**IDM Project**

**Song recommender system**

Ahmad Raza Khawaja 17926

Mustafa Shabbir Karimi 14066

Mohammad Memon 17846

Dataset Link: <https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks?select=data.csv#_ABSTRACT_RENDERER_ID_1424>

**1. Problem Description**:

The dataset file contains more than 170,000 songs collected from Spotify Web API. Spotify is a Digital, cloud-based music platform that provides cross-device access to over 50 million songs, and a rapidly rising number of podcasts and videos. Founded in Stockholm in 2008. It offers a free, ad-supported option and a paid, ad-free version at $9.99 (US) per month.

Spotify pays royalties to musicians (it actually contracts a third party to handle this) based on the number of plays their songs receive. Typically, this is between $0.006 and $0.0084 per play.

The company Swedish, but its HQ is in Luxembourg these days. Listeners can sign up in over 80 countries. However, Spotify is currently not available in Pakistan officially. In April 2020, Spotify reached 133 million premium users. In countries affected by the COVID-19 pandemic, Spotify registered a fall in users in late February, but it has seen a recovery.

There are three recommendation models at work on Spotify:

* Collaborative filtering: Uses your behavior and that of similar users.
* Natural Language Processing (NLP): For song lyrics, playlists, blog posts, social media comments.
* Audio models: Used on raw audio.

But here Spotify recommender system faces an issue of “Cold start”, this constitutes a problem mainly for collaborative filtering algorithms due to the fact that they rely on the user interactions to make recommendations. If no interactions are available, then a pure collaborative algorithm cannot recommend a song to the new user. To solve this problem, we decided to build a content-based filtering system where we will recommend similar songs to the user which he has already listened to base on similarity between the songs. It will somewhat help in solving the cold start issue for Spotify.

We decided to take on this problem using both **Knime and Python code.**

**2. Data Description**:

The dataset includes 170653 rows and 19 columns. The columns or features of the dataset include:

Primary:

* id (Id of track generated by Spotify)

Numerical:

* acousticness (Ranges from 0 to 1)
* danceability (Ranges from 0 to 1)
* energy (Ranges from 0 to 1)
* duration\_ms (Integer typically ranging from 200k to 300k)
* instrumentalness (Ranges from 0 to 1)
* valence (Ranges from 0 to 1)
* popularity (Ranges from 0 to 100)
* tempo (Float typically ranging from 50 to 150)
* liveness (Ranges from 0 to 1)
* loudness (Float typically ranging from -60 to 0)
* speechiness (Ranges from 0 to 1)
* year (Ranges from 1921 to 2020)

Dummy:

* mode (0 = Minor, 1 = Major)
* explicit (0 = No explicit content, 1 = Explicit content)

Categorical:

* key (All keys on octave encoded as values ranging from 0 to 11, starting on C as 0, C# as 1 and so on…)
* artists (List of artists mentioned)
* release\_date (Date of release mostly in yyyy-mm-dd format, however precision of date may vary)
* name (Name of the song)

Here **valence** refers to 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).

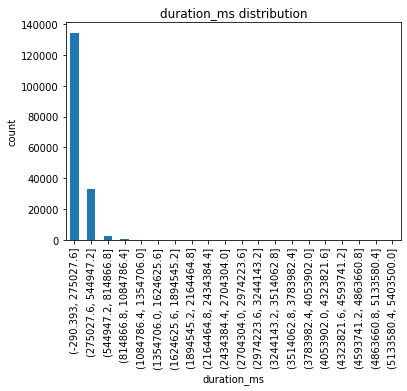
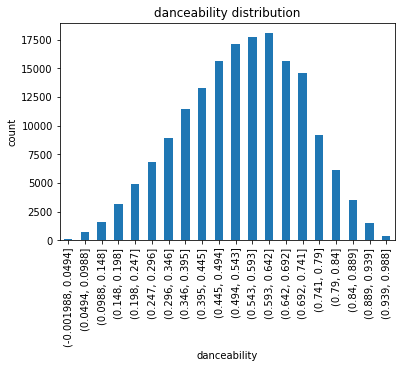
**Liveness** Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live.

