SONG RECOMMENDER USING K MEANS CLUSTERING

DSA PROJECT - DATA SCIENCE ANALYSIS- M VIBHUVAN

Project Background: Many users in Spotify listen to various songs which they like to hear, but there are songs which they may like but aren't aware of, our project attempts to solve this issue by recommending songs to the user upon his/her liking of previous songs through K Means clustering algorithm.

Project Overview: We have taken a Spotify dataset, which consists of songs over a century, with various features pertaining to the song, we have done Data cleaning, EDA to get insights about the data, and then we performed the most important step, i.e, applying K Means clustering to the dataset to group similar songs into clusters, now given a person's favorite song, we used Recommendation system concept using cosine distance to recommend songs from the corresponding cluster of that given song.

DETAILED DESCRIPTION:

1) Understanding the data: The data consists of total 170653 entries with a total of 19 columns/features which are as follows-

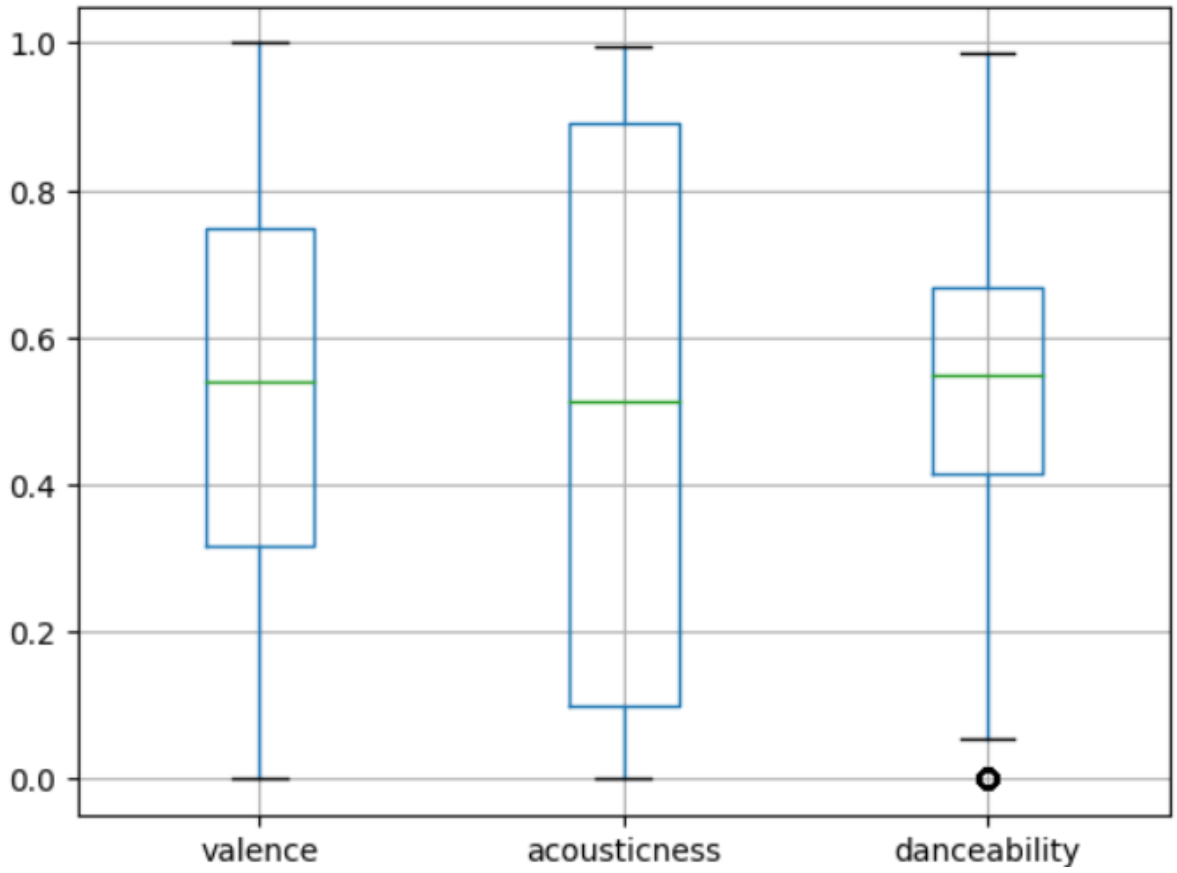
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   valence           170653 non-null  float64
