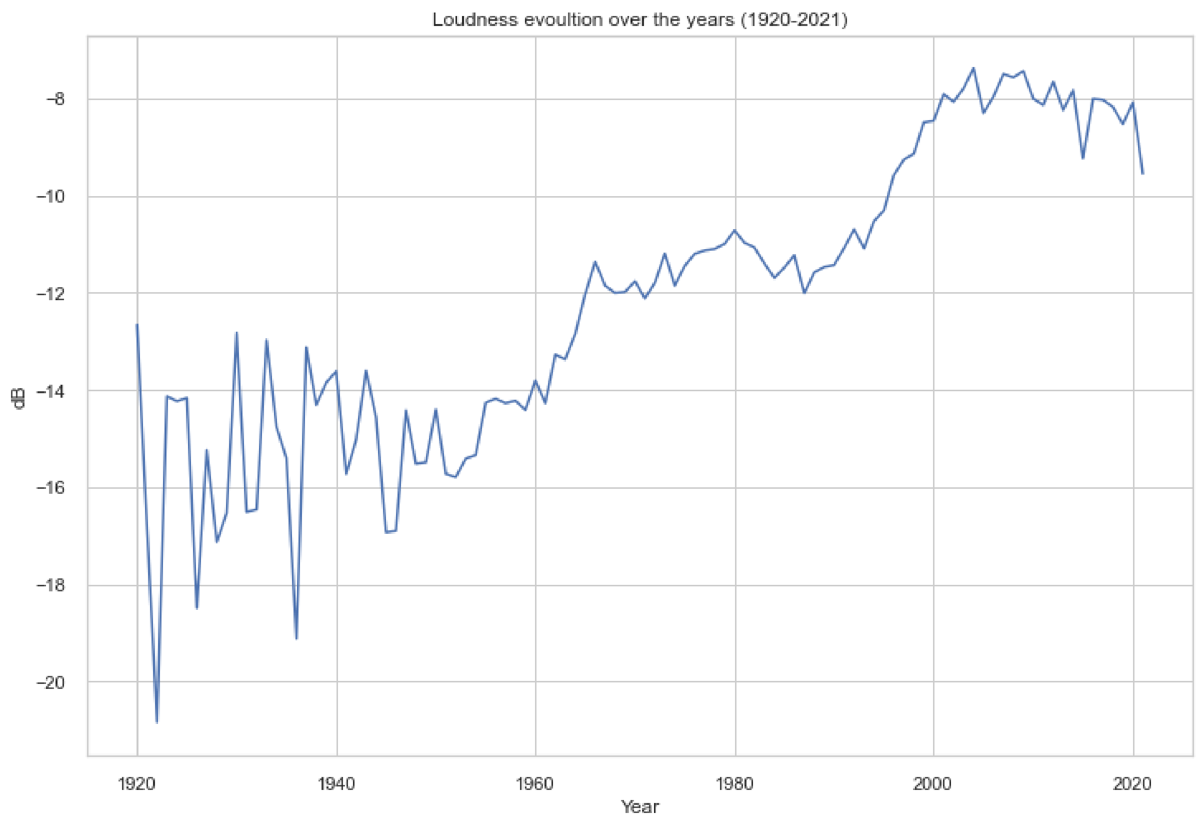


Figure 10. Average sound measurement of loudness (in dB) of all songs in each year from 1920 until 2021

In **Figure 11** I plotted a graph which describes the average evolution of duration (in milliseconds) in all songs in each year from 1920 until 2021. Considering only one of the datasets, the milliseconds range from 4937 to 5338302. From this plot it can be deduced that all songs range, on average, from approximately 2,33 minutes to approximately 4,66 minutes.

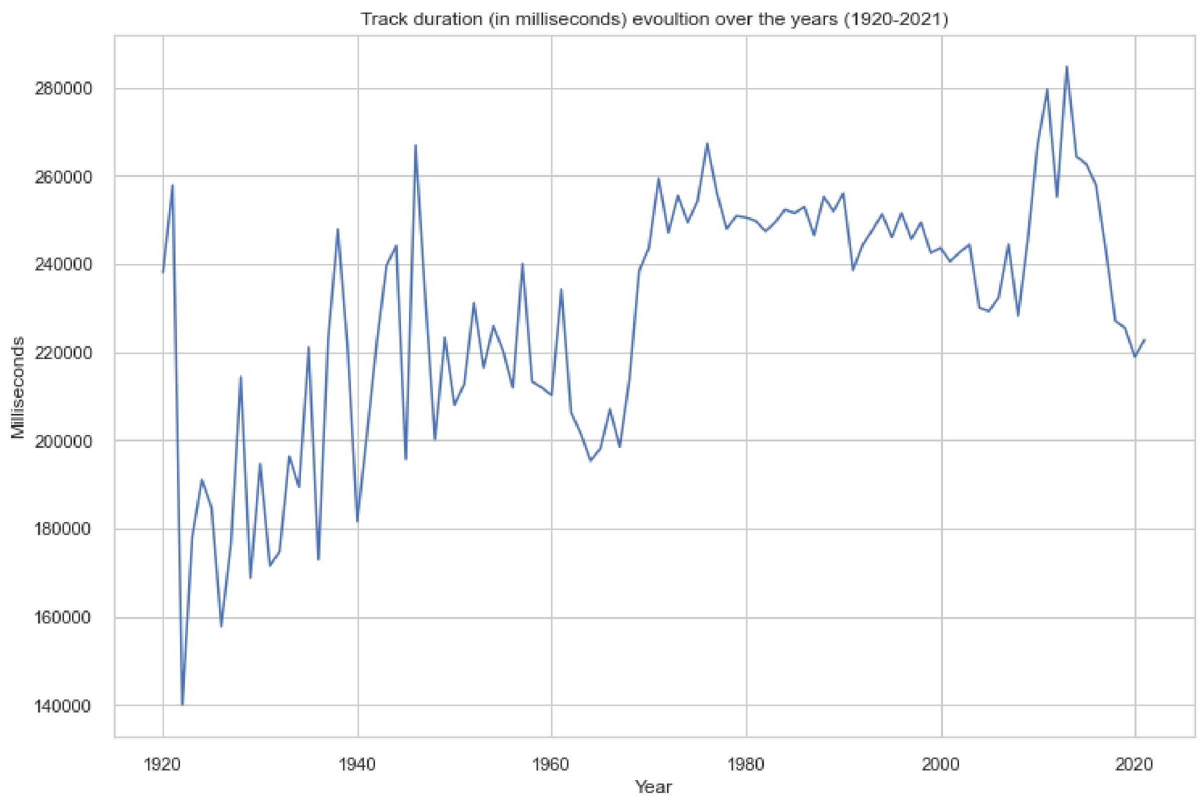


Figure 11. Average duration (in ms) in all songs in each year from 1920 until 2021

In **Figure 12** I made a horizontal bar plot which describes the top 10 artists with the most songs in the final dataset resulted from merging the two datasets used for the music recommender system.

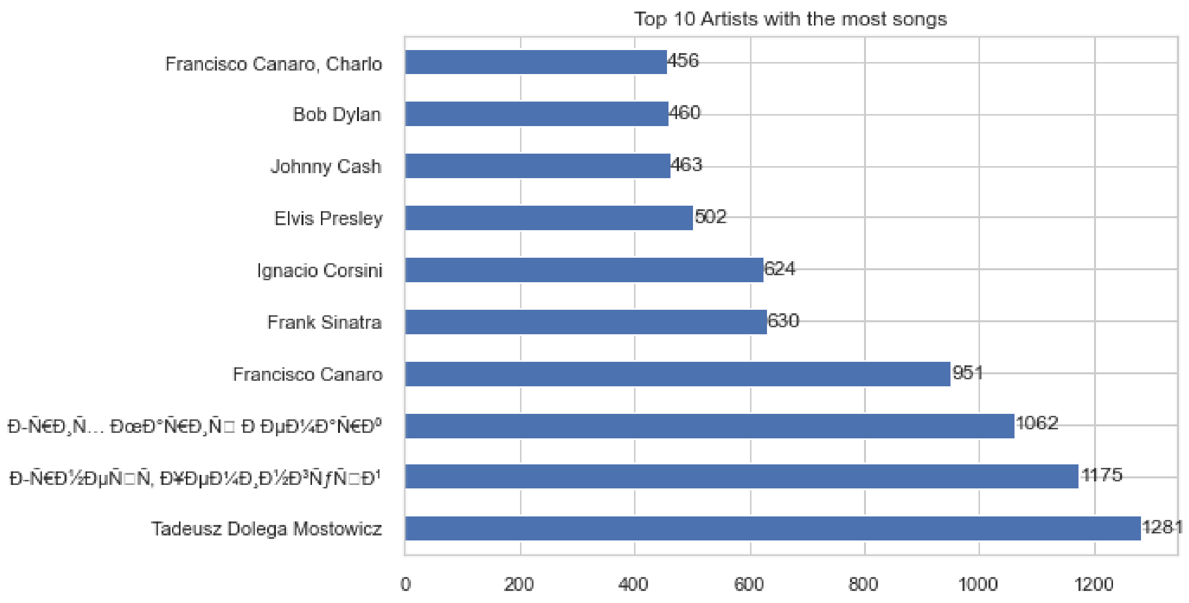


Figure 12. Horizontal bar plot of Top 10 Artists with most songs

In **Figure 13** it is plotted a pie-chart which describes the percentages of songs in major and minor mode of both datasets merged.

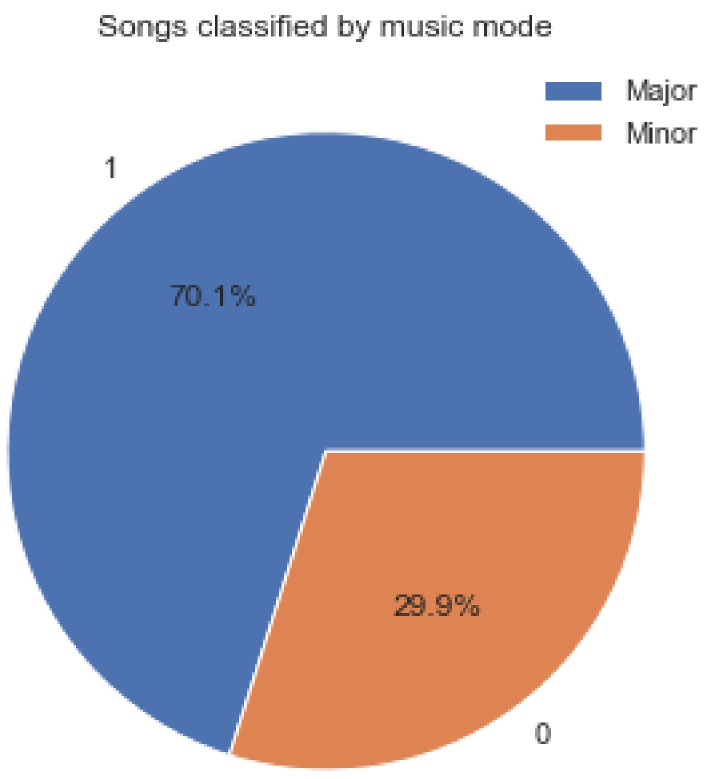


Figure 13. Songs classified by music mode

In **Figure 14** it is visualized a pie-chart which describes the percentages of songs with explicit content (offensive language, different kinds of slurs, etc.) of only one of the datasets used.

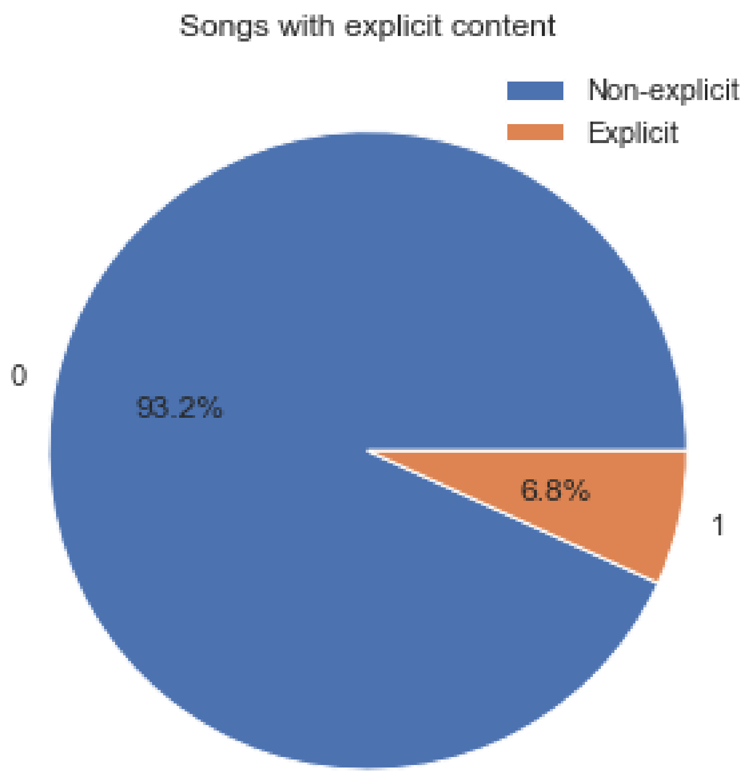


Figure 14. Songs with explicit content

In **Figure 15** the pie-chart describes the type of playlists the songs from one of the datasets used belongs to.

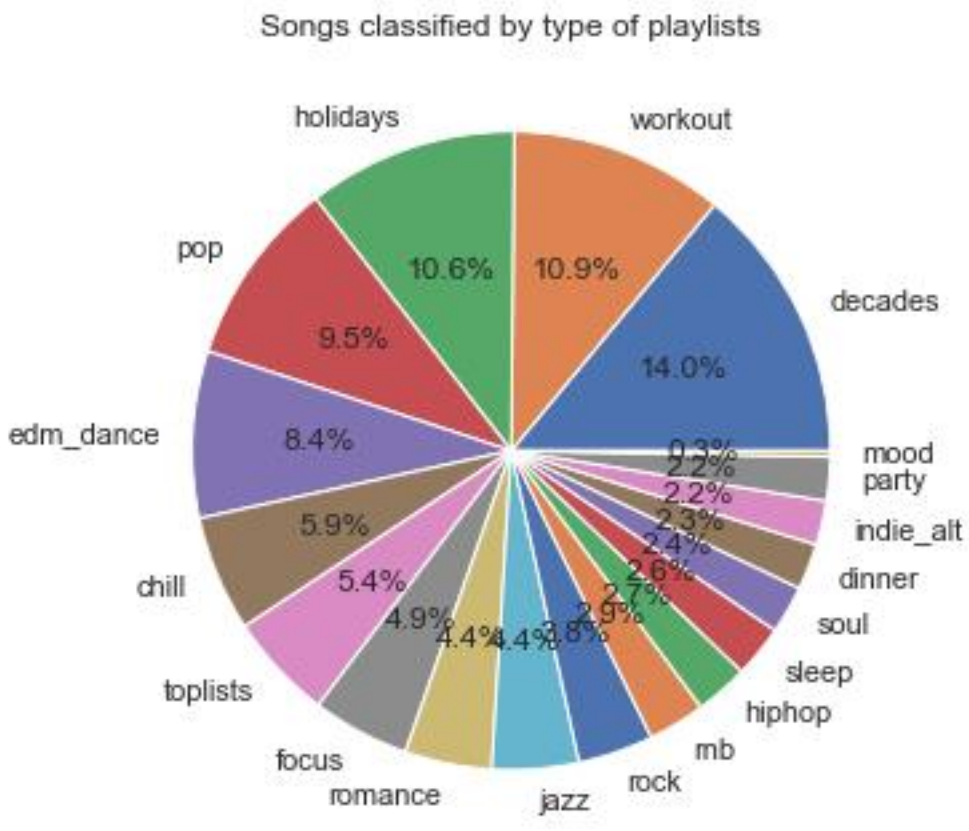


Figure 15. Songs classified by type of playlists

In **Figure 16** there are described the percentages of songs according to their keys. “A key is a group of pitches or scale that forms the basis of any musical composition.“ [KM06]

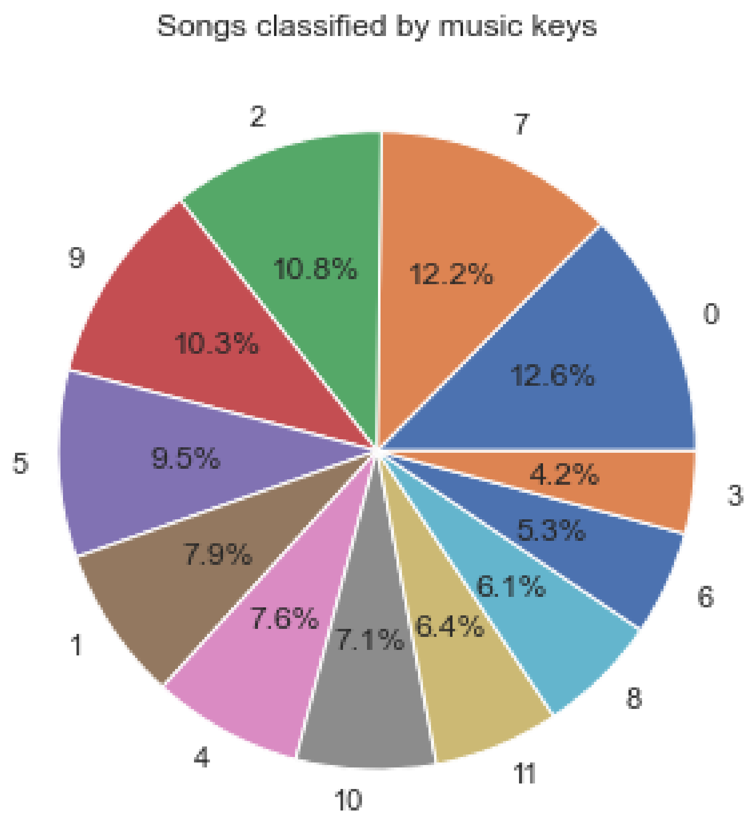


Figure 16. Songs classified by music keys

At last, but not least, in **Figure 17** it is displayed a pie-chart which classifies songs by their time signature.