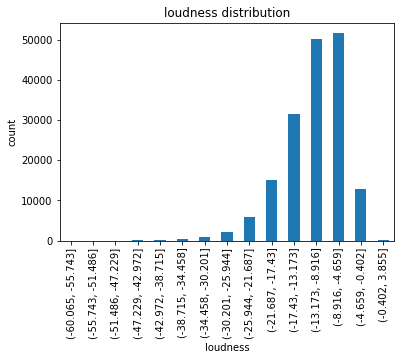
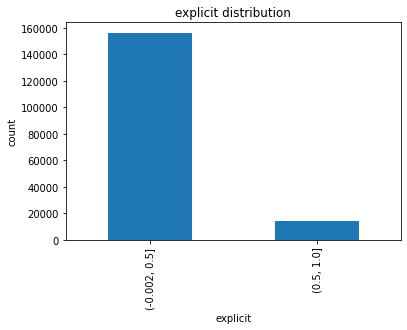
**Instrumental-ness** Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”.

The overall estimated **tempo** of a track is in beats per minute (BPM). tempo is the speed or pace of a given piece and derives directly from the average beat duration.

The estimated overall **key** of the track. Integers map to pitches using standard Pitch Class notation. E.g., 0 = C, 1 = C♯/D♭, 2 = D, and so on.

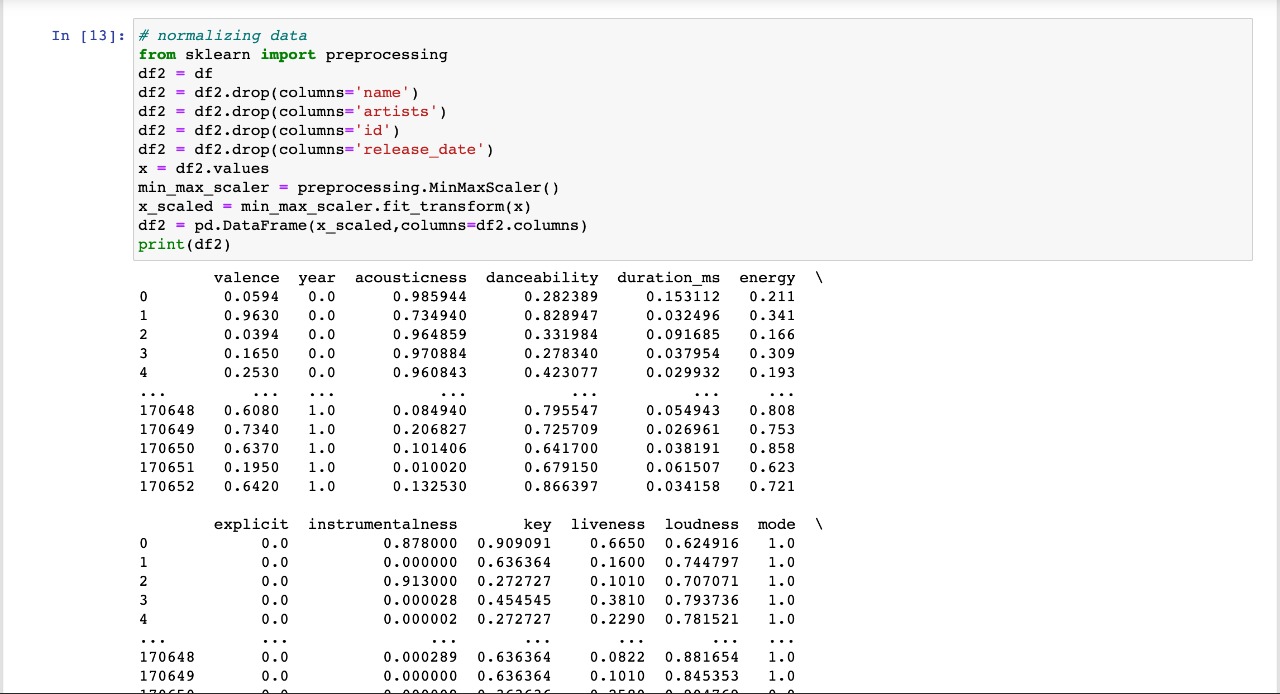
**Mode** Indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.