 1   year              170653 non-null  int64  
 2   acousticness      170653 non-null  float64
 3   artists           170653 non-null  object 
 4   danceability      170653 non-null  float64
 5   duration_ms       170653 non-null  int64  
 6   energy            170653 non-null  float64
 7   explicit          170653 non-null  int64  
 8   id                170653 non-null  object 
 9   instrumentalness  170653 non-null  float64
 10  key               170653 non-null  int64  
 11  liveness          170653 non-null  float64
 12  loudness          170653 non-null  float64
 13  mode              170653 non-null  int64  
 14  name              170653 non-null  object 
 15  popularity        170653 non-null  int64  
 16  release_date      170653 non-null  object 
 17  speechiness       170653 non-null  float64
 18  tempo             170653 non-null  float64
```

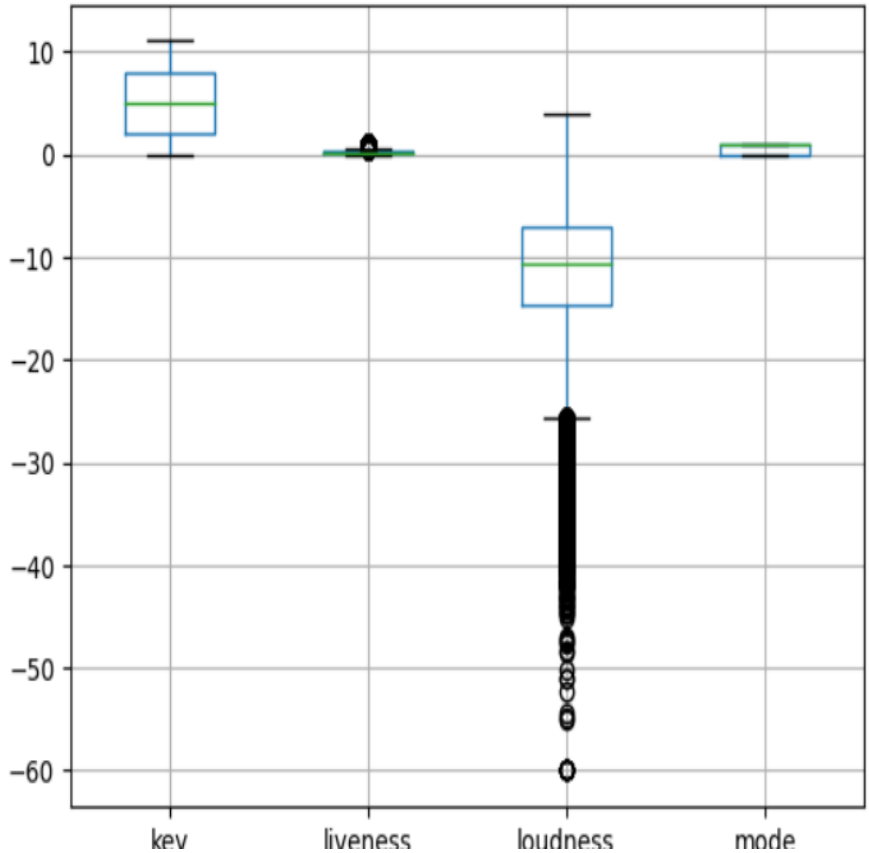
2) Data Cleaning: Since we are only interested in numerical parameters, we have dropped the year, artists, id, release_date columns, furthermore we checked if they are any null values, but there aren't any -

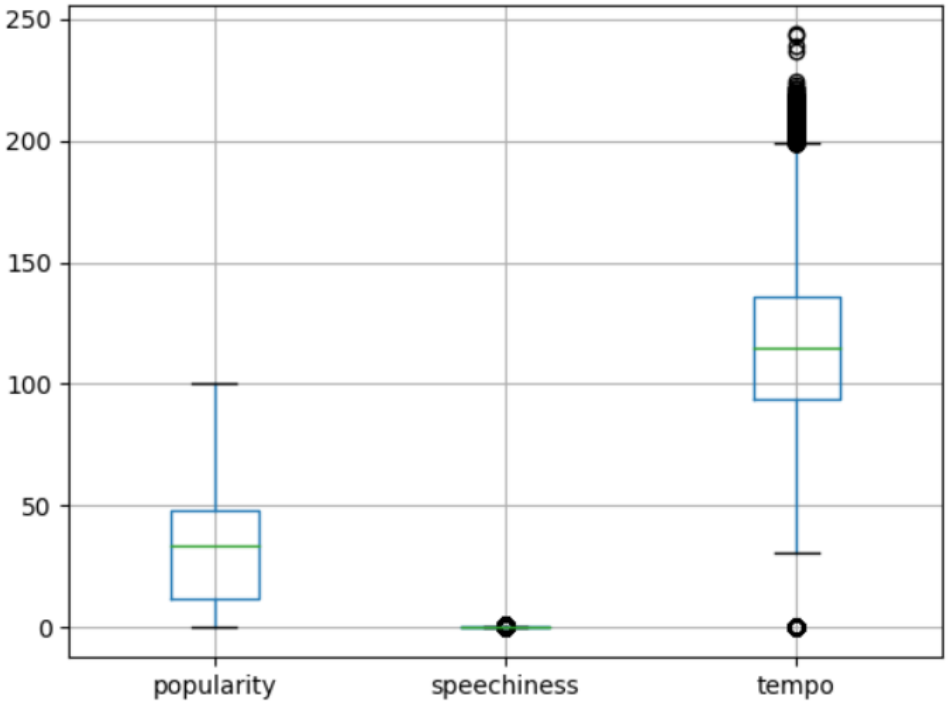
```
df.isnull().sum()/len(df)
#There are no null values
```