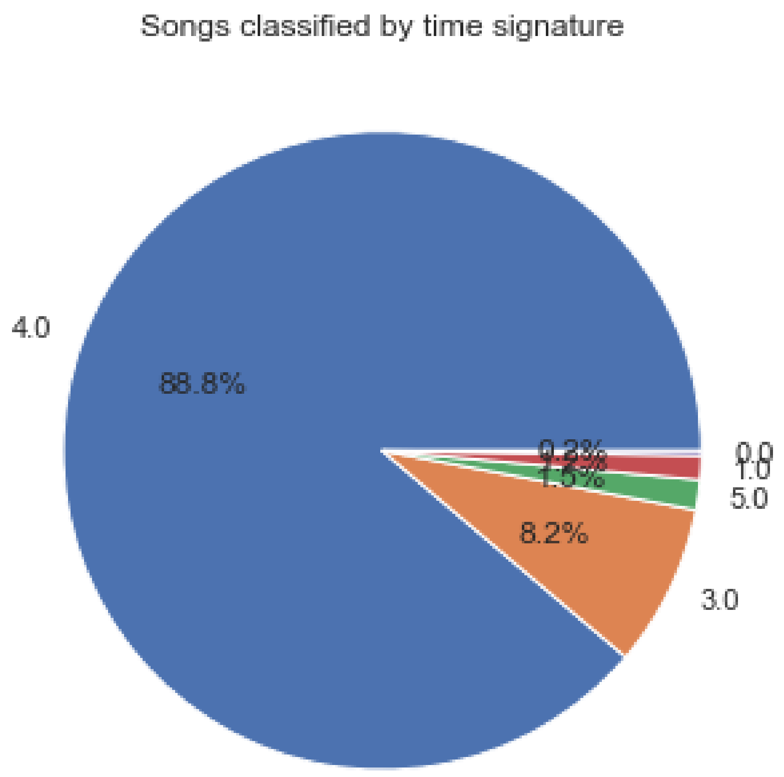


Figure 17. Songs classified by time signature

3.3 Dataset preparation

3.3.1 Data loading

As aforementioned in the **3.2.1 Data description** section, there are two datasets used in this particular music recommender system. Initially, both datasets have the following common fields that contain metadata about our music songs: *id*, *artists*, *name*, *popularity*, *danceability*, *energy*, *key*, *loudness*, *mode*, *speechiness*, *acousticness*, *instrumentalness*, *liveness*, *valence*, *tempo* and *type*.

One initial dataset in **.csv** extension format is final.csv, which was downloaded from the public-available aforementioned GitHub repository [RD20], has, apart from the aforementioned common fields, *preview*, *time signature* and *label* fields. This adds up to 19 data fields and the dataset has 10,449 entries.

The other dataset has the same extension and was initially called data.csv [XZ21], which was downloaded from Jovian.io, and has, apart from the aforementioned common fields, *explicit*, *release date* and *year* fields. This adds up to 19 data fields as well and the dataset has 174,389 entries.

But before the inclusion of the datasets in the music recommender system, there were done some additional preprocessing steps done for dataset preparation, which will be described below in the following subsections.

3.3.2 Data filtering and cleaning

Firstly, I noticed when scrolling on both datasets that there were data entries in which there were a few to a lot of "garbage characters". The only conclusion I could reach is the fact that since the data collected by both people was saved in a .csv format, it means that certain characters were "corrupted" to the point where they weren't readable due to the fact that files in .csv format don't support certain character encodings for characters in languages that don't belong to English language. This was confirmed when trying to rename certain song titles and artist names, the moment I saved the file after editing, when re-opening the file, the editing on both song titles and artist names disappeared, meaning that .csv simply couldn't support their character encodings. This led me to the decision to try to convert the files into **.xlsx** formats before using them in my recommender system, because I noticed files with Excel extensions support the necessary character encoding for all possible characters in all languages. This way, after converting the files in .xlsx formats, I tried my best to modify artist names and songs titles in order to increase the chances for my lyrics retrieval API to find the lyrics corresponding to a certain song. Currently, there is still content with "garbage" characters due to the fact that I simply couldn't find certain original artist names and song titles after trying to find them by making use of their preview links on Shazam app or simply introducing them on Google and DuckDuckGo search engines. Moreover, since there are a lot of songs, unfortunately I simply couldn't have all the time to refactor all the possible artist names and song titles.

Another important aspect to mention is the fact that before re-naming artist names and song titles with corrupted characters, I had to do some further data cleaning by removing [,] and certain " characters, because the content on artist names and song titles seemed to be initially parsed in a JSON format, by simply making use of **find and replace** functionality from Excel.

Only afterwards, I wrote a short Python script for checking if both datasets have any duplicate values and found out that only one of them (the one with most data, i.e., approximately 174,000 data entries) had 2159 song duplicates, by considering the id column for identifying duplicates.

3.3.3 Data merging

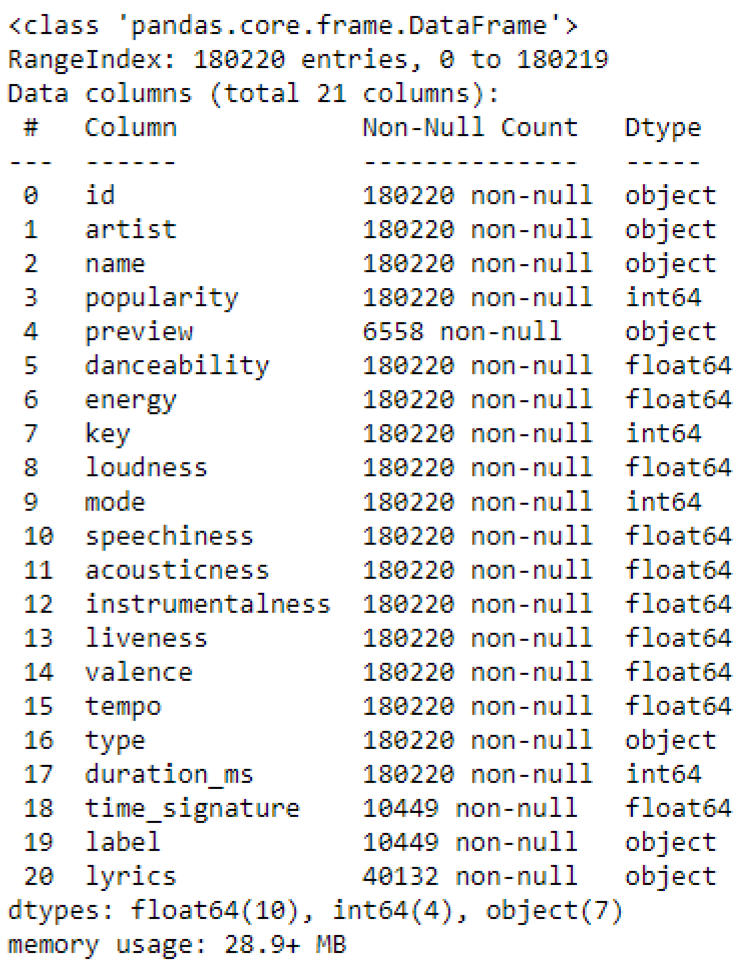
Afterwards, when merging both datasets, we check again for any possible duplicates, if there are, we simply compute the mean of all non-categorical song features (*acousticness*, *danceability*, etc.) and store them in the final dataset. When the merging is done, the final dataset will have a total of 180,220 entries.

3.3.4 Lyrics collection

After the merging was done, I thought about collecting for a *lyrics* field some lyrics data, which could potentially be proven to be pretty useful for the recommender system. The lyrics were collected using **lyrics.ovh**, a simple API where the name of the artist and the title of the song need to be provided in an URL in order to get a potential JSON response having a lyrics object. The URL has the following format: https://api.lyrics.ovh/v1/artist name/song title. One major problem that I encountered when gathering lyrics was the fact there were a lot of queries to be made in order to scan the whole final dataset, which consequently resulted in very long running time periods until the final dataset is scanned as a whole. Therefore, I ended up writing a multi-processing algorithm, which was meant to make multiple queries to the lyrics retrieval API in order to reduce run-time, using all my logical cores, which number 12. Nevertheless, if it had been to scan the whole final dataset, it would have taken at least one or even two days for the whole routine to be finished running. This is why, instead of going through the whole final dataset, I decided to scan alternately and separately chunks of 10,000 songs, by ending up having 18 different versions of the same dataset exported into Excel, which were then merged into a single final version of the dataset. So far, I managed to retrieve approximately 40,132 song lyrics, but could be potentially be little less, since **lyrics.ovh** can parse content with only tags such as [**Instrumental]**, which means that certain songs aren't supposed to have any lyrics.

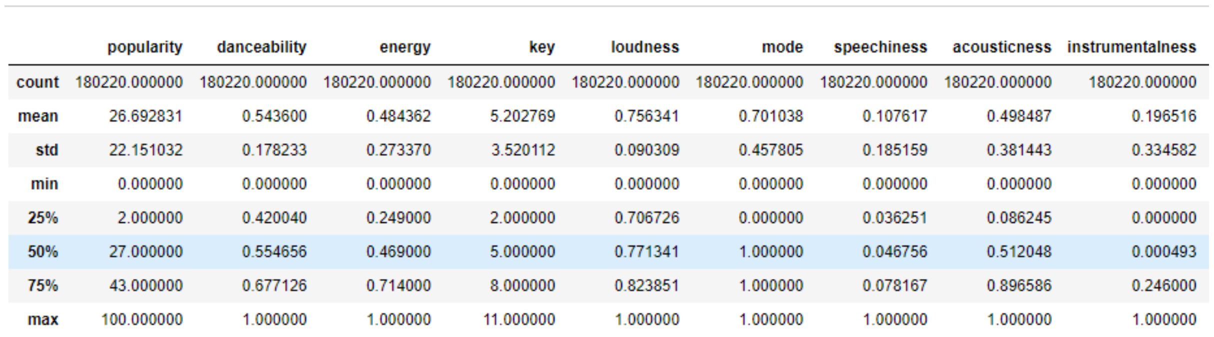
3.4 Dataset used

The final dataset has the following descriptions in both **Table 1** and **Table 2**. In the former figure the data columns are described by providing the total numbers of columns, their data type, their column labels, memory usage, range index and their non-null count and the total number of columns of certain data types. In the latter figure there are summarized statistical values (mean, standard deviation, minimum, etc.) about the data columns with numerical values.