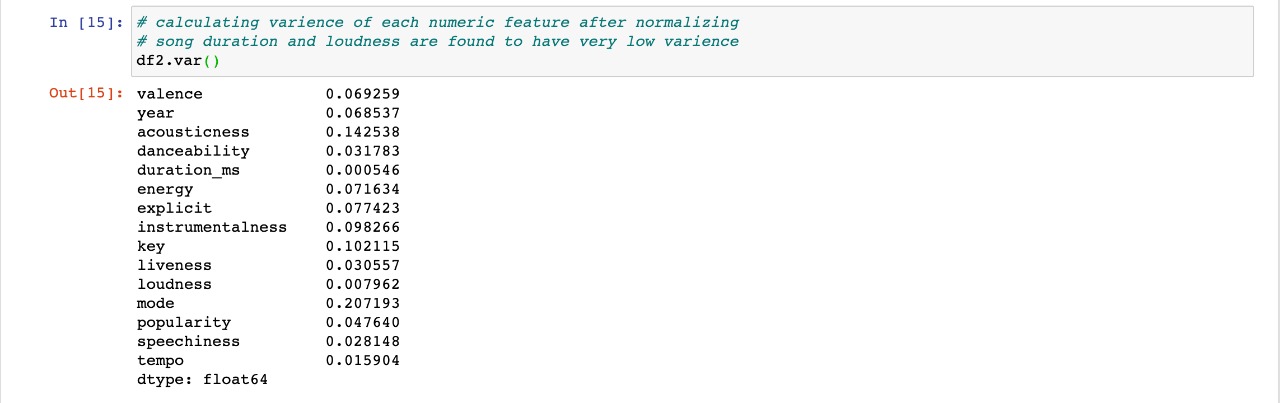


****

We have made some graphs to display the statistical information of some of the labels.

We noticed that only the release\_date column contained missing values, but instead of fixing the values in this column we decided to drop it for our recommender system. We did this because the year column already contained the values for each song and it contained no missing values.