```
valence             0.0
acousticness        0.0
danceability        0.0
duration_ms         0.0
energy              0.0
explicit            0.0
instrumentalness    0.0
key                 0.0
liveness            0.0
loudness            0.0
mode                0.0
name                0.0
popularity          0.0
speechiness         0.0
tempo               0.0
dtype: float64
```

We then checked if there are any duplicate values, there were 806 duplicates, we have removed those too. To check outliers, we have plotted box plots of all the features:





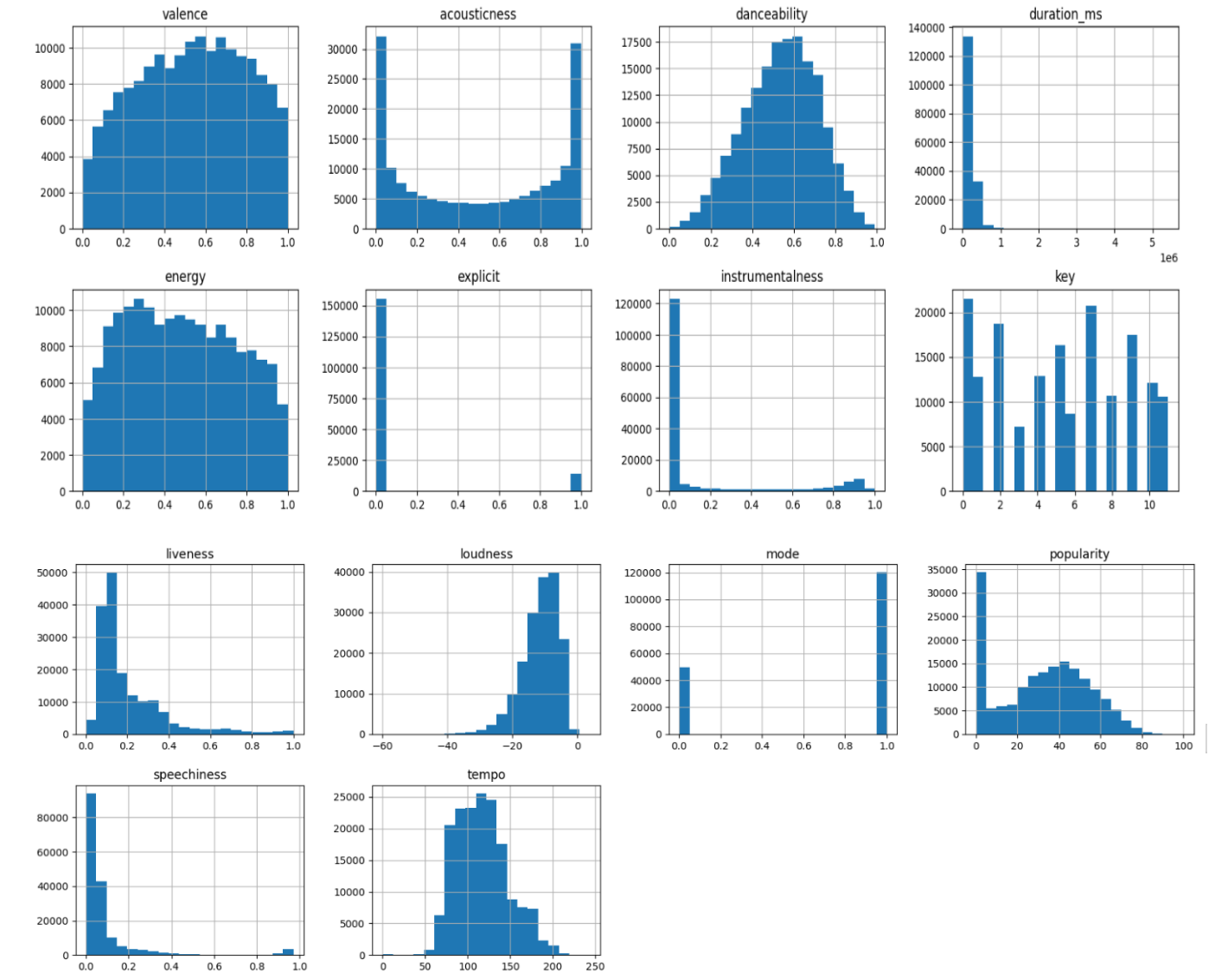


According to the box plots, we have removed the outliers accordingly.

3) EDA:

The distribution plots of each feature are as follows-

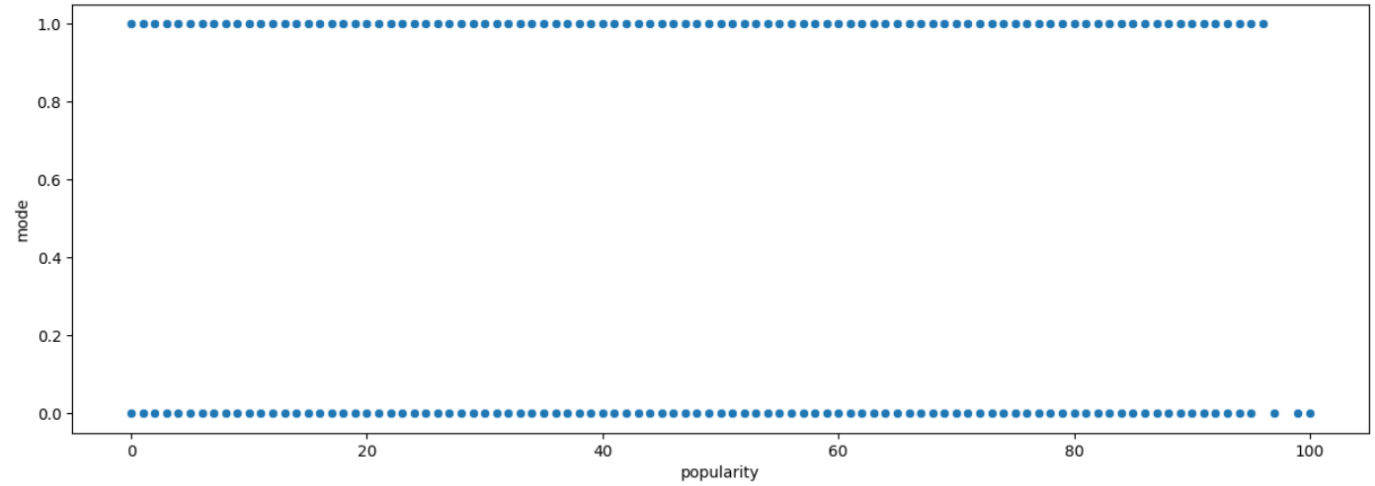
```
df.hist(bins=20, figsize=(20,15))
plt.show()
```



So, as we can see valence and energy follow uniform distribution, while danceability, loudness, tempo follow approximately normal distribution.

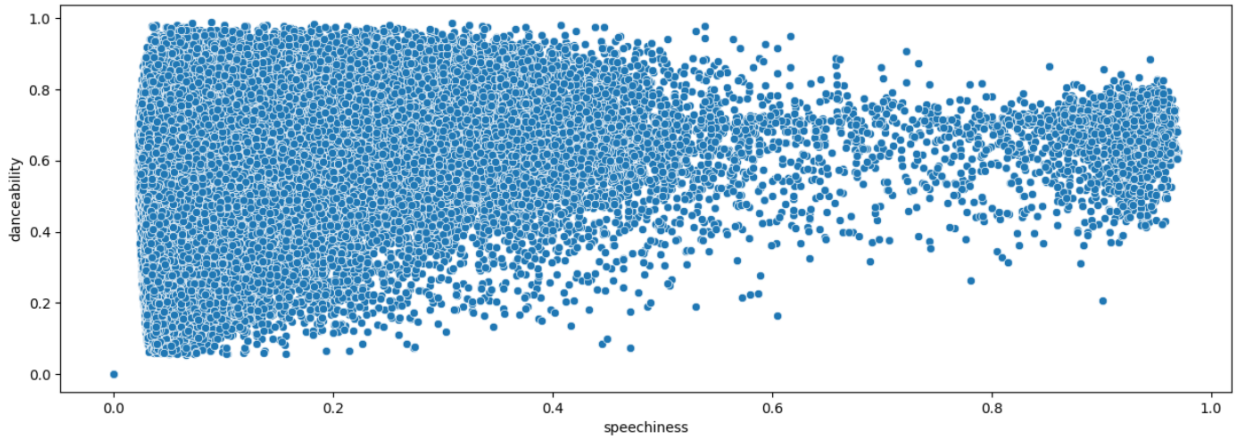
Scatter plots of popularity vs mode and danceability vs speechiness are as follows-