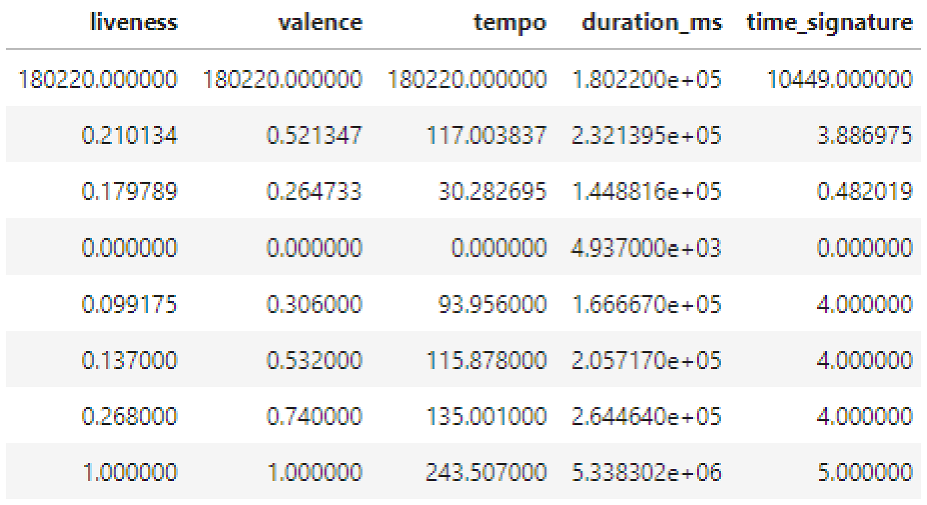


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180220 entries, 0 to 180219
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   id                180220 non-null  object 
 1   artist            180220 non-null  object 
 2   name              180220 non-null  object 
 3   popularity        180220 non-null  int64  
 4   preview           6558 non-null    object 
 5   danceability      180220 non-null  float64
 6   energy            180220 non-null  float64
 7   key               180220 non-null  int64  
 8   loudness          180220 non-null  float64
 9   mode              180220 non-null  int64  
 10  speechiness       180220 non-null  float64
 11  acousticness      180220 non-null  float64
 12  instrumentalness  180220 non-null  float64
 13  liveness          180220 non-null  float64
 14  valence           180220 non-null  float64
 15  tempo             180220 non-null  float64
 16  type              180220 non-null  object 
 17  duration_ms       180220 non-null  int64  
 18  time_signature    10449 non-null   float64
 19  label             10449 non-null   object 
 20  lyrics            40132 non-null   object 
dtypes: float64(10), int64(4), object(7)
memory usage: 28.9+ MB
```

Table 1. Illustration of the information table provided by info method of Data Frame object



	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness
count	180220.000000	180220.000000	180220.000000	180220.000000	180220.000000	180220.000000	180220.000000	180220.000000	180220.000000
mean	26.692831	0.543600	0.484362	5.202769	0.756341	0.701038	0.107617	0.498487	0.196516
std	22.151032	0.178233	0.273370	3.520112	0.090309	0.457805	0.185159	0.381443	0.334582
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.420040	0.249000	2.000000	0.706726	0.000000	0.036251	0.086245	0.000000
50%	27.000000	0.554656	0.469000	5.000000	0.771341	1.000000	0.046756	0.512048	0.000493
75%	43.000000	0.677126	0.714000	8.000000	0.823851	1.000000	0.078167	0.896586	0.246000
max	100.000000	1.000000	1.000000	11.000000	1.000000	1.000000	1.000000	1.000000	1.000000



liveness	valence	tempo	duration_ms	time_signature
180220.000000	180220.000000	180220.000000	1.802200e+05	10449.000000
0.210134	0.521347	117.003837	2.321395e+05	3.886975
0.179789	0.264733	30.282695	1.448816e+05	0.482019
0.000000	0.000000	0.000000	4.937000e+03	0.000000
0.099175	0.306000	93.956000	1.666670e+05	4.000000
0.137000	0.532000	115.878000	2.057170e+05	4.000000
0.268000	0.740000	135.001000	2.644640e+05	4.000000
1.000000	1.000000	243.507000	5.338302e+06	5.000000

Table 2. Illustration of statistical data from data columns with numerical values

3.5 Preprocessing algorithm

Prior to the recommendation algorithm phase, the dataset needs to undergo some preprocessing steps, not only for the recommendation algorithm to work, but to also improve its performance as best as possible. The preprocessing steps needed to be performed are the following: **Data normalization**, **Dimensionality reduction** and **Clustering**. All these procedures in depth are described in the following subsections of this chapter.

3.5.1 Data normalization

To start with, it is undergone a selection of features, where each column (which in this context can be also referred to as features) that has certain useful numerical values will be classified as non-categorical and the rest of the remaining columns will be categorized as categorical data, which some of it will be used mainly for outputting purposes (e.g. artists, song title, etc.), but except that, they won't be in any way further down in the whole recommendation process.

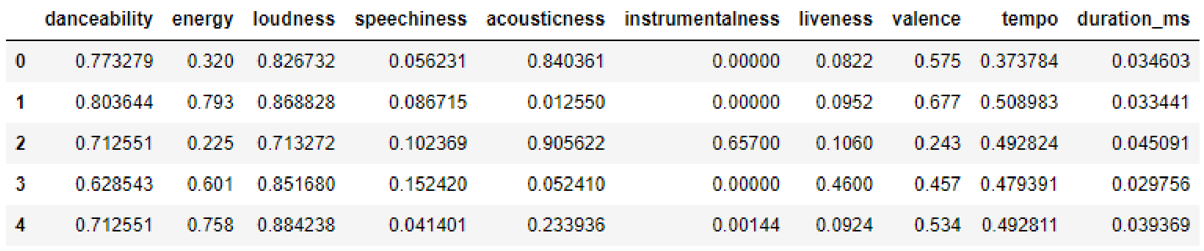
Once this step was finished, it was concluded that the non-categorical features that will be further used are the following: *danceability*, *energy*, *loudness*, *speechiness*, *acousticness*, *instrumentalness*, *liveness*, *valence*, *tempo* and *duration in milliseconds*. Afterwards, **Min-Max scaling** will be applied to the selected features. This type of rescaling is the simplest approach and consists in rescaling the range of features to scale the range usually either in [0, 1] or [-1, 1] range. Selecting the target range depends heavily on the nature of data. The general formula is used for a min-max of [0, 1] and is given as: $' = \frac{\mathrm{x} - \min(\mathrm{x})}{\max(\mathrm{x}) - \min(\mathrm{x})}$, where x is an original value and x' is the normalized value.

In case it is needed to fit the data into a custom range [a, b], the formula becomes:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

There are other normalization approaches such as **mean normalization**, where instead of subtracting the minimum value of x in the general formula, the average value of x is subtracted, and **standardization** (also known as **Z-score normalization**) which is similar to mean normalization, but instead of dividing the difference between any value x and the average value of x to difference between the maximum value of x and minimum value of x, it is divided to the standard deviation.

In our case, the type of Min-Max scaling that will be applied is the one from the general formula which fits all the values of the selected features in [0, 1] range. The first 5 values (indexed from 0 to 4) can be shown in the **Table 3** below to highlight them after Min-Max scaling was applied on them.



	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
0	0.773279	0.320	0.826732	0.056231	0.840361	0.00000	0.0822	0.575	0.373784	0.034603
1	0.803644	0.793	0.868828	0.086715	0.012550	0.00000	0.0952	0.677	0.508983	0.033441
2	0.712551	0.225	0.713272	0.102369	0.905622	0.65700	0.1060	0.243	0.492824	0.045091
3	0.628543	0.601	0.851680	0.152420	0.052410	0.00000	0.4600	0.457	0.479391	0.029756
4	0.712551	0.758	0.884238	0.041401	0.233936	0.00144	0.0924	0.534	0.492811	0.039369

Table 3. First 5 re-scaled values of all non-categorical features used for the recommender system

3.5.2 Dimensionality reduction

After the non-categorical data was re-scaled, because of its large dimensionality (numbering 10 total dimensions) at the current point the current data frame needs to be reduced in a way such that it can still keep most of its former data. One of the most used techniques for dimensionality reduction is **Principal Component Analysis** (**PCA**) [ZJ21], which is the technique that I applied on my data frame.

Generally speaking, reducing the number of variables of a dataset consequently comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. By simplicity, it means that smaller data frames are easier to explore and visualize, which consequently makes analyzing data much easier and faster for machine learning algorithms without nonessential variables to process.

Before jumping to reducing data dimensionality, I computed the correlation matrix of all non-categorical features to check if there are features with high correlations and, therefore, potentially need to be considered for removal.

As far as the correlation metrics used for the computation of such matrices, there are usually 3 default built-in methods in Data Science: Pearson coefficient (also known as standard correlation coefficient) [AS21], Kendall Tau correlation coefficient and Spearman rank correlation. But in this context, only Pearson coefficient will be described, since this is the method applied for computing the correlation matrix.

Pearson correlation can measure the linear relationship between two variables (or features in this context), but fails to capture the non-linear relationship of the two variables. It assumes that both variables are normally distributed and can be used for nominal variables or continuous variables. In our context for the recommender system, since Min-Max scaling was previously applied, it can safely be stated that the data frame is normally distributed.

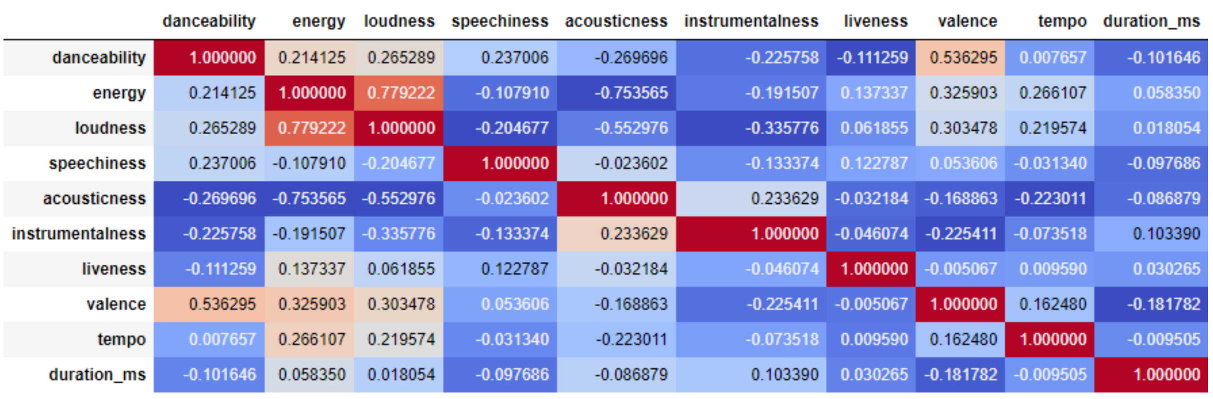
Pearson correlation coefficient between two variables X and Y can be computed using the following formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where, r is the Pearson Correlation Coefficient and ranges from -1 to 1, x_i is the current value from X variable, y_i is the current value from Y variable, $\bar{x}$ is the mean of values in X variable and $\bar{y}$ is the mean of values in Y variable.

As far as limitations of Pearson correlation are concerned, apart from the fact that it can’t capture the non-linear relationship of any two variables, a key limitation is that it cannot distinguish between independent and dependent variables. Thus, even if a relationship between two variables is found, it does not indicate which variable was ‘the cause’ and which was ‘the effect’. Another limitation is that it doesn’t provide information on the slope of the line and only indicates the existence and type of relationship. The slope must be found by creating a scatter plot. And when considering using Pearson correlation coefficient, it is important to make sure that the values in features are not ordinal (where sequence matters).

Coming back to our data frame of selected features, after applying the Pearson metrics for computing the correlation matrix, the following results can be displayed in **Table 4** below:



	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
danceability	1.000000	0.214125	0.265289	0.237006	-0.269696	-0.225758	-0.111259	0.536295	0.007657	-0.101646
energy	0.214125	1.000000	0.779222	-0.107910	-0.753565	-0.191507	0.137337	0.325903	0.266107	0.058350
loudness	0.265289	0.779222	1.000000	-0.204677	-0.552976	-0.335776	0.061855	0.303478	0.219574	0.018054
speechiness	0.237006	-0.107910	-0.204677	1.000000	-0.023602	-0.133374	0.122787	0.053606	-0.031340	-0.097686
acousticness	-0.269696	-0.753565	-0.552976	-0.023602	1.000000	0.233629	-0.032184	-0.168863	-0.223011	-0.086879
instrumentalness	-0.225758	-0.191507	-0.335776	-0.133374	0.233629	1.000000	-0.046074	-0.225411	-0.073518	0.103390
liveness	-0.111259	0.137337	0.061855	0.122787	-0.032184	-0.046074	1.000000	-0.005067	0.009590	0.030265
valence	0.536295	0.325903	0.303478	0.053606	-0.168863	-0.225411	-0.005067	1.000000	0.162480	-0.181782
tempo	0.007657	0.266107	0.219574	-0.031340	-0.223011	-0.073518	0.009590	0.162480	1.000000	-0.009505
duration_ms	-0.101646	0.058350	0.018054	-0.097686	-0.086879	0.103390	0.030265	-0.181782	-0.009505	1.000000

Table 4. Results of correlation matrix of all selected features

When analyzing the results of the correlation matrix, it is first of all important how to interpret them and there are 5 cases by which it can be drawn a conclusion on the relationship between two features as well as the magnitude defined by the strength of the variables:

- **Perfectly positive correlation**: When the correlation values is exactly 1.
- **Positive correlation**: When correlation values falls between 0 to 1.
- **No correlation**: When correlation value is 0.
- **Negative correlation**: When correlation value falls between -1 to 0.
- **Perfectly negative correlation**: When correlation value is exactly -1.

The linear relationship between any two variables can be illustrated graphically in the following figure below:

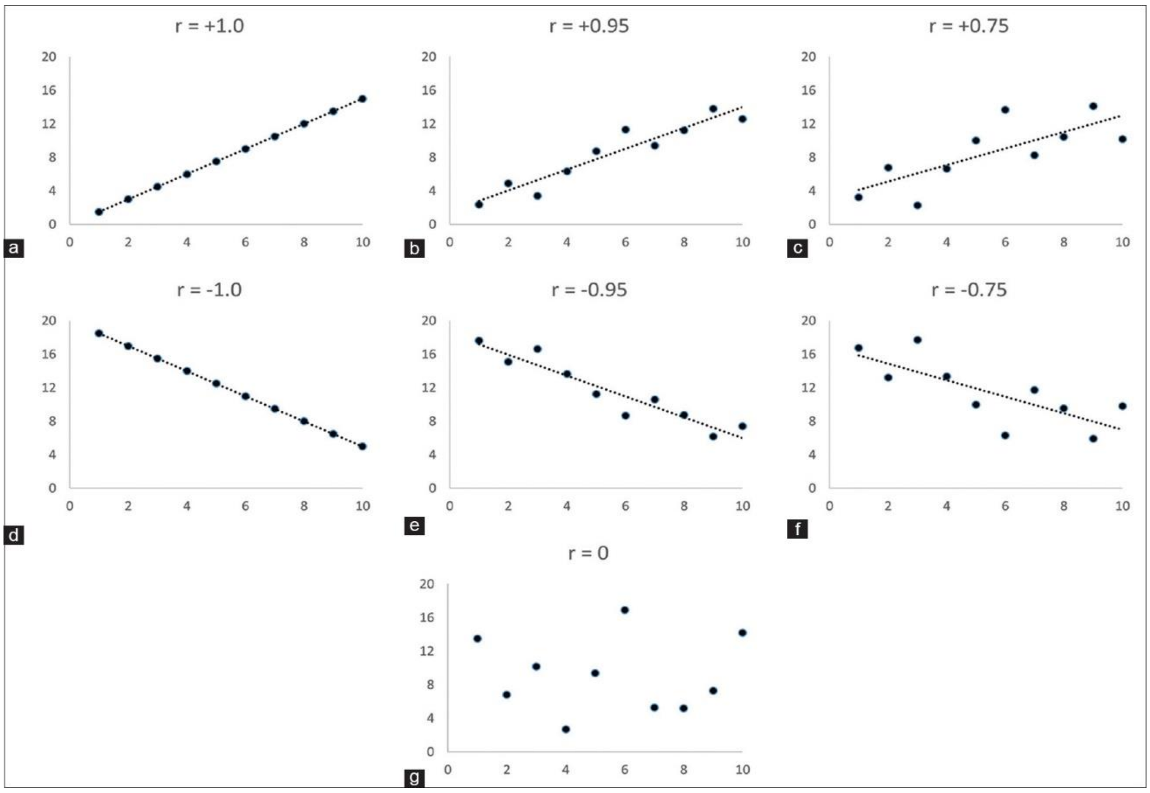


Figure 18. Illustration of all 5 linear relationship cases through 7 different plots [AS21]

As far as our data is concerned, it can be concluded that a few high correlations between the selected features are to be considered. Starting from the highest correlation, we have positive correlation between *loudness* and *energy*, *danceability* and *valence*, *energy* and *valence*, *loudness* and *valence*, *tempo* and *energy*, and there are a few more positive correlations, but the following are getting much lower to the point where they aren't worth taking into consideration. As far as the lowest correlations are concerned, the two most noticeable are the one between *energy* and *acousticness,* followed by *loudness* and *acousticness*. Considering, the results obtained after computing the correlation matrix, it can with certainty be concluded that the data dimensionality needs to be reduced.