****

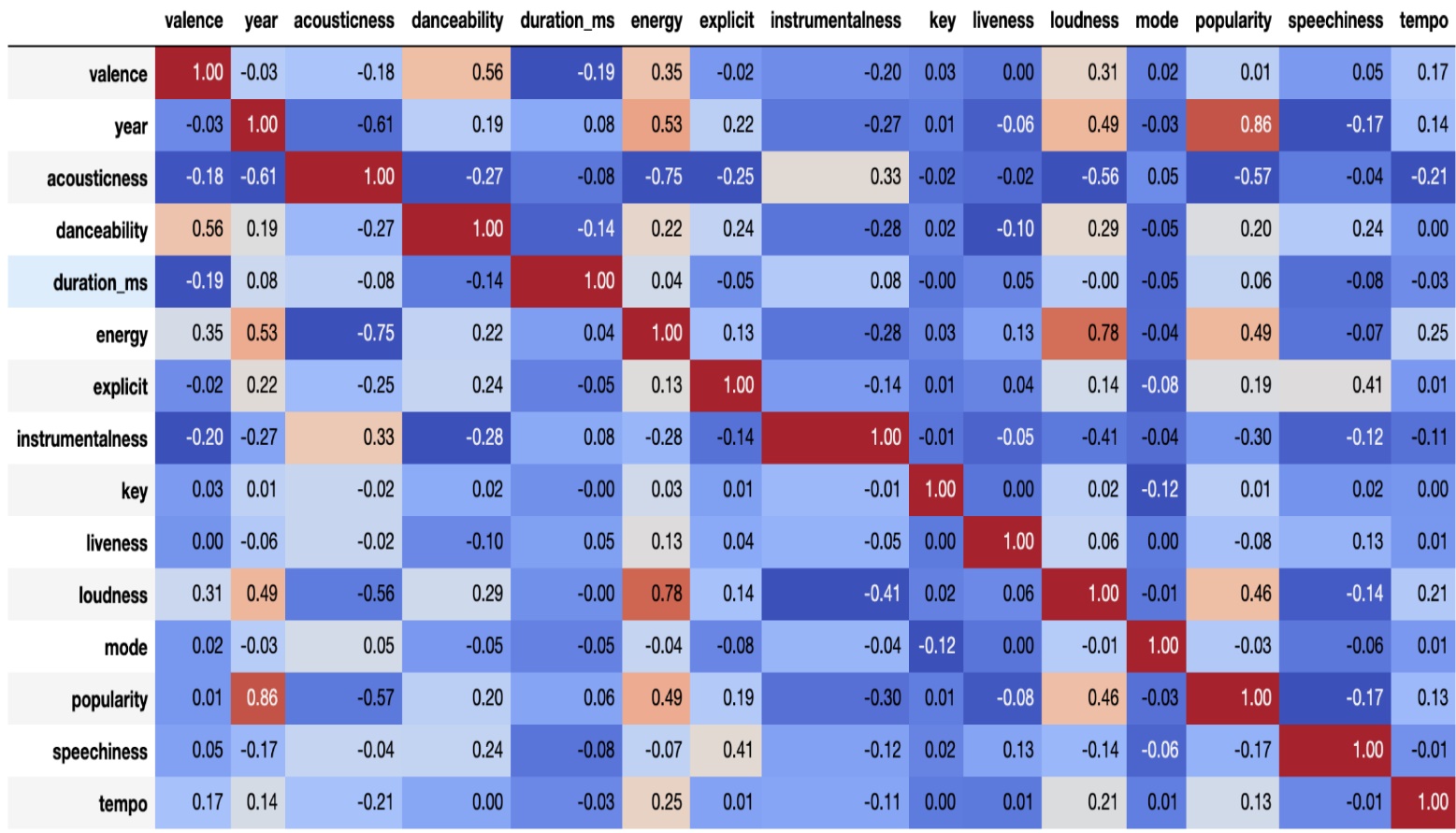
****

The columns, namely duration, loudness and explicit, were removed due to them having very low variance.

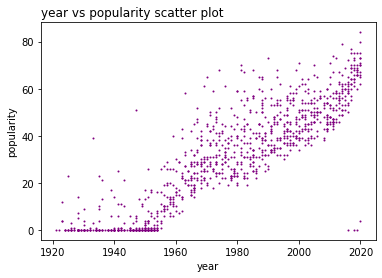
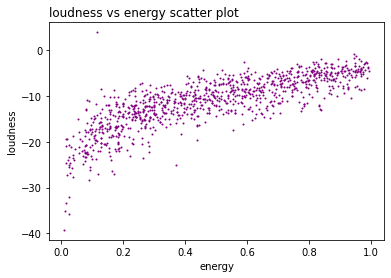
The same thing was done in Knime using low variance filter.

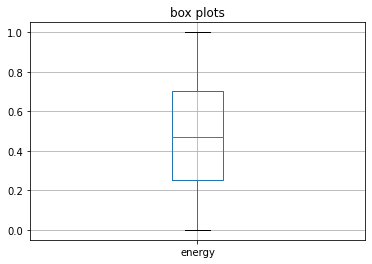
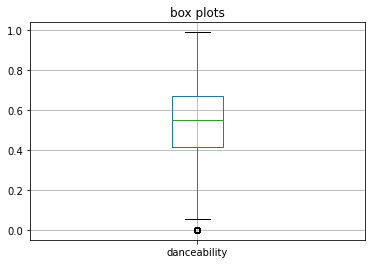
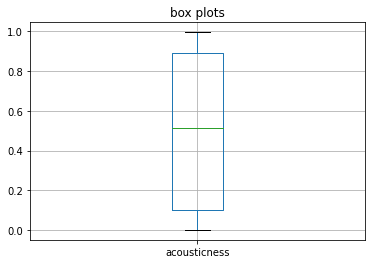
**3. Data pre-processing**:

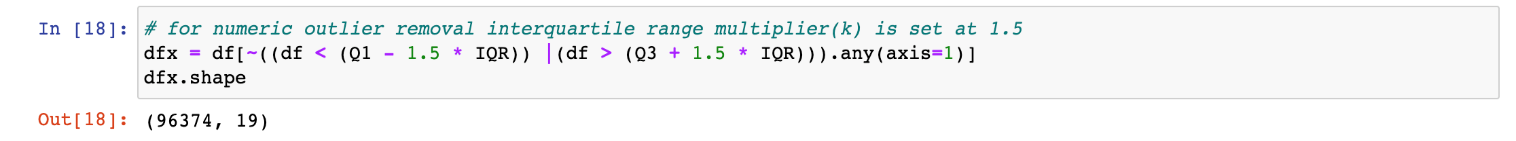
Firstly, we used correlation matrix to identify columns with high correlation. These values were obtained using Pearson Correlation Coefficient.

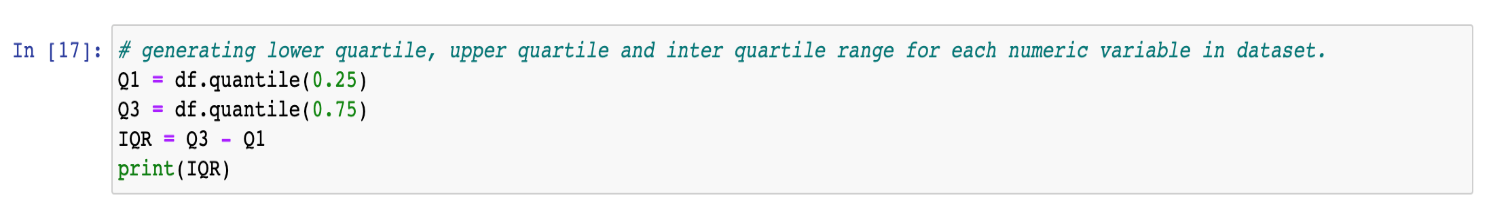


We noticed that the columns “year” and “popularity” had a high correlation value of 0.86. The next pair contained “energy” and “loudness” with a correlation value of 0.78. We made scatterplots to show graphical representation of both these pairs.

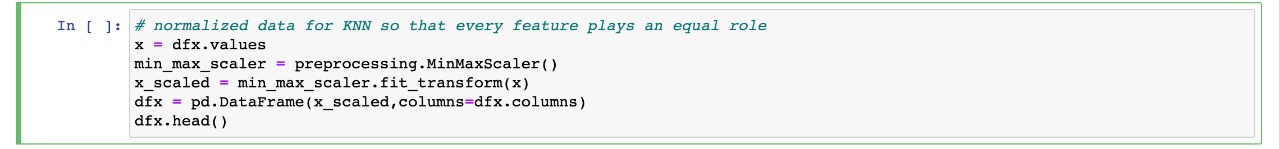
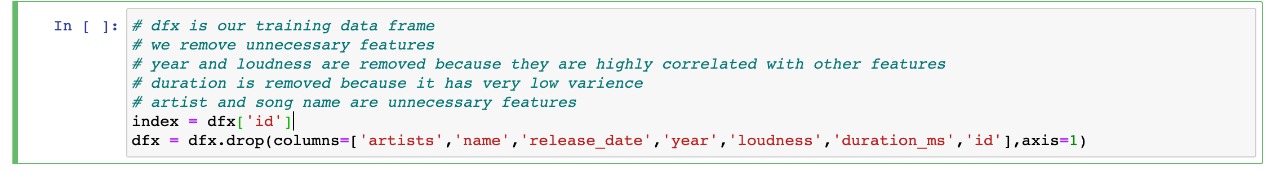


By observing the graphs, we can see that there is a correlation between the features. Next, we removed outliers from our data.