```
plt.figure(figsize=(15,5))
sns.scatterplot(x="popularity", y="mode", data=df)
plt.show()
# Woah, mode is equally distributed
```



We can see mode is equally distributed with popularity

```
plt.figure(figsize=(15,5))
sns.scatterplot(x="speechiness", y="danceability", data=df)
plt.show()
# Sing less, dance more
```

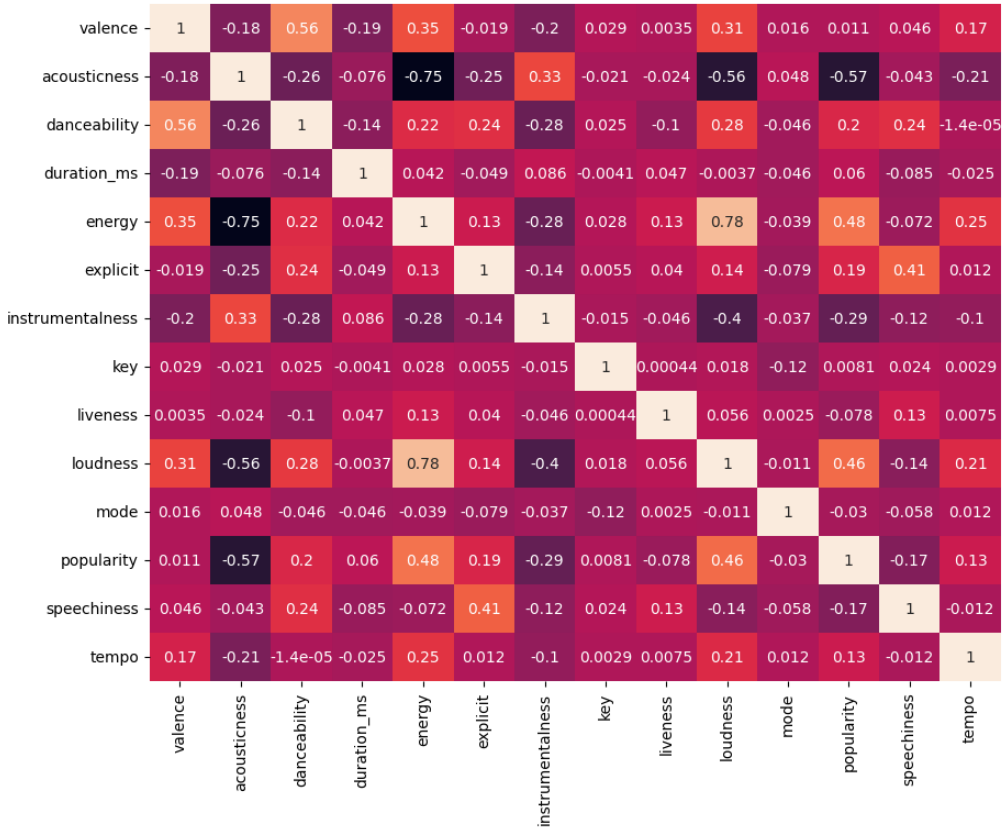


So more the speechiness, less is the danceability

4) Data Preparation:

A correlation heatmap is a commonly used tool in data analysis that allows users to visualize the correlation coefficients between pairs of variables in a dataset. The heatmap is created by calculating the correlation coefficients between each pair of variables and plotting these values as a color-coded matrix.

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),annot=True,cbar=False)
plt.show()
# Energy and acousticness are very negatively correlated, since many features are correlated with each other, we'll use PCA
```



It can help users identify patterns or clusters within the dataset, making it a valuable tool for exploratory data analysis. It is particularly useful for large datasets, as it allows users to quickly identify correlations without having to manually examine every variable. Additionally, the color-coded matrix makes it easy to interpret the data and identify potential areas for further analysis. Since, there are correlations between some features, it would be wise to use PCA, but before that we have done Standard Scaling.