Coming back to Principal Component Analysis, the way this technique works is by the following steps:

1) **Standardization**

As illustrated below, this is done by subtracting each value from each feature of the data frame by the average value of the corresponding feature and then dividing the result of subtraction by the standard deviation value of the corresponding feature.

$$x' = \frac{x - \bar{x}}{\sigma}$$

$$\sigma = \frac{\sum(x-\mu)^2}{N}$$

2) **Covariance Matrix Computation**

The covariance matrix is a dxd symmetric matrix (where d is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables.

Since the covariance of a variable with itself is its variance (Cov(a, a) = Var(a)), in the main diagonal (top left to bottom right) the variances of each initial variable are situated. Apart from that, since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

3) **Computation of eigenvalues and eigenvectors of the covariance matrix**

Eigenvectors and eigenvalues are linear algebra concepts that need to be computed from the covariance matrix in order to determine the **principal components** of our selected features.

Principal components are, simply put, new features that are constructed as linear combinations of the initial features. These combinations are done in a way such that the new features (i.e. principal components) are uncorrelated and most of the information within the initial features is compressed into the first components. To put this into perspective of the selected features, the initial number of features give us the same number of principal components (10 to be more precise), but PCA attempts to put the most information possible in the first component, then the maximum remaining information and so on, until it is obtained something similar to **Figure 19**. In this figure, it is revealed the percentage of each explained variance ratio for every principal component numbered from 1 to 10.

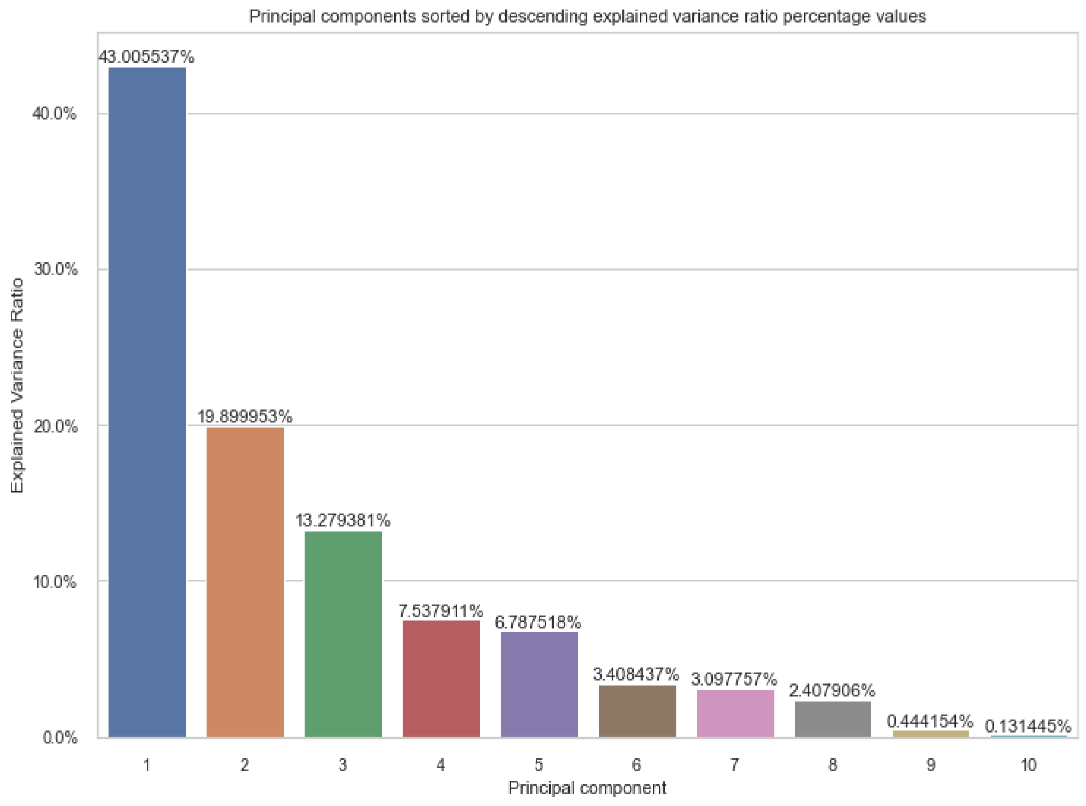


Figure 19. Principal components sorted by descending explained variance ratio percentage values

Organizing information in principal components this way will allow our data frame to have its dimensionality reduced without losing much information, and this is possible by discarding the components with low information and considering the remaining components as new unnamed features.

An important thing to realize here is that, the principal components are less interpretable and don't have any real meanings since they are constructed as linear combinations of the initial features.

Geometrically speaking, principal components represent the directions of the data that **explain a maximal amount of variance**, in other words, the lines that capture most information of the data. The relationship between variance and information in this context is that the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion is along a line, the more information it has. Another way to comprehend this is to think of the principal components as new axes that provide the best angle to see and evaluate data, so that the differences between the observations are better visible.

4) Sort eigenvalues and their correspondent eigenvectors

After solving both $\det(A - \lambda I) = 0$ and $(A - \lambda I)v = 0$ equations for determining the eigenvalues, respectively eigenvectors, where A is the covariance matrix of features, λ are the eigenvalues, I the identity matrix and v the matrix of eigenvectors resulted from eigenvalues, the eigen values are sorted from highest to lowest values and the eigenvectors are sorted according to their eigenvalues.

5) Pick the first k eigenvalues and form a matrix of eigenvectors corresponding to their eigenvalues

When it comes to choosing a **k** number for picking up the first **k** eigenvalues and, thus, forming a matrix of eigenvectors corresponding to their eigenvalues, it is equivalent to asking ourselves "In how many principal components do we want to store the initial data as much as possible?". Therefore, there is no general answer or solution to this problem, because it really depends heavily on the application or scenario one is dealing with. However, there is a commonly-used heuristic which implies plotting the cumulative explained variance ratio relative to the total number of principal components, as it can be illustrated in our experimental context in **Figure 20** below. This approach reveals a way in which one can choose how much information one is willing to discard in order to reduce dimensionality for future steps or algorithms that might require as little memory space as possible in order to have better performance overall. Even so, it's up to us to choose the number of most preserving components of data, which most likely will result to discarding the ones of lesser significance, depending on what we are looking for.

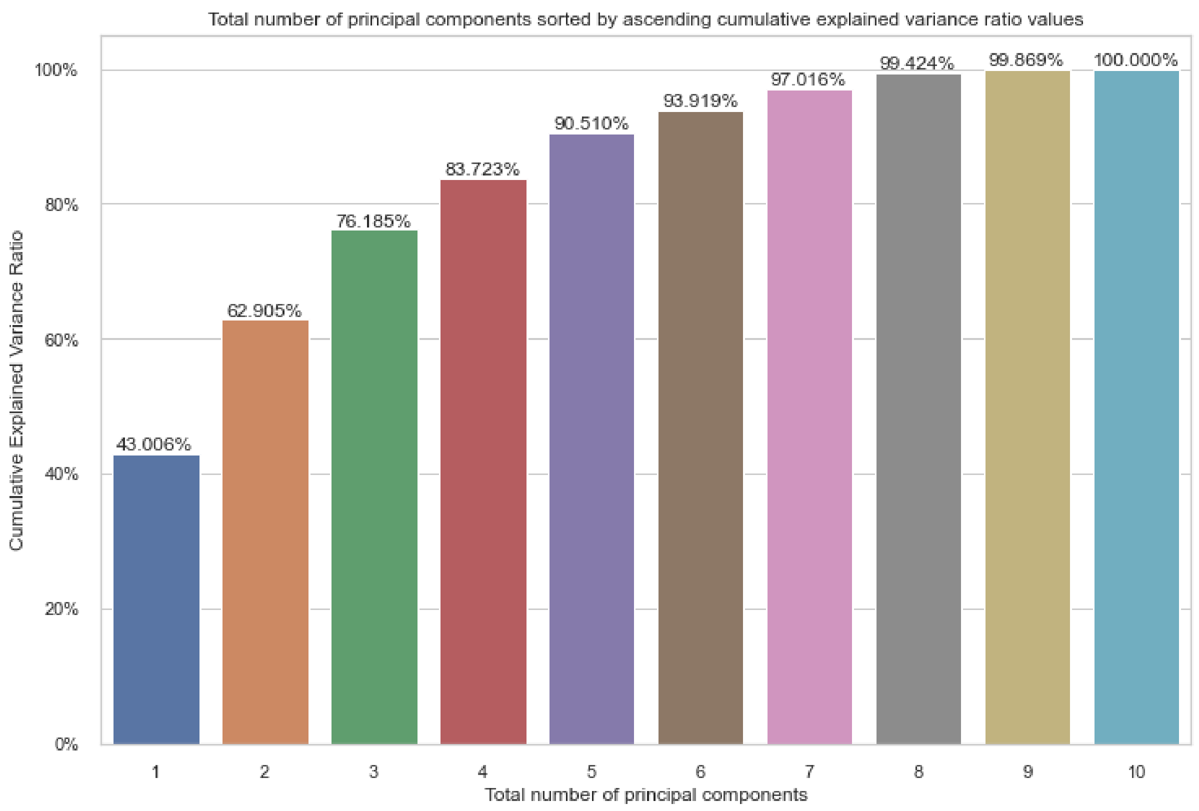


Figure 20. Total number of principal components sorted by ascending cumulative explained variance ratio values

6) Recast the data along the principal component axes

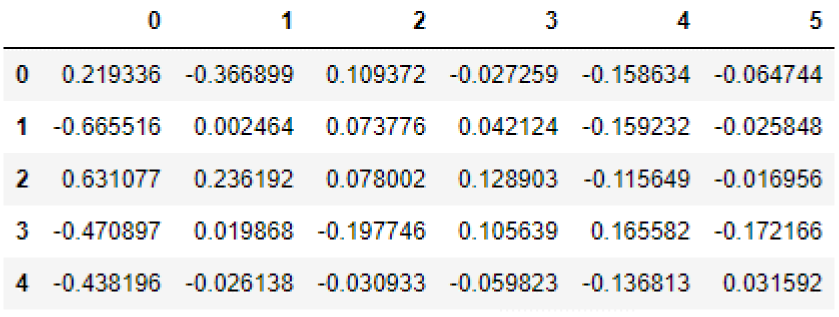
In other words, in this final step, what we actually do is transform the original feature matrix, after it was previously standardized.

Starting over from the previous steps, apart from standardization, there weren't made any changes on the feature matrix. So far, we've simply selected the principal components and formed the **feature vector**, but as far as the input data is concerned, it simply the same in terms of original axes (i.e., in terms of initial variables or features)

In this step, which is the last one as stated above, the aim is to make use of the feature vector formed using the eigenvectors of the covariance matrix in order to reorientate the data from the original axes to the ones represented by the principal components (hence the name Principal Component Analysis). This is done by multiplying the transpose of the original data set by the transpose of the feature vector, as it can be shown below.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

After applying PCA on our data frame of selected features by specifying to keep **the first 6 principal components**, in **Table 5** it can be shown the first 5 values, each of them corresponding to its principal component.



	0	1	2	3	4	5
0	0.219336	-0.366899	0.109372	-0.027259	-0.158634	-0.064744
1	-0.665516	0.002464	0.073776	0.042124	-0.159232	-0.025848
2	0.631077	0.236192	0.078002	0.128903	-0.115649	-0.016956
3	-0.470897	0.019868	-0.197746	0.105639	0.165582	-0.172166
4	-0.438196	-0.026138	-0.030933	-0.059823	-0.136813	0.031592

Table 5. Illustration of the new first 5 values related to each principal component

3.5.3 Clustering

At this point, the data was not only re-scaled more than once, but also had its dimensionality reduced, losing the least significant information possible. Now, as far as our recommender system is concerned, the kind of machine learning problem it revolves around is classification, thus there is the need for clustering our current data and also a need for a number of clusters, since the dataset used doesn't really have any labels worth being used for classification.

,,Clustering is one of the most common unsupervised machine learning problems. Similarity between observations is defined using some inter-observation distance measures or correlation-based distance measures''[MO19]. Currently, there are 5 categories of clustering methods:

- Hierarchical Clustering
- Partitioning Methods
- Density-based Clustering
- Model-based Clustering
- Fuzzy Clustering

In our music recommender system context, it was decided on K-Means algorithm (which is partitioning-method based) as clustering strategy, because it is relatively simple to implement, scales to large data sets, guarantees convergence, can warm-start the positions of centroids, easily adapts to new examples, and, finally, generalizes to clusters of different shapes and sizes, such as elliptical clusters. Some drawbacks of K-Means are:

- **Choosing *k* manually**: It is indicated to use a “Loss vs Clusters” plot to find the optimal k
- **Being dependent on initial values**: For a low **k**, this dependence can be mitigated by running k-means several times with different initial values and picking the best result. As **k** increases, advanced versions of K-Means are needed to pick better values of the initial centroids (called k-means seeding).
- **Clustering data of varying sizes and density**: K-Means has trouble clustering data where clusters are of varying sizes and density.
- **Scaling with number of dimensions**: As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples. This is why before clustering our feature data for our recommendation system, we applied PCA dimensionality reduction technique.

When it comes to **determining the optimal number of clusters**, a variety of measures have been proposed in the literature for evaluating clustering results and the term **clustering validation** is used to design the procedure of evaluating the results of a clustering algorithm. That’s why I will only list and describe as shortly and clearly as possible 3 commonly used approaches, since those 3 approaches are also the only ones provided by **Yellowbrick API** that I used throughout the development of the recommender system for finding the optimal number of clusters. The 3 metrics made available by Yellowbrick API are the following:

- **Elbow method**
- **Silhouette**
- **Calinski-Harabasz index**

Elbow method

Probably the most well-known and commonly-used approach in cluster analysis, the **elbow method** [IDB20] consists of plotting the explained variation as a function of the number of clusters and picking the elbow of curve as the number of clusters to use for the next processing steps. Interestingly enough, the same method can be used, as in our case, to choose the number of principal components by which the data is transformed into a smaller dimensionality, but also for choosing the number of parameters in other data-driven models.

This method involves computing the Within-Cluster-Sum of Squared Errors (WSS on short) for different k numbers of clusters and selecting the k for which the first change in WSS starts to diminish. This way, the idea behind the elbow method is that the explained variation changes rapidly for a small number of clusters and then it slows down leading to an elbow formation in the curve. As such, the elbow point actually proves to be the most optimal number of clusters that we can use for our clustering algorithm.

Therefore, the elbow method curve method is helpful because it shows how increasing the numbers of clusters contribute to separation of the clusters in a meaningful way, not in a marginal way.

Silhouette

„It is a method of interpretation and validation of consistency within clusters of data, which provides a concise graphical representation of how well each object has been classified.

As far as the silhouette value is concerned, it is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.'' [PJR87]

When determining the silhouette coefficient for a point i, the formula is defined as follows:

$$\mathbf{s(i)} = \frac{\mathbf{b(i) - a(i)}}{\mathbf{max\{a(i), b(i)\}}}$$

where $\boldsymbol{b(i)}$ is the smallest average distance of point $\boldsymbol{i}$ to all points in any other cluster and $\boldsymbol{a(i)}$ is the average distance of $\boldsymbol{i}$ from all other points in its cluster. Both distances can be illustrated below:

$$\boldsymbol{a(i)} = \frac{\mathbf{1}}{|\boldsymbol{C_i}| - \mathbf{1}} \sum_{\boldsymbol{j \in C_i, i \neq j}} \boldsymbol{d(i,j)}$$

where $|\boldsymbol{C_i}|$ is the number of points belonging to cluster $\boldsymbol{i}$ and $\boldsymbol{d(i,j)}$ is the distance between data points $\boldsymbol{i}$ and $\boldsymbol{j}$ in the cluster $\boldsymbol{C_i}$ (the sum of distances of a point $\boldsymbol{i}$ to the other points $\boldsymbol{j}$ in the cluster $\boldsymbol{C_i}$ is divided by $|\boldsymbol{C_i}| - \mathbf{1}$ because the distance $\boldsymbol{d(i,i)}$ is not included in the sum). $\boldsymbol{a(i)}$ can be interpreted as a measure of how well $\boldsymbol{i}$ is assigned to its cluster (the smaller the value, the better the assignment). The cluster with the smallest mean dissimilarity is said to be the "neighbouring cluster" of $\boldsymbol{i}$ because it is next best fit cluster for point $\boldsymbol{i}$.

$$\mathbf{b(i)} = \min_{\mathbf{j \neq i}} \frac{\mathbf{1}}{|\mathbf{c_j}|} \sum_{j \in C_i} d(i,j)$$

When it comes to the distance metrics by which the silhouette value can be computed, it can be computed with any distance metric (usually Euclidian or Manhattan distance).

Calinski-Harabasz Index

,,The Calinski-Harabasz Index is based on the idea that clusters that are very compact and well-spaced from each other are good clusters. The index is computed by dividing the variance of the sums of squares of the distance between the cluster centers. The higher the Calinski-Harabasz Index value, the better the clustering model. The formula for Calinski-Harabasz Index is defined as:

$$CH_k = \frac{B}{k-1} \cdot \frac{n-k}{w}$$

where k is the number of clusters, n is the number of records in data, BCSM (between cluster scatter matrix) computes the separation between clusters and WCSM (within cluster scatter matrix) calculates compactness within clusters.'' [IDB20]

Considering the evaluation techniques described above, after making multiple data experiments on a number of clusters ranging from 2 to 10 with different numbers of principal components for reducing data and analyzing the results of all the three evaluation metrics, I found particularly interesting the two following results: in **Figure 21** there are the results for the optimal number of clusters by **Silhouette** metric and in **Figure 22** there are the results for the optimal number of clusters by **Elbow method/Distortion** metric. Both metrics were used for evaluating the optimal number of clusters for K-Means algorithm.

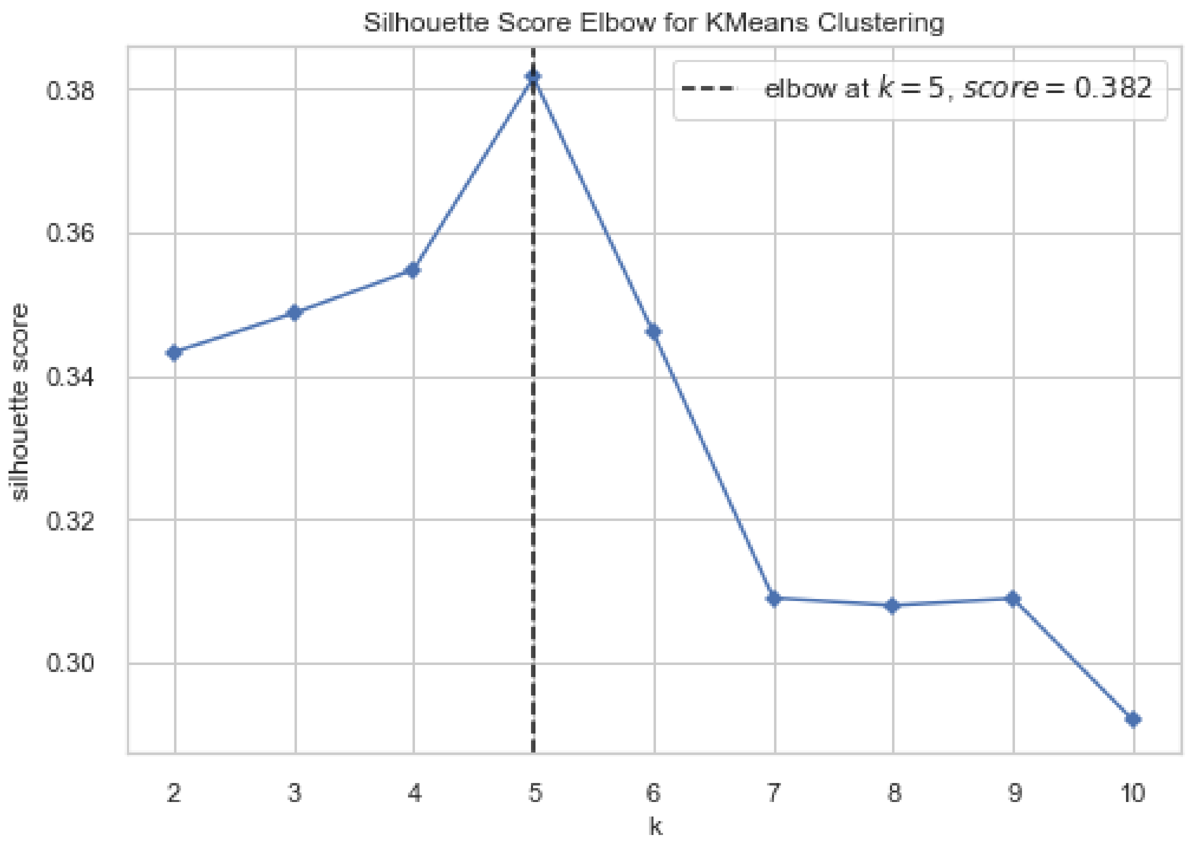


Figure 21. Silhouette Score for K-Means Clustering

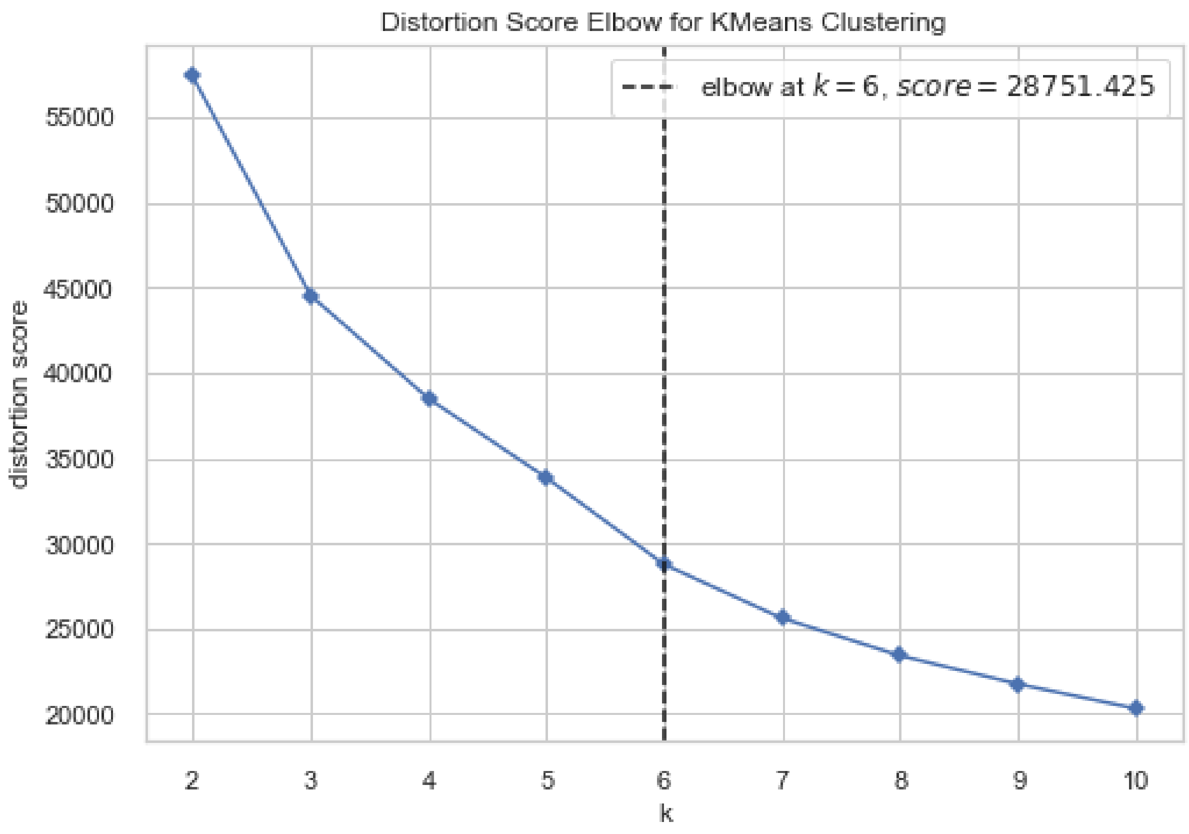


Figure 22. Distortion/Elbow method Score for K-Means Clustering

As far as Calinski-Harabasz Index is concerned, the only result worth noticing is the one in **Figure 23**, where for only 2 principal components, we get the highest score on 5 clusters. But, by deciding on 2 principal components and 5 clusters with this metric, we only preserve like 62 or 63% of the initial information from our dataset of selected features, which can have a meaningful impact such that there are much bigger chances of making bad recommendations.

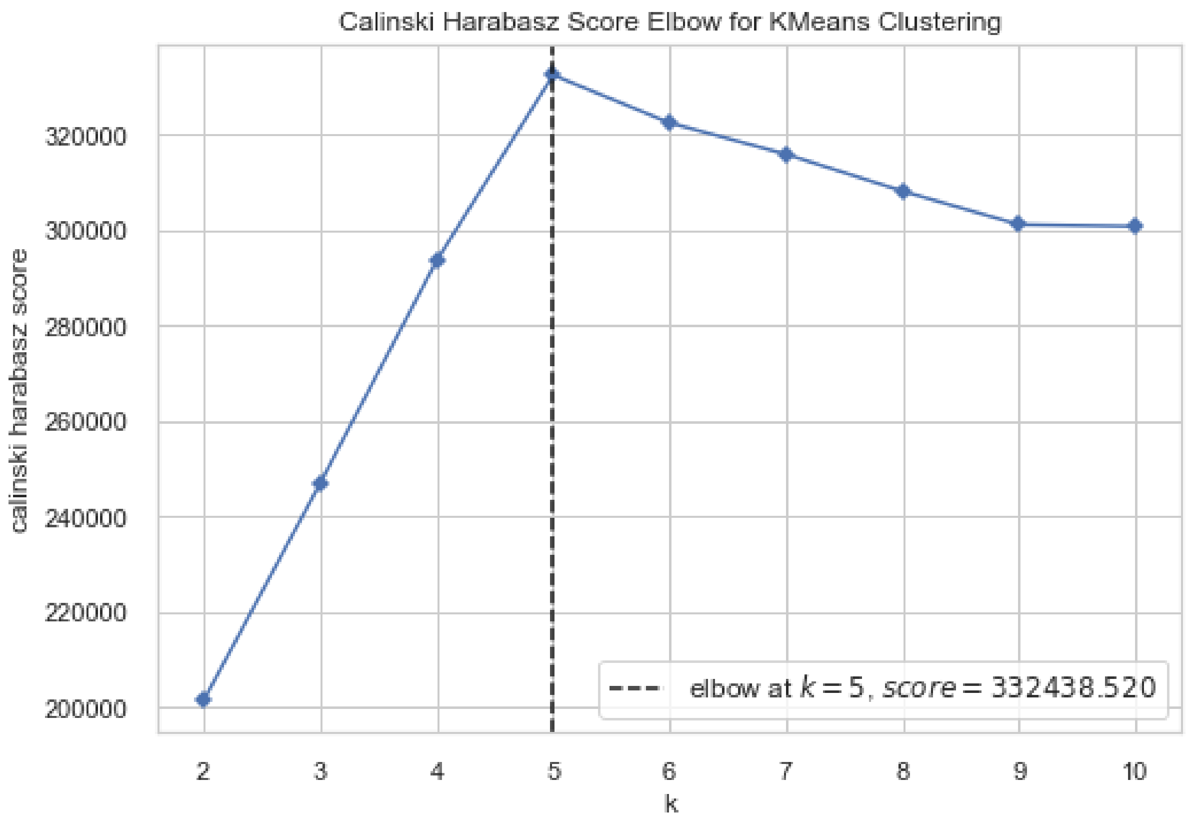


Figure 23. Calinski-Harabasz Score for K-Means Clustering

Now, in order to decide on a final number of principal components and clusters for K-Means clustering algorithm, we can also visualize the results of the data being distributed by K-Means clustering algorithm from the 3 different metrics with different number of clusters in **Figure 24**, **Figure 25** and **Figure 26**. From analyzing the results, the decision that was taken was to keep the data with 6 principal components and cluster it on 6 clusters, because this way the data has more variance preservation, more clusters to be grouped by and most importantly it is more evenly distributed than the way the other metrics evaluated the most optimal number of clusters regardless of number of principal components, by which the data is reduced and preserved as best as possible.

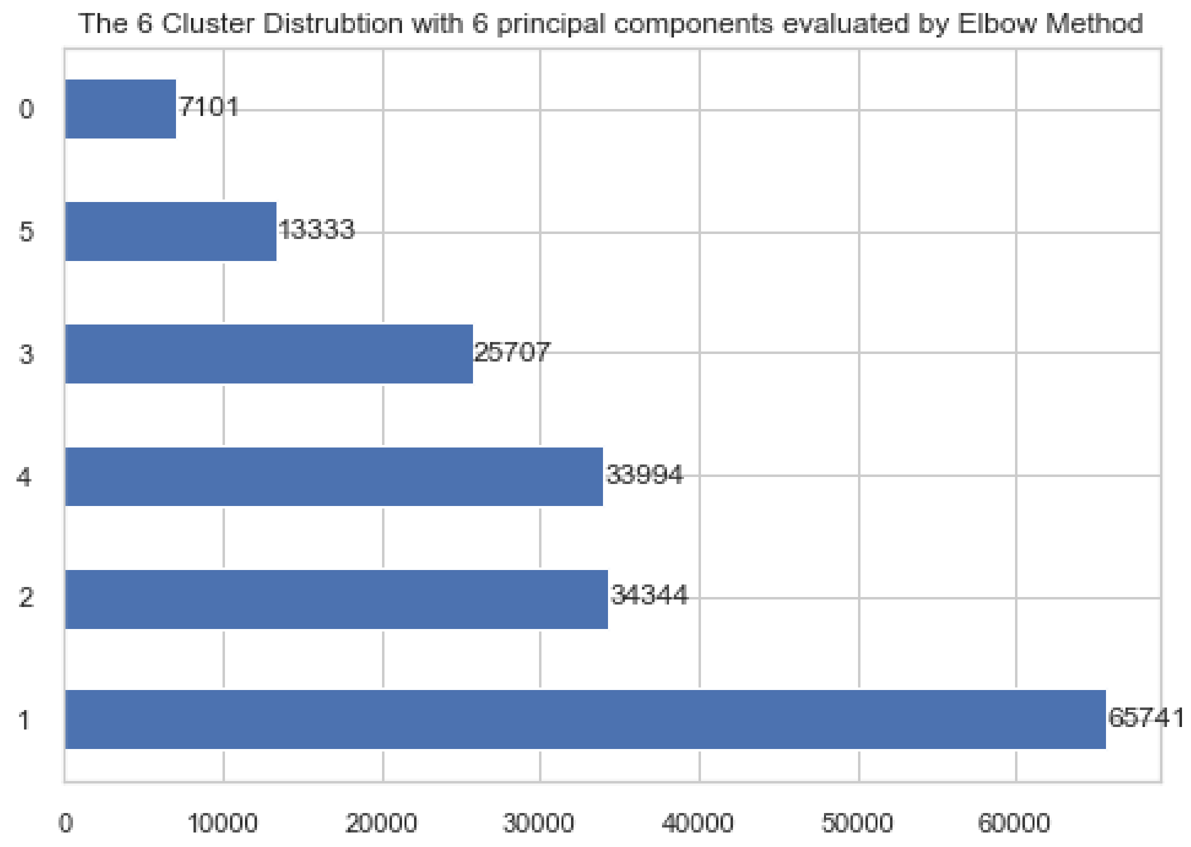


Figure 24. The 6 Cluster Distribution with 6 principal components evaluated by Elbow Method

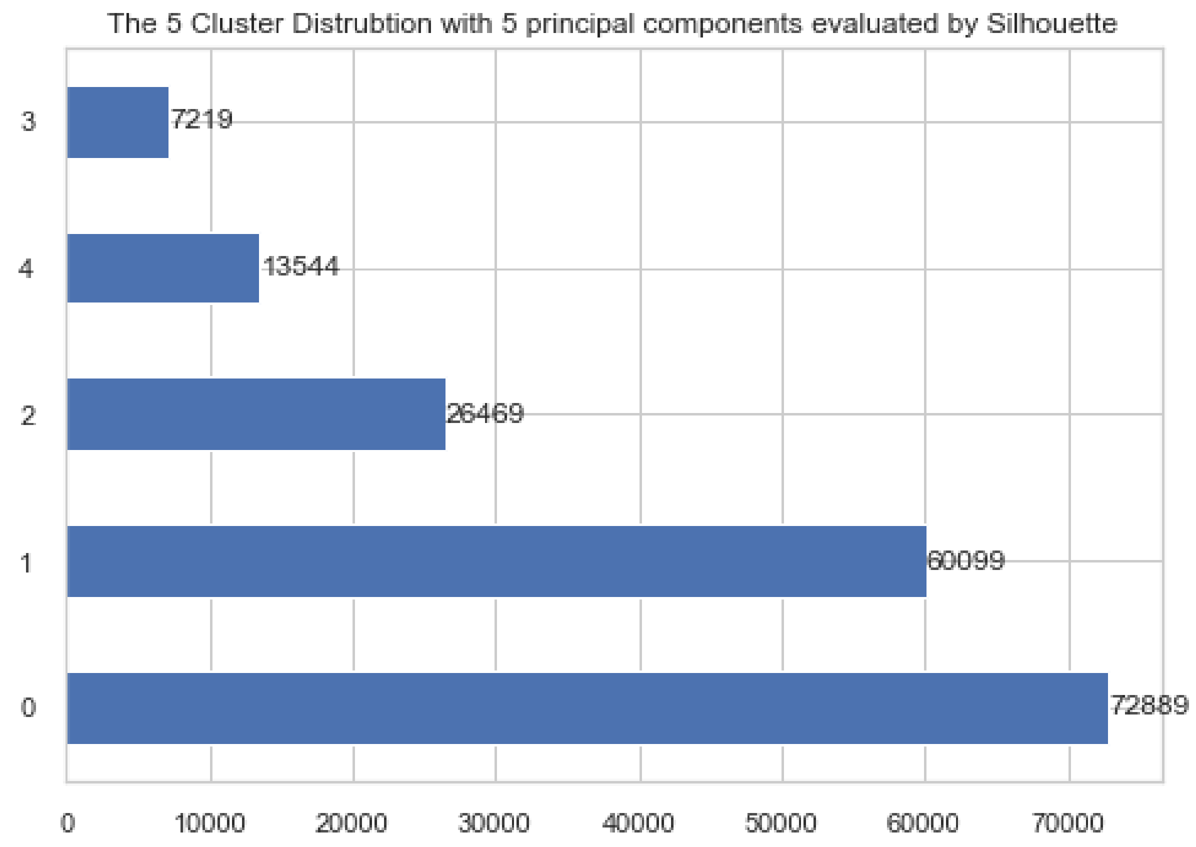


Figure 25. The 5 Cluster Distribution with 5 components evaluated by Silhouette

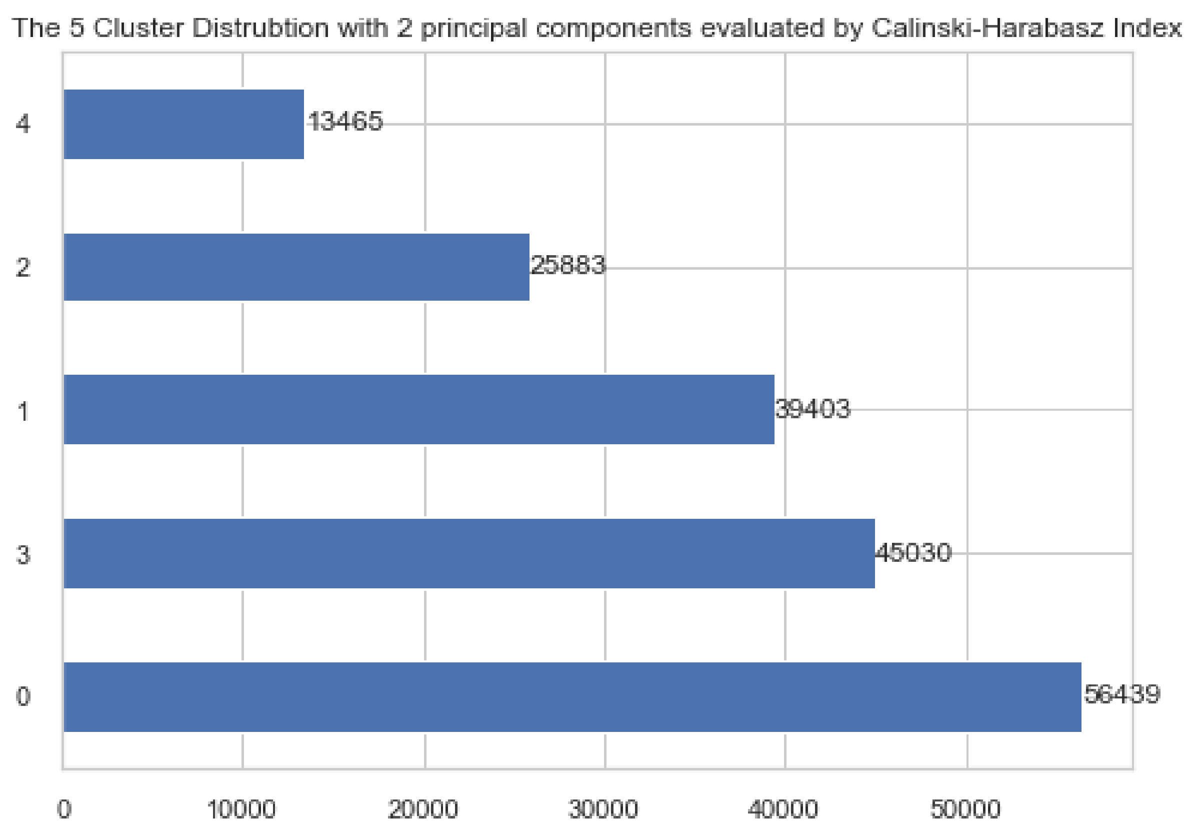


Figure 26. The 5 Cluster Distribution with 2 principal components evaluated by Calinski-Harabasz Index

When it comes to **dependence on the initialization of the centroids** or **mean points**, **K-Means++** [SV21] is the algorithm which is used to overcome this particular drawback of K-Means algorithm. It guarantees a more intelligent initialization of centroids and improves the nature of clustering. Apart from the initialization, the rest of the algorithm is the same as the standard K-Means algorithm. In other words, **K-Means++** is the standard K-Means algorithm coupled with a smarter initialization of the centroids.

The steps involved in initialization algorithm are the following:

I. Randomly select the first centroid from the data points
II. For each data point compute its distance from the nearest, previously chosen centroid.
III. Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid).
IV. Repeat steps 2 and 3 until k centroids have been sampled.

After deciding on the number of clusters and the number of principal of components which influenced the optimal number of clusters, we get the following plot in **Figure 27** after clustering the data, where there are represented only the first 2 principal components. The purple area represents the data in the first cluster, yellow represents the second cluster, turquoise represents the third cluster, light blue represents fourth cluster, green represent the fifth one and, finally, the area in dark blue represents the sixth and last cluster.

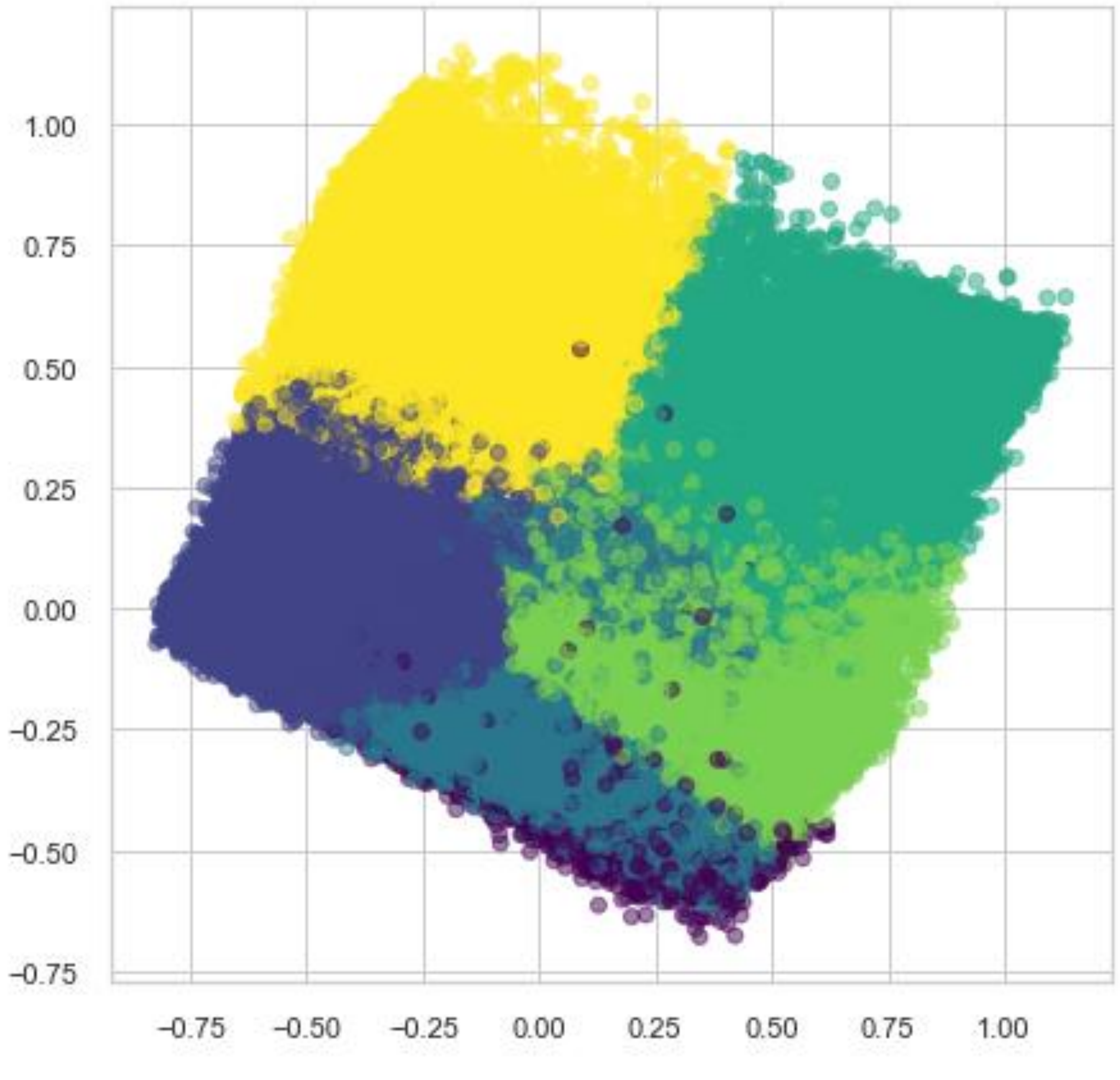


Figure 27. Illustration of the first two principal components of the feature data reduce to 6 dimensions with PCA distributed on 6 clusters

3.6 Recommendation algorithm

Eventually, at this point, the data has already been preprocessed and the system is now ready for the recommendation algorithm.

The recommendation algorithm consists of a method ***recommend_songs(song_number, number_of_recommendations)*** which takes as parameters the **supposed song that is currently played** (which is parsed as a number which represents the row of the dataset) and the **number of recommendations** which are supposed to be displayed while that song is being played.

3.6.1 Train test split

The first step the recommendation algorithm consists of is splitting the dataset into training and test sets.

A goal of supervised learning is to build a model that performs well on new data. In case there is new data, it’s a good idea to see how well the model performs on it. The problem is that there might not be new data, but the experience can be simulated with a procedure like a train test split. In this approach there are included both data features as input and the cluster labels as output target.

When splitting data, there isn’t usually a defined, concrete way, but some common approaches are **70-30** ratio, **80-20** or even a **90-10** ratio, depending mainly on how vast the data we are dealing with actually is. In this context, I have only managed to test data with an **80-20** train test split, having **144,176** songs for training set and **36,044** songs for testing set.

Next, the model is trained in order to make predictions, and after making predictions, both predicted values and test values are compared in order to evaluate the performance of the model created after training.

3.6.2 k-NN (k-Nearest Neighbours) for song classification

As mentioned earlier, now that we have our data split into training and test sets, we need a machine learning model that needs to fit our data and based on its training is able to predict labels for new songs as best as possible in order to get the best possible recommendations. In this algorithm, it was chosen k-NN as learning method.

„k-NN is a non-parametric supervised learning method which is used both for classification and regression. In both cases, the input consists of the k closest training examples in a dataset. The output depends on whether k-NN is used for classification or regression (in our context it will be used for only classification). In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbours (*k* is a positive integer, typical small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour.” [KN22]

3.6.3 Determining the cluster membership of the song

In this step, it is determined the cluster membership of the song that is assumed to be currently played. This is done by first checking if the song belongs either to training set or test set. If it belongs to the training set, we simply take the cluster label assigned by the K-Means clustering algorithm and if it belongs to the test set, we take the cluster label predicted by k-NN algorithm.

3.6.4 Sampling songs

Since there are a lot of songs, especially for cluster with label 1, the set of songs corresponding to cluster label will be sampled in such a way that not all songs that belong to that cluster label will be used for song recommendations. This way, computation time will be drastically reduced and the recommender system can provide different recommendations based on the same song that is currently played, offering the possibility to discover other songs.

3.6.5 TF-IDF

,,TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

This is done by multiplying two metrics: how many times a word appears in a document (TF), and the inverse document frequency of the word across a set of documents (IDF).'' [BS19]

Term frequency, $tf(t,d)$, is the relative frequency of term t within document d.

$$\mathbf{tf(t,d)} = \frac{\mathbf{f_{t,d}}}{\mathbf{\Sigma_{t' \in d} f_{t',d}}}$$

Where $f_{t,d}$ is the raw count of a term in a document, i.e., the number of times that term t occurs in document d (counting each occurrence of the same term separately). The denominator is simply the total number of terms in document d (counting each occurrence of the same term separately).

,,The **inverse document frequency** is a measure of how much information the word provides, i.e., if it is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term and then taking the logarithm of that quotient): '' [TI11]

$$\mathbf{f(t,d)} = \mathbf{log} \frac{\mathbf{N}}{\mathbf{|\{d \in D: t \in D\}|}}$$

Where:

- $\mathbf{N}$: total number of documents in corpus $\mathbf{N} = |\mathbf{D}|$
- $|\{\mathbf{d} \in \mathbf{D}: \mathbf{t} \in \mathbf{D}\}|$: number of documents where the term $\mathbf{t}$ appears (i.e., $\mathbf{tf}(\mathbf{t}, \mathbf{d}) \neq \mathbf{0}$). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to 1 + $|\{\mathbf{d} \in \mathbf{D}: \mathbf{t} \in \mathbf{D}\}|$**.**

In our context, our “documents” and “collection of documents” are actually the songs and collection of songs and the “texts” in each “document” are the lyrics of each song.

This particular algorithm from Information Retrieval is used to measure the similarity of songs in terms on lyrics only. It is important to consider the fact that since the songs are clustered and sampled, by having a smaller number than the total number of songs of a cluster label that the currently played song belongs to as well, the corpus (or set of songs) is drastically reduced as well, reducing this way not only the time complexity, but also the space complexity, which will definitely make an impact on our recommendations.

3.6.6 Cosine similarity