Next, we generated box plots for the columns to get an idea for the numeric outliers.

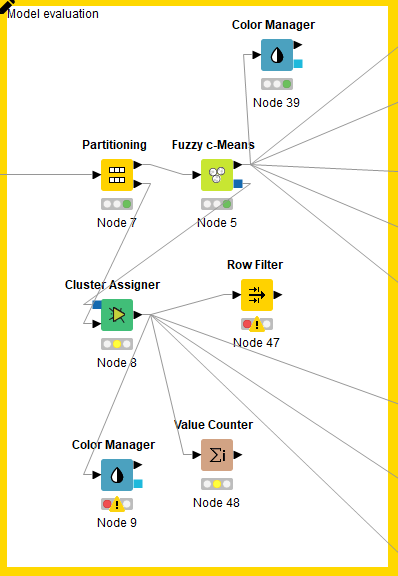


**4. Model Building:**

****

We used KNN algorithm to train our model. We ask the user for input (which is a song’s name), which we use to recommend similar songs based on nearest neighbor. For the code, we used MacBook Pro 2017. This was done in Jupiter Notebook using Pandas, NumPy, sklearn and matplotlib.

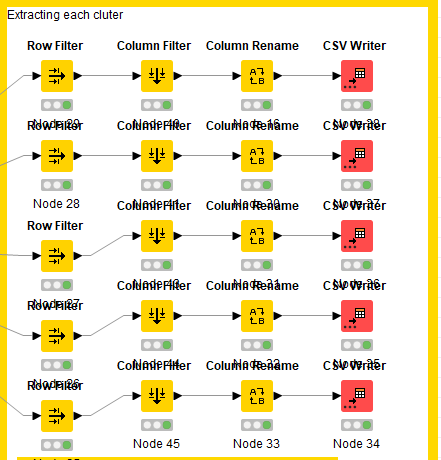
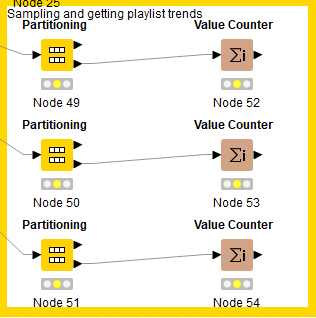
In our Knime approach, we used the Fuzzy C-mean model. The best part of Fuzzy C is that it tells you about the winner cluster and also about the other clusters and their membership in every other cluster. To change things up, out of 170,000 records, we used 160,000 for clustering and the remaining to check how well the assignment has worked in case new records are added. For our Knime approach, we used a PC with i3-10th Gen Processor, Nvidia GTX 1650 and 8 GB RAM.