Standard scaling, also known as standardization or z-score normalization, is a technique used in data preprocessing to transform variables to have a mean of zero and a standard deviation of one. This technique is widely used in machine learning and data analysis to normalize data and improve the performance of various algorithms.
The standard scaling process involves subtracting the mean value of the variable from each data point and then dividing by the standard deviation of the variable. This creates a new

variable with a mean of zero and a standard deviation of one. The formula for standard scaling is:

z = (x - mean) / std

where z is the standardized value, x is the original value, mean is the mean value of the variable, and std is the standard deviation of the variable.
One of the main benefits of standard scaling is that it ensures that variables with different scales and units are comparable. When variables have different scales, algorithms that rely on distance metrics like K Means clustering or gradient descent optimization can be affected.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_std = pd.DataFrame(scaler.fit_transform(df.select_dtypes(np.number)))
df_std.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	-1.786434	1.282366	-1.470805	4.767582	-1.018347	-0.304639	2.283275	1.365643	2.624533	-1.523798	0.643667	-1.265964	-0.380149	-1.171062
1	1.649865	0.616762	1.596809	-0.400567	-0.532267	-0.304639	-0.529996	0.512170	-0.262501	-0.175700	0.643667	-1.220038	1.940496	-1.823109
2	-1.862492	1.226456	-1.192448	2.135583	-1.186606	-0.304639	2.395421	-0.625794	-0.599798	-0.599941	0.643667	-1.220038	-0.396708	-0.213902
3	-1.384848	1.242430	-1.493528	-0.166683	-0.651917	-0.304639	-0.529907	-0.056812	1.000934	0.374633	0.643667	-1.311891	-0.387509	-0.547125
4	-1.050193	1.215806	-0.681179	-0.510417	-1.085651	-0.304639	-0.529990	-0.625794	0.131965	0.237270	0.643667	-1.357817	-0.371563	-0.496441

PCA (Principal Component Analysis):

Now we have performed PCA to reduce the dimensionality and redundancy in correlations amongst features. It is a statistical technique that is widely used for dimensionality reduction and feature extraction. The basic idea of PCA is to transform a high-dimensional dataset into a lower-dimensional space while retaining the most important information. PCA works by finding the principal components of the dataset. These components are linear combinations of the original variables that capture the maximum amount of variance in the data. The first principal component explains the most variance, the second principal component explains the second-most variance, and so on. The principal components are orthogonal to each other, meaning that they are independent and non-redundant.
First, the covariance matrix of the data is calculated. The covariance matrix shows the relationships between each pair of variables in the dataset. The diagonal elements of the covariance matrix are the variances of each variable, while the off-diagonal elements represent the covariances between pairs of variables.
Next, the eigenvectors and eigenvalues of the covariance matrix are calculated. The eigenvectors are the directions of maximum variance in the data, while the eigenvalues represent the amount of variance explained by each eigenvector. The eigenvectors are sorted by their corresponding eigenvalues, with the highest eigenvalue corresponding to the first principal component.
Finally, the data is transformed into the new coordinate system defined by the eigenvectors. Each data point is projected onto the eigenvectors to obtain its principal component scores.

The number of principal components used in the analysis can be selected based on the amount of variance that they explain. Typically, the first few principal components that explain most of the variance in the data are retained, while the rest are discarded, in our case we have selected 9 components according to the variance scores.

```
from sklearn.decomposition import PCA
import numpy as np

# assuming X is your standardized dataset
# create a PCA instance with desired number of components
pca = PCA(n_components=9)

# fit the PCA instance on X


# transform X to the new coordinate system
data_pca = pd.DataFrame(pca.fit_transform(df_std))

# print the explained variance ratios for each component
print(pca.explained_variance_ratio_)
```

```
[0.24827604 0.12031908 0.09785871 0.08041057 0.07798829 0.06693632
 0.06512211 0.05990392 0.05358838]
```

```
] data_pca.head()
```

	0	1	2	3	4	5	6	7	8
0	3.933814	2.136420	3.030339	0.780624	2.210729	2.369602	0.399567	2.946328	-0.601865
1	0.335127	-3.004669	-1.150453	-0.157795	0.004288	1.766469	-0.484535	0.851947	0.093403
2	3.448474	1.667464	0.989997	0.141154	-0.523394	0.088042	1.227853	1.397276	-0.770671
3	1.876500	0.671853	0.371296	1.346314	0.334716	0.303551	-1.235009	-0.601500	0.593276
4	1.853034	0.062713	-0.364621	0.957422	-0.503504	0.165130	-0.715372	-0.590724	0.712427

Performing PCA before K-means clustering can improve the quality of the clustering results by reducing the influence of irrelevant features and focusing on the most important ones. This can lead to better separation between clusters and more accurate assignments of data points to the appropriate cluster.

5) K Means Clustering:

Elbow Method:

The elbow method is a popular technique used in K-means clustering to determine the optimal number of clusters to use for a given dataset.

The method works by calculating the sum of squared distances between each point in a dataset and the centroid of its assigned cluster, for different values of K. The sum of squared distances is also known as the "inertia" of the clustering solution.