Based on tf-idf score from songs based on lyrics similarity, it is done a further step for lyrics similarity, by computing the cosine similarity of all songs in corpus and the song that is currently played.

$$\boldsymbol{cos(\theta) = \frac{C \cdot P}{\|C\| \cdot \|P\|} = \frac{\sum_{i=1}^{n} C_i P_i}{\sqrt{\sum_{i=1}^{n} C_i^2}\sqrt{\sum_{i=1}^{n} P_i^2}}}$$

Where C is a song selected from the cluster where the supposedly played song belongs to and P is the supposedly played song.

3.6.7 Rescaling popularity of the sampled songs and Euclidean distance from the all sampled songs to played song

Io order to make use of the popularity of the songs, lyrics similarity and audio characteristics in a custom distance, it is firstly needed to make sure that all of them have values in the same range. Since, cosine function is in [-1, 1] range, we need to re-scale both popularity of sampled songs and Euclidean distances from all sampled songs to the played song to [-1,1] via **Min-Max** scaling technique as well in order for the custom distance to be successfully used for generating the most relevant songs

3.6.8 Computing similarity distances and sorting songs based on them

At this point, the similarity distance between a song selected from the cluster where the supposedly played song belongs to and the supposedly played song is computing using the following custom-made formula:

$$\boldsymbol{CustomDist(c,p) = \frac{MinMaxPopularity(c) + cos(c,p) + MinMaxEuclideanDist(c,p)}{3}}$$

$$\boldsymbol{MinMaxPopularity(c) = -1 + \frac{Popularity(c) - min(P) * 2}{max(P) - min(P)}}$$

$$\boldsymbol{MinMaxEuclideanDist(c) = 1 - \frac{EuclideanDist(c,p) - minEuclideanDist(D) * 2}{maxEuclieanDist(D) - minEuclideanDist(D)}}$$

Where MinMaxPopularity(c) is the popularity rating of the current song being compared scaled initially from [0, 100] to [-1, 1], MinMaxEuclideanDist(c, p) is the Euclidean distance based on principal component values from the played song to the current song scaled to [-1, 1] , $\cos(c, p)$ is cosine similarity on lyrics between current song and played song, $\boldsymbol{Popularity(c)}$ is the initial popularity rating value of the current song, $\boldsymbol{min(P)}$ is the minimum value of the Popularity vector formed of the song samples, $\boldsymbol{max(P)}$ is the maximum value of the Popularity vector formed of the song samples, $\boldsymbol{EuclideanDist(c,p)}$ is the Euclidean distance based on principal component values from the played song to the current song, $\boldsymbol{maxEuclieanDist(D)}$ is the maximum Euclidean distance based on principal component values from the played song to the current song, $\boldsymbol{minEuclideanDist(D)}$ is the minimum Euclidean distance based on principal component values from the played song to the current song.

By scaling all similarity measurements [-1,1], it is this way established the fact that any song in range (0,1] is to some degree similar to a song, which suggestions are made on, and in range [-1,0) is less and less similar to some degree to a song to the song, which suggestions are made on.

When computing all custom similarity distances, they are stored in an array of tuples which consist of custom distance metric value, artist name, song title, type of playlist and eventually a preview link to the song. And after we append all custom similarity distances in array, we simply sort the array based on these computed metrics.

3.6.9 Displaying the best recommendations in descending order

Finally, in this step, after computing all distance metrics of all songs to the currently played song, we simply display several best recommendations, depending on how many we are interested in, for the currently played song.

4 IMPLEMENTATION AND EVALUATION OF THE EXPERIMENTAL SYSTEM

4.1 Programming languages, libraries and platforms used

4.1.1 Programming languages

Python

„Python is a high-level, interpreted, dynamically-typed, garbage-collected and general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming and is often described as a "batteries included" language due to its comprehensive standard library." [AP12]

4.1.2 Libraries

NumPy

„NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more." [NP08]

Pandas

„Pandas is a Python package that provides fast, flexible and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block of doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language.''[AR20]

A few of the main features that Pandas does well are:

- Easy handling of missing data (represented as **NaN**, **NA**, or **NaT**) in floating point as well as non-floating-point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let **Series**, **DataFrame**, etc. automatically align the data for you in computations.
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases and saving/loading data from the ultrafast HDF5 format
- Time series-specific functionality: data range generation and frequency conversion, moving window statistics, date shifting and lagging

Scikit-learn

,,Scikit-learn is a free software machine learning library for the Python programming language''[FP11]. ,,It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.''[NSP21]

Multiprocessing

,,Multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.

The multiprocessing module also introduces APIs which do not have analogs in the threading module. A prime example of this is the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism). The following examples demonstrates the common practice of defining such functions in a module so that child processes can successfully import that module.’’[MP21]

4.1.3 Platforms

Anaconda

„Anaconda is a distribution of Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analysis, etc.), which aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux and MacOS.’’ [AA12] „Package versions in Anaconda are managed by the package management system *conda*.’’[C17] This package manager was spun out as a separate open-source package as it ended up being useful on its own and for things other than Python.

PyCharm

„PyCharm is an integrated cross-platform (with Windows, MacOS and Linux versions) development environment (IDE) used in computer programming, specifically for the Python programming language’’[JS]. „It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes) and supports web development with Django as well as data science with Anaconda.’’[HE19]

Jupyter Notebook

Jupyter Notebook is a web-based interactive computational environment for creating notebook documents.

A Jupyter Notebook document is a browser-based REPL containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media. Underneath the interface, a notebook is a JSON document, following a versioned schema, usually ending with the “. ipynb” extension.

Jupyter Notebook can connect to many kernels to allow programming in different languages. A Jupyter kernel is a program responsible for handling various types of requests (code execution, code completions, inspection) and providing a reply. Kernels talk to other components of Jupyter using ZeroMQ, and thus can be on the same or remote machines. Unlike many other Notebook-like interfaces, in Jupyter, kernels are not aware that they are attached to a specific document, and can be connected to many clients at once. Usually kernels allow execution of only a single language, but there are a couple of exceptions. By default, Jupyter Notebook ships with the IPython kernel.

A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell. To simplify visualization of Jupyter notebook documents on the web, the nbconvert library is provided as a service through NbViewer, which can take a URL to any publicly available notebook document, convert it to HTML on the fly and display it to the user.

4.2 Visualization modules for data and results

Matplotlib

Matplotlib is a comprehensive library for creating static, animated and interactive visualizations in Python and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. [MPL02]

Seaborn

Seaborn [SB12] is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with *pandas* data structures.

Some of the functionalities that Seaborn offers are:

- A dataset-oriented API for examining relationships between multiple variables
- Convenient views onto the overall structure of complex datasets
- Specialized support for using categorical variables to show observations or aggregate statistics
- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds of dependent variables
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations

- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mappings and statistical aggregations to produce informative plots.

Yellowbrick

„Yellowbrick extends the Scikit-Learn API to make model selection and hyperparameter tuning easier. Under the hood, it's using Matplotlib. The primary goal of Yellowbrick is to create a sensical API similar to Scikit-Learn.

Visualizers are the core objects in Yellowbrick. They are similar to transformers in Scikit-Learn. Visualizers can wrap a model estimator – similar to how the "ModelCV" (e.g. RidgeCV, LassoCV) methods work.''[YB16]

Some of the most popular visualizers from Yellowbrick include:

- **Feature Visualization**
- **Classification Visualization**
- **Regression Visualization**
- **Clustering Visualization**
- **Model Selection Visualization**
- **Target Visualization**
- **Text Visualization**

4.3 Experimental results and Evaluation of the experimental system

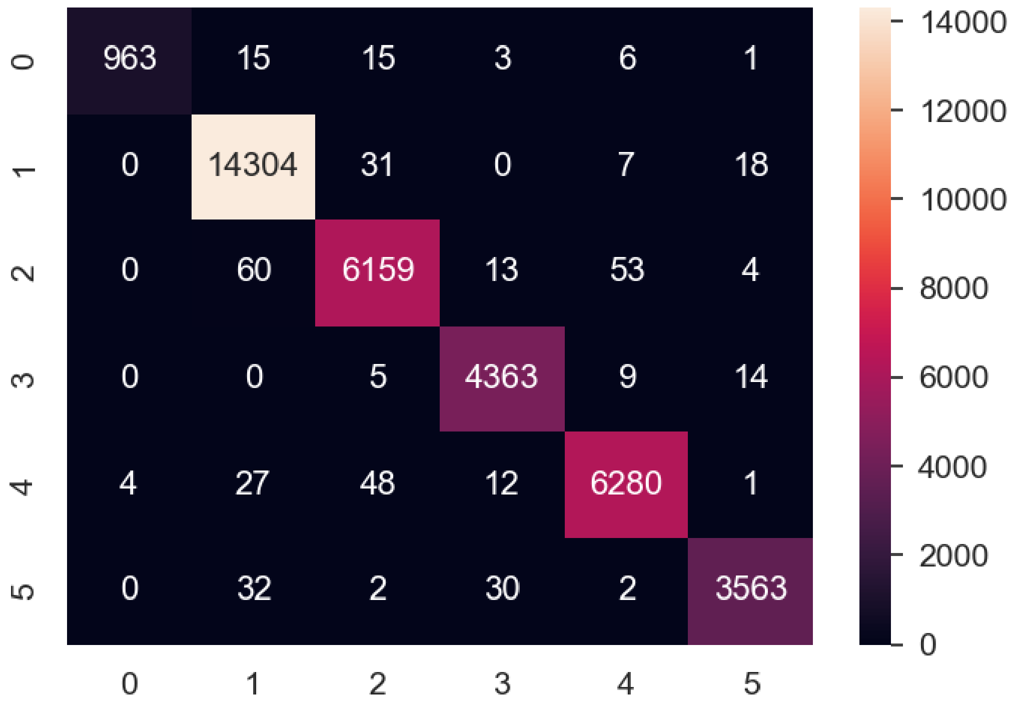
4.3.1 Evaluation of experimental system

As far as the evaluation of experimental system is concerned, there this system is content-based and there is no data about the user, this means the experiment is only meant to simulate unauthenticated users who access content and get recommendations based on item-similarities. As such, there were few evaluations metrics to consider, however the most important ones that I could think of and apply were **Confusion Matrix** and the **finding the optimal *k* for k-NN**.

Confusion Matrix [SS97] is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class (or it can be vice-versa, since both variants are found in literature).

In the context of our actual system, the confusion matrix evaluates how well k-NN knows to label songs by clusters. This kind of evaluation can give an idea of how well the new songs are labeled such that we can get recommendations as relevant as possible based on song characteristics. [PD11]

In **Figure 28**, we have the results of the evaluation in which we can see that the system does a pretty decent job in labeling new songs in the appropriate cluster.



	0	1	2	3	4	5
0	963	15	15	3	6	1
1	0	14304	31	0	7	18
2	0	60	6159	13	53	4
3	0	0	5	4363	9	14
4	4	27	48	12	6280	1
5	0	32	2	30	2	3563

Figure 28. Confusion Matrix Evaluation on Test Data

As for the optimal number of neighbours, from analyzing the graph in **Figure 29**, it can be concluded that the most performing value is in range [25,30], after testing the k-NN performance on range

[1,30].

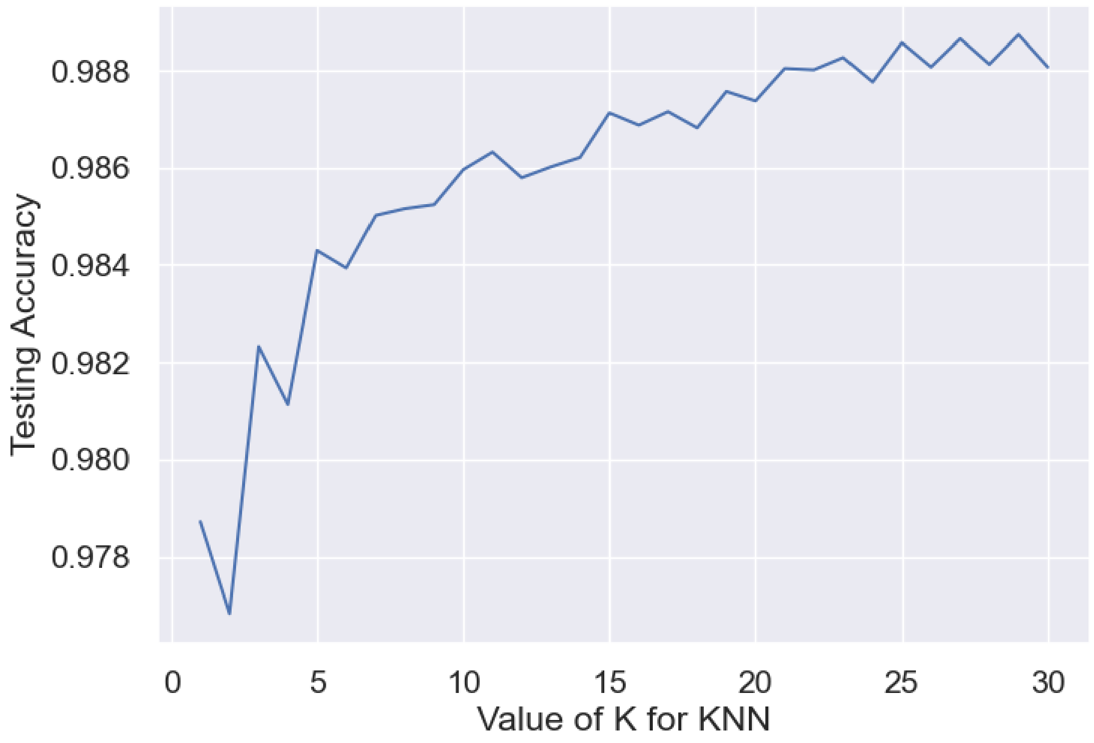
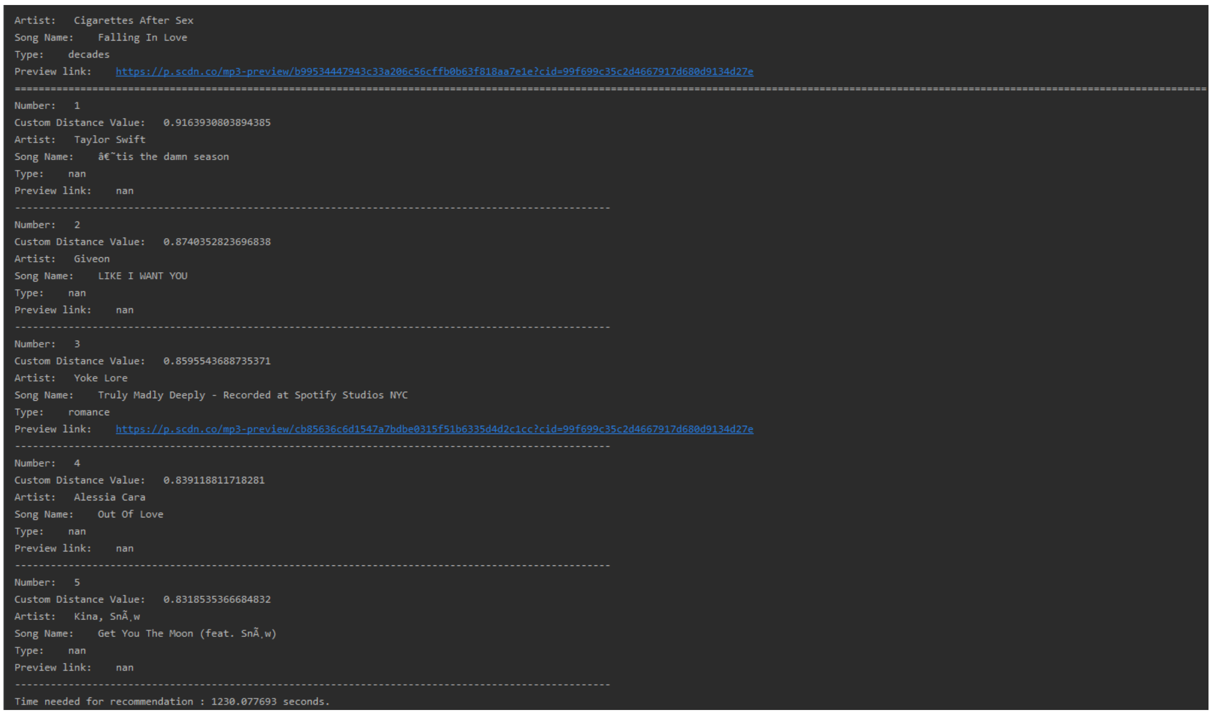


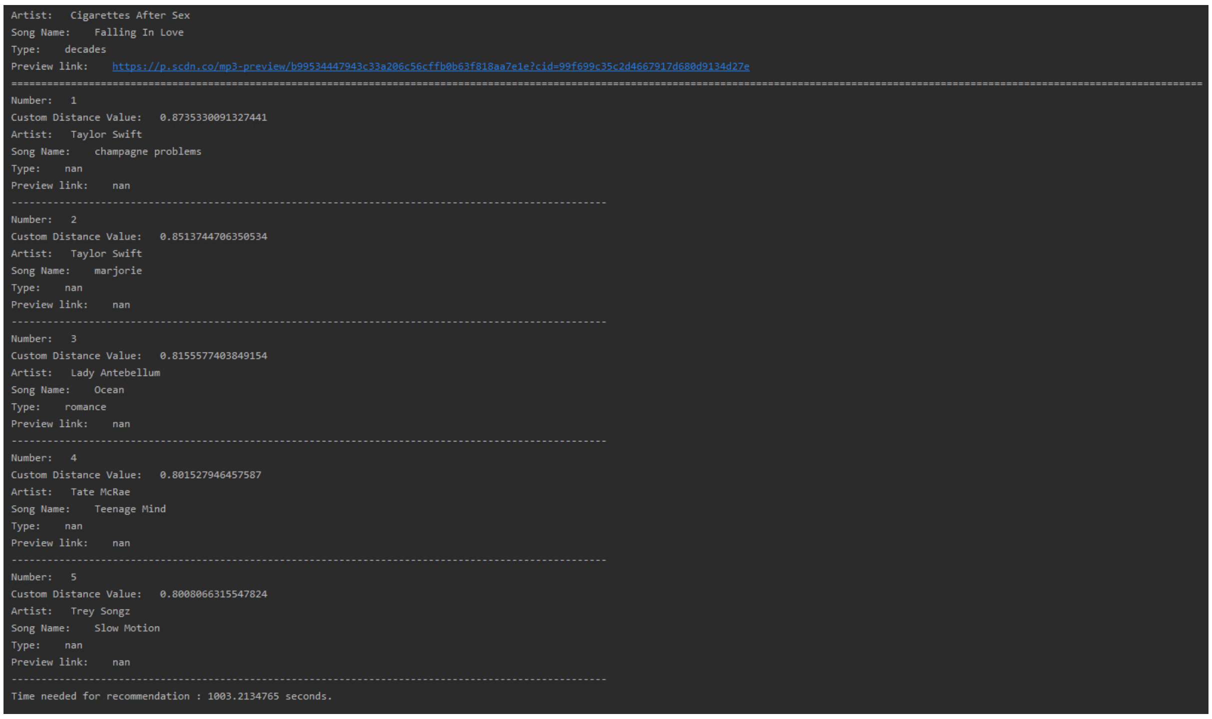
Figure 29. Most performing values of k for k-NN in 1-30 range

4.3.2 Experimental results

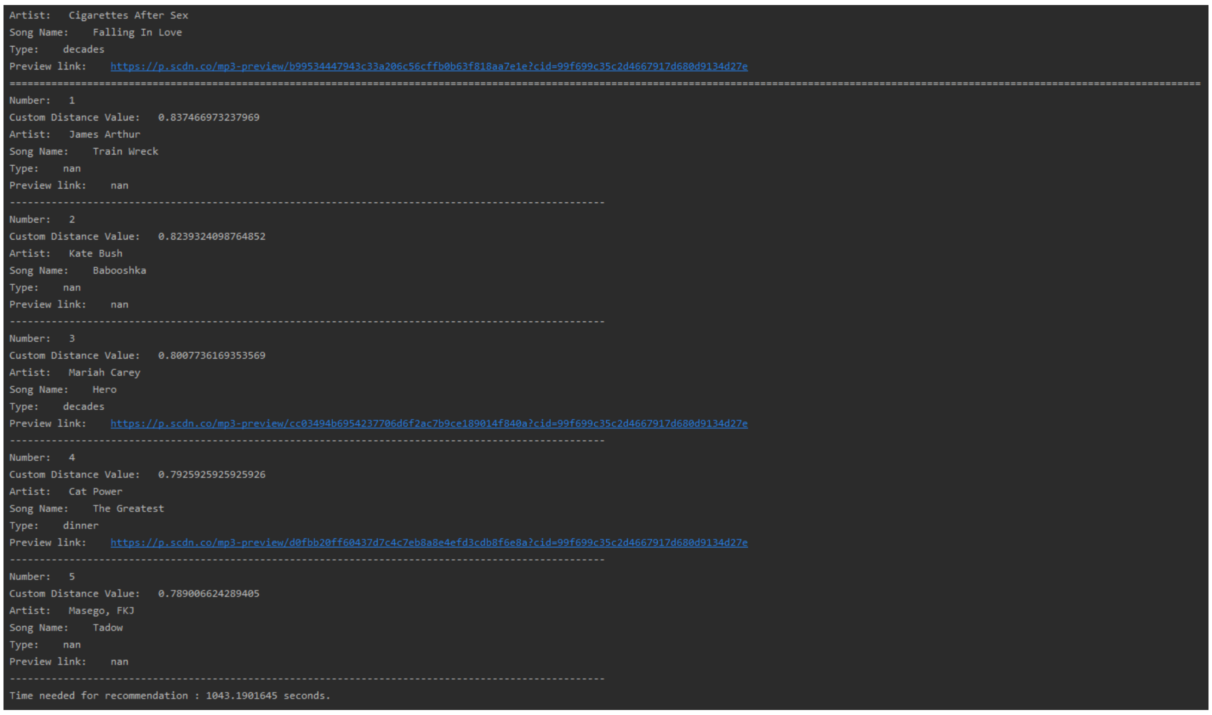
In this section, I provided some result samples, simulating a scenario in which the user plays the song (in this case *Falling in Love* by *Cigarettes After Sex*). The recommendation was based on a sample of 1000 random songs from the cluster the song played belongs to and the number of provided recommendations were 5. The time displayed for the recommendation system (meaning data loading, preprocessing steps and recommendation algorithm) is provided as well and the results in **Figure 30**.