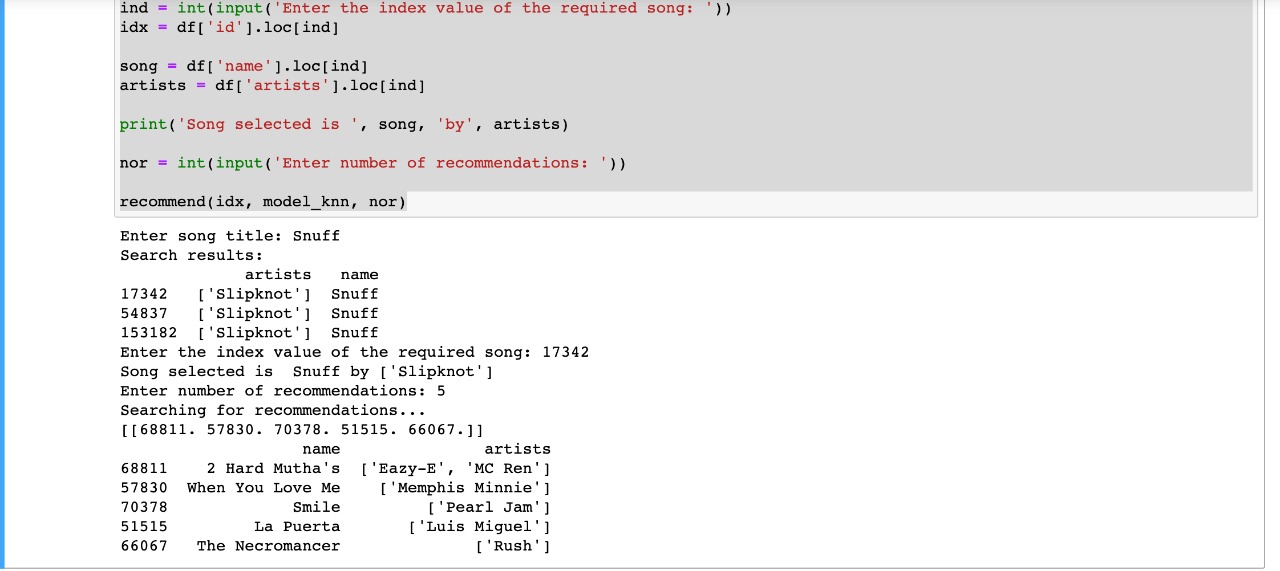
**5. Evaluation:**

In our KNN model, we specify the artist/song that the user chooses and based on that song we get recommendations. We can limit the number of recommendations we receive. The most important thing that our recommendation system does is that it does not need a minimum specification of songs to recommend new ones. Hence this solves our problem of “cold start”. There was a bit of confusion on how to evaluate our KNN recommendation system because here we don't have any kind of test data or class labels on basis of which we can say that our recommendation is correct, and also, we have no explicit method of telling whether the recommendations are precise or not.

In our Knime model, each individual file has been compiled based on individual clusters. Based on the majority of the songs in the cluster, a playlist has been created to be recommended to each user. Some of the examples of the playlists are Indie/Rock, Heavy Metal, Hip Hop and Old Classical. We have also created customized user playlist by sampling songs that were not used for cluster creation. When dealing with a playlist instead of a song, we first identify how many songs of each cluster exist in the playlist. Based on this, we see the trends and then recommend songs to the person based on their taste from each of our cluster’s master-file.



**6. Findings:**



Based on the song that the user enters, our recommendation system recommends similar songs (as shown in the image above). The limitation of this system is that we cannot perform any scoring as there are no labels for comparison.

On our Knime model, we also included the ability to recommend songs based on a playlist. Each playlist is examined by identifying which cluster each song belongs to and the songs from most occurring cluster is recommended to the user.

We would suggest adding some more categories to the data. The categorization may be based on the genres of the songs (e.g., hip hop, rock etc.). If the songs would already be assigned genres, we can score our model on the metrics of how many songs of each genre are in our cluster. We would suggest Spotify to use our model for newcomers. They may use our model for recommending songs until they have enough songs to continue using their own, present, system. This allows for a hybrid system where they get to keep their own system while also avoiding the cold start problem.

One more thing that we would like to suggest Spotify is to monitor the number of recommendation that the user responds positively to. By using this metric, we can identify the shortcomings in our algorithm and the success rate with actual numbers. By identifying the number of songs, we positively recommend, we will have a much better understanding on where we need to work on our algorithm to achieve better numbers. A good approach would be to add a like or dislike button with each recommendation.

These models are ready to be used as a sub-application in any of the major music applications like Apple Music and Pandora.

The workflow can be broken down into following basic steps:

* Training a machine learning model on a local system.
* Wrapping the inference logic into a flask application.
* Using docker to containerize the flask application.
* Hosting the docker container on an AWS ec2 instance and consuming the web-service.

Our model will expire when:

* (Knime) The clusters we have made are based on danceability, loudness (etc.). In the future if we wish to make clusters for new features, we would have to make clusters for every feature from scratch.
* (Code) When new data is entered then the model always calculates the nearest neighbors from scratch. If we decide to change the perimeters of weight calculation or change the algorithm then in that case our model will expire.