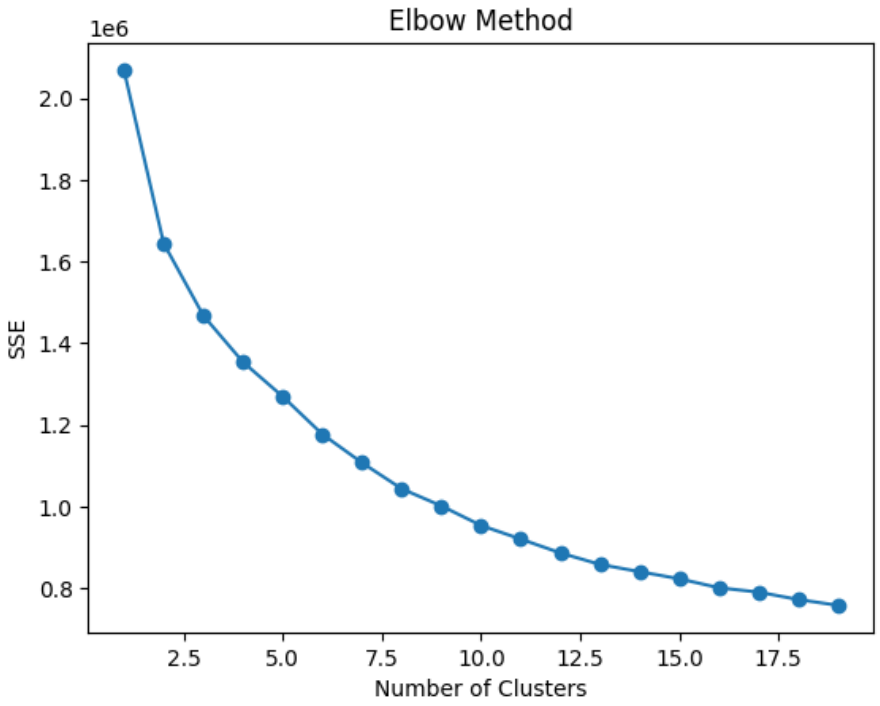
To apply the elbow method, we start by running K-means clustering on the dataset for a range of values of K, typically from 1 to 10 or 20. For each value of K, we calculate the sum of squared distances (inertia) of the resulting clusters.
We then plot the values of inertia against the number of clusters K. The resulting plot should resemble an "elbow" shape, where the x-axis represents the number of clusters K and the y-axis represents the corresponding value of inertia.
The "elbow" in the plot represents the optimal number of clusters for the dataset. This is typically the point where the rate of decrease in inertia slows down significantly, indicating that adding more clusters does not significantly improve the quality of the clustering solution.
In our case, we have run number of clusters from 1 to 20, the plot is as follows-

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt


# initialize lists to store the results
wcss = []

# iterate over the cluster range
for i in range(1,20):
    # create a KMeans instance with n_clusters
    Kmeans = KMeans(n_clusters=i, n_init=10, random_state=42)
    # fit the KMeans instance on X
    Kmeans.fit(data_pca)
    wcss.append(Kmeans.inertia_)

# plot the elbow curve
plt.plot(range(1,20), wcss, 'o-')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
plt.title('Elbow Method')
plt.show()
```



So, as we can clearly see, the elbow is forming at 6, hence we chose the optimal number of clusters to be 6. Now, coming to the most important part,

K Means Clustering algorithm:

K-means is a clustering algorithm that partitions data points into k clusters based on their similarity.The k-means algorithm is a type of centroid-based clustering technique that aims to partition a set of n data points into k clusters. The algorithm works as follows:

1) Initialization: The algorithm starts by selecting k random data points as the initial centroids.
2) Assignment: Each data point is assigned to the nearest centroid based on its distance using a distance metric such as Euclidean distance, Manhattan distance, or cosine distance. This step creates k clusters.
3) Recalculation: The centroids of each cluster are recalculated by taking the mean of all the data points assigned to that cluster.
4) Reassignment: Each data point is reassigned to the nearest centroid based on the updated centroid locations.
5) Repeat: Steps 3 and 4 are repeated until the centroids no longer move significantly or a maximum number of iterations is reached.
6) Output: The final output of the algorithm is a set of k clusters, where each cluster is represented by its centroid.
The k-means algorithm aims to minimize the sum of the squared distances between each data point and its assigned centroid. This objective function is known as the within-cluster sum of squares (WCSS), and it measures the quality of the clustering. The algorithm tries to find the optimal partition of the data that minimizes the WCSS.
The k-means algorithm has a few important parameters that need to be specified, such as the number of clusters k and the distance metric used. The value of k is typically determined by a trial-and-error process or domain knowledge, while the distance metric is chosen based on the characteristics of the data.

Here, we have fit the data,

```
k_means = KMeans(n_clusters=6, verbose=False)
k_means.fit(data_pca)
labels = k_means.predict(data_pca)
df['cluster_label'] = labels
```

Now, we are going to visualize the clusters through the first 2 principal components,

```
pca = PCA(n_components=2)
song_embedding = pca.fit_transform(df_std)
projection = pd.DataFrame(columns=['x', 'y'], data=song_embedding)
projection['title'] = df['name']
projection['cluster'] = df['cluster_label']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'title'])
fig.show()
```

Silhouette score:

The silhouette score is a metric used to evaluate the quality of a clustering solution by measuring how similar a data point is to its own cluster compared to other clusters. The score ranges from -1 to 1, with a higher score indicating better clustering quality.