```
Artist:   Cigarettes After Sex
Song Name:    Falling In Love
Type:    decades
Preview link:    https://p.scdn.co/mp3-preview/b99534447943c33a206c56cffb0b63f818aa7e1e?cid=99f699c35c2d4667917d680d9134d27e
=================================================================================================================================================================================================
Number:   1
Custom Distance Value:   0.9163930803894385
Artist:   Taylor Swift
Song Name:    â€˜tis the damn season
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   2
Custom Distance Value:   0.8740352823696838
Artist:   Giveon
Song Name:    LIKE I WANT YOU
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   3
Custom Distance Value:   0.8595543688735371
Artist:   Yoke Lore
Song Name:    Truly Madly Deeply - Recorded at Spotify Studios NYC
Type:    romance
Preview link:    https://p.scdn.co/mp3-preview/cb85636c6d1547a7bdbe0315f51b6335d4d2c1cc?cid=99f699c35c2d4667917d680d9134d27e
----------------------------------------------------------------------------------------------------
Number:   4
Custom Distance Value:   0.839118811718281
Artist:   Alessia Cara
Song Name:    Out Of Love
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   5
Custom Distance Value:   0.8318535366684832
Artist:   Kina, SnÃ¸w
Song Name:    Get You The Moon (feat. SnÃ¸w)
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Time needed for recommendation : 1230.077693 seconds.
```



```
Artist:   Cigarettes After Sex
Song Name:    Falling In Love
Type:    decades
Preview link:    https://p.scdn.co/mp3-preview/b99534447943c33a206c56cffb0b63f818aa7e1e?cid=99f699c35c2d4667917d680d9134d27e
=================================================================================================================================================================================================
Number:   1
Custom Distance Value:   0.8735330091327441
Artist:   Taylor Swift
Song Name:    champagne problems
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   2
Custom Distance Value:   0.8513744706350534
Artist:   Taylor Swift
Song Name:    marjorie
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   3
Custom Distance Value:   0.8155577403849154
Artist:   Lady Antebellum
Song Name:    Ocean
Type:    romance
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   4
Custom Distance Value:   0.801527946457587
Artist:   Tate McRae
Song Name:    Teenage Mind
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Number:   5
Custom Distance Value:   0.8008066315547824
Artist:   Trey Songz
Song Name:    Slow Motion
Type:    nan
Preview link:    nan
----------------------------------------------------------------------------------------------------
Time needed for recommendation : 1003.2134765 seconds.
```



```
Artist:   Cigarettes After Sex
Song Name:    Falling In Love
Type:    decades
Preview link:    https://p.scdn.co/mp3-preview/b99534447943c33a206c56cffb0b63f818aa7e1e?cid=99f699c35c2d4667917d680d9134d27e
==============================================================================================================================================================================================
Number:   1
Custom Distance Value:   0.837466973237969
Artist:   James Arthur
Song Name:    Train Wreck
Type:    nan
Preview link:    nan
-------------------------------------------------------------------------------------------------
Number:   2
Custom Distance Value:   0.8239324098764852
Artist:   Kate Bush
Song Name:    Babooshka
Type:    nan
Preview link:    nan
-------------------------------------------------------------------------------------------------
Number:   3
Custom Distance Value:   0.8007736169353569
Artist:   Mariah Carey
Song Name:    Hero
Type:    decades
Preview link:    https://p.scdn.co/mp3-preview/cc03494b6954237706d6f2ac7b9ce189014f840a?cid=99f699c35c2d4667917d680d9134d27e
-------------------------------------------------------------------------------------------------
Number:   4
Custom Distance Value:   0.7925925925925926
Artist:   Cat Power
Song Name:    The Greatest
Type:    dinner
Preview link:    https://p.scdn.co/mp3-preview/d0fbb20ff60437d7c4c7eb8a8e4efd3cdb8f6e8a?cid=99f699c35c2d4667917d680d9134d27e
-------------------------------------------------------------------------------------------------
Number:   5
Custom Distance Value:   0.789006624289405
Artist:   Masego, FKJ
Song Name:    Tadow
Type:    nan
Preview link:    nan
-------------------------------------------------------------------------------------------------
Time needed for recommendation : 1043.1901645 seconds.
```

Figure 30. Recommendations based on the same song

5 CONCLUSIONS AND SUGGESTIONS ON IMPROVEMENTS

The recommender system is far from being one which could work in a real-world scenario. There is no user data which can be used for a hybrid approach which could combine the advantages of both content-based and collaborative-filtering approaches. The number of lyrics collected so far are very few for TF-IDF to work properly and there is also a problem for choosing the number of songs for sampling. If we choose all the songs for recommedations based on a song belonging to a very large cluster, not only we will most likely make the same recommendations, but takes a lot of time to compute all similarity metrics.

Overall, there are a lot of improvements to be done, not only efficiency-wise, but also especially performance-wise.

Therefore, as future work, I would consider collecting data about users, which should be integrated well with the current dataset, more lyrics to have better accuracy in providing suggestions as relevant as possible and maybe implementing some multi-processing algorithms for computing multiple similarity distance in parallel. I would also take into consideration the majority vote system of k-NN into the one based on probabilities, such that more songs from other clusters can be explored.

6 WEB AND BIBLIOGRAPHICAL REFRENCES AND RELATED WORK

[Roc19] – Baptiste Rocca, Introduction to recommender systems, Jun 3 2019, available on https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

[J] – About Jovian – Data Science and Machine Learning, available on https://blog.jovian.ai/about

[PL] – Paul Lamere, Welcome to Spotipy! – spotipy 2.0 documentation, available on https://spotipy.readthedocs.io/en/master/

[PC22] – Pitch class,available on http://openmusictheory.com/pitch(Class).html

[TS22] – Eddie Bond, Time signatures: how music is organized and measured, May 10 2022, available on https://www.skoove.com/blog/time-signatures-explained/

[AS21] – Ashray Saini, Different Type of Correlation Metrics Used by Data Scientists, September 22 2021, available on https://www.analyticsvidhya.com/blog/2021/09/different-type-of-correlation-metrics-used-by-data-scientist/

[ZJ21] – Zakaria Jaadi, A Step-by-Step Explanation of Principal Component Analysis, published on April 1 2021, updated on December 1 2021, available on https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[MO19] – Matt. O, 10 Tips for Choosing the Optimal Number of Clusters, Jan 27 2019, available on https://towardsdatascience.com/10-tips-for-choosing-the-optimal-number-of-clusters-277e93d72d92

[PJR87] – Peter J. Rousseeuw (1987). "Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis". *Computational and Applied Mathematics*. **20**: 53–65. doi:10.1016/0377-0427(87)90125-7

[IDB20] – Indraneel Dutta Baruah, Cheat sheet for implementing 7 methods for selecting the optimal number of clusters in Python, Oct 25 2020, available on https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad

[SV21] – Savy Akholsa, improved by Akanksha_Rai and ruhelaa48, ML | K-means++ Algorithm, last updated on 13 Jul 2021, available on https://www.geeksforgeeks.org/ml-k-means-algorithm/

[BS19] – Bruno Stecanella, Understanding TF-IDF: A Simple Introduction, May 11th 2019, available on https://monkeylearn.com/blog/what-is-tf-idf/

[TI11] – Rajaraman, A.; Ullman, J.D. (2011). "Data Mining" (PDF). *Mining of Massive Datasets*. doi:10.1017/CBO9781139058452.002. ISBN 978-1-139-05845-2.

[AP12] – „About Python”. Python Software Foundation. Archived from the original on 20 April 2012. Retrieved 24 April 2012., second section „Fans of Python use the phrase „batteries included” to describe the standard libray, which covers everything from asynchronous processing to zip files.” available on https://www.python.org/about/

[NP08] – NumPy Documentaion, @Copyright 2008-2022, Numpy Developers. Created using Sphinx 4.2.0, available on https://numpy.org/doc/stable/

[KN22] – Altman, Naomi S. (1992). "An introduction to kernel and nearest-neighbor nonparametric regression" (PDF). The American Statistician. **46** (3): 175–185. doi:10.1080/00031305.1992.10475879. hdl:1813/31637

[KM06] – Berlin, Boris; Sclater, Molly; and Sinclair, Kathryn (2006). *Keys to Music Rudiments*, p.21. Gordon V. Thompson. ISBN 9781457496509. "The key signature, meaning 'a sign which shows the key'..."

[RD20] – Ronak Doshi, Arvind Ram Sankar, Nithin Raj, Song Recommendation System, Feb 20 2020, available on https://github.com/ronak66/Song-Recommendation-System#readme

[YB16] – Yellowbrick : Machine Learning Visualization, @Copyright 2016-2019, The scikit-yb developers, Revision 051a055c, available on https://www.scikit-yb.org/en/latest/

[AA12] – About Anaconda, avaiable on https://web.archive.org/web/20200419034550/https://www.anaconda.com/media-kit/

[C17] – Conda, @Copyright 2017, Anaconda, Inc. Revision b5d2af57 available on https://conda.io/en/latest/

[XZ21] – Xiaomeng Zhang, spotify-dataframe, 2021, available on https://jovian.ai/zhangxm963/spotify-dataframe/v/39/files?filename=spotify-dataset-19212020-160k-tracks/data.csv

[DR22] – Roy, D., Dutta, M. A systematic review and research perspective on recommender systems. *J Big Data* **9**, 59 (2022). https://doi.org/10.1186/s40537-022-00592-5

[BK20] – Buomsoo Kim, Introduction to Matrix Factorization – Collaborative filtering with Python 12, 25 Sep 2020, available on https://buomsoo-kim.github.io/recommender%20systems/2020/09/25/Recommender-systems-collab-filtering-12.md/

[MP21] – multiprocessing – Process-based parallelism, Last updated on Jun 21, 2022, availalbe on https://docs.python.org/3/library/multiprocessing.html

[AR20] – Alok Rawat, The Power of Pandas:Python, Sep 25 2020,available on https://itnext.io/the-power-of-pandas-python-253c11b40b88

[FP11] - Fabian Pedregosa; Gaël Varoquaux; Alexandre Gramfort; Vincent Michel; Bertrand Thirion; Olivier Grisel; Mathieu Blondel; Peter Prettenhofer; Ron Weiss; Vincent Dubourg; Jake Vanderplas; Alexandre Passos; David Cournapeau; Matthieu Perrot; Édouard Duchesnay (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research*. **12**: 2825–2830.

[NSP21] - "NumFOCUS Sponsored Projects". NumFOCUS. Retrieved 2021-10-25.

[JS] - "JetBrains Strikes Python Developers with PyCharm 1.0 IDE". eWeek., available on https://www.eweek.com/c/a/Application-Development/JetBrains-Strikes-Python-Developers-with-PyCharm-10-IDE-304127/

[HE19] - Haagsman, Ernst (4 April 2019). "Collaboration with Anaconda, Inc". *PyCharm Blog*. Retrieved 26 May 2019.

[SB11] – An introduction to seaborn, @ Copyright 2012-2021, Michael Waskom, Created using Sphinx 3.3.1, available on https://seaborn.pydata.org/introduction.html

[MPL02] – Matplotlib API Reference, @Copyright 2002 – 2012, John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team, available on https://matplotlib.org/stable/api/index.html#module-pylab

[SS97] - Stehman, Stephen V. (1997). "Selecting and interpreting measures of thematic classification accuracy". *Remote Sensing of Environment*. **62** (1): 77–89. Bibcode:1997RSEnv..62...77S. doi:10.1016/S0034-4257(97)00083-7.

[PD11] - Powers, David M. W. (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation". *Journal of Machine Learning Technologies*. **2** (1): 37–63. S2CID 55767944

A. SOURCE CODE

Github link: https://github.com/Neri-kun/Licenta

It was integrated in Jupyther Notebook and is structured accordingly in the following parts:

- Data Loading : In this section of the notebook, the datasets are loaded as input by specificying their file location.
- Data Visualisation : In this section, visualizations are made in order to have a better, more general understanding of the nature of data from the datasets.
- Data Preprocessing: In this section, the data is being preprocessed in a definite order of steps, in orderd to be prepared for the recommendation algorithm.
- Recommendation algorithm: In this section, the recommendation takes place and the algorithm used for suggesting songs takes as parameters the number corresponding to the song being played in the dataset and the number of recommendations. Apart from that, results of suggestions for the song which is played are displayed.
- Evaluation of the experimental system.: In this section, it is mentioned the techniques used for evaluating the experimental system.

B. CD / DVD

7 INDEX