```
print("Silhouette score for number of clusters = 6 is", silhouette_score(data_pca, k_means.labels_))
```

```
Silhouette score for number of clusters = 6 is 0.15437082072716413
```

This is a pretty good Silhouette score indicating that our clustering worked well.

6) Song Recommendation system:

Recommendation systems are widely used in various domains, such as e-commerce, movie streaming, and social media. These systems aim to suggest items to users that they might be interested in based on their preferences and behavior. One popular method for making such recommendations is using cosine distance.

Cosine distance is a measure of similarity between two vectors in a high-dimensional space. In recommendation systems, each user and item is represented as a vector of features. For example, in a song recommendation system, the features are song mode, tempo and so on arranged in a form of vector. The cosine distance between two vectors is calculated as the cosine of the angle between them.

To make recommendations using cosine distance, we first compute the cosine similarity between the vectors representing the user and all items. The cosine similarity between two vectors is defined as the dot product of the two vectors divided by the product of their magnitudes. The dot product of two vectors is calculated by multiplying each corresponding element of the vectors and summing the results. The magnitude of a vector is the square root of the sum of the squares of its elements.

Once we have calculated the cosine similarity between the user vector and all item vectors, we can sort the items by their similarity scores and recommend the top N items to the user. This is known as a nearest neighbor algorithm, where we find the N items that are closest to the user vector in terms of cosine similarity.

The formula for calculating the cosine similarity between two vectors A and B is:

cosine_similarity(A, B) = dot_product(A, B) / (magnitude(A) * magnitude(B))

where dot_product(A, B) refers to the dot product between vectors A and B, and magnitude(A) and magnitude(B) refer to the magnitudes of vectors A and B, respectively.

```
def recommend_song(ind, df, no_of_songs):
    df_input_train = df.loc[[ind]].reset_index(drop=True)
    df_important = df[df['cluster_label'] == df.loc[ind, 'cluster_label']].reset_index(drop=True)
    scaler = StandardScaler()
    df_input_new = df_input_train.loc[:, df_input_train.columns != 'cluster_label']
    df_important_new = df_important.loc[:, df_important.columns != 'cluster_label']
    df_input_std = pd.DataFrame(scaler.fit_transform(df_input_new.select_dtypes(np.number)))
    df_important_std = pd.DataFrame(scaler.fit_transform(df_important_new.select_dtypes(np.number)))
    distances = cdist(df_input_std, df_important_std, 'cosine')
    index = list(np.argsort(distances)[:, :no_of_songs][0])
    recommended_songs = df_important.iloc[index]
    return recommended_songs
```

So, for our recommender, the index of the song from our dataset liked by the user is given as input along with the number of songs he/she wants to be recommended and the dataset. Then, from that given index, the appropriate cluster label is found out since we have stored that cluster label data in the dataset and the cosine distance nearest neighbor search is done in that specific cluster since similar songs to the input song have been grouped in that specific cluster. The nearest 5 distances are found out between our input song feature vector and the nearby songs feature vectors from that specific cluster and then they are arranged in an ascending order so as to give top priority to closest songs, so we are making use of the clusters formed by our K Means algorithm, where our entrie's cluster label is found out and used for the search according to the cosine distance rule.

RESULTS:

Input Song data:

```
print(df.iloc[5])
```

```
valence                    0.196
acousticness               0.579
danceability               0.697
duration_ms               395076
energy                     0.346
explicit                       0
instrumentalness           0.168
key                            2
liveness                    0.13
loudness                 -12.506
mode                           1
name               Gati Mardika
popularity                     6
speechiness                 0.07
tempo                    119.824
cluster_label                  0
Name: 5, dtype: object
```

6.1) Recommendening the data of best 5 songs you would like to hear

```
[262] df_test = recommend_song(5, df, 5)
      print(df_test)
```

	valence	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	name	popularity	speechiness	tempo	cluster_label
0	0.165	0.967	0.275	210000	0.3090	0	0.000028	5	0.381	-9.316	1	Danny Boy	3	0.0354	100.109	0
21709	0.318	0.953	0.471	137093	0.0657	0	0.096400	0	0.124	-19.594	0	The Side of a Hill	23	0.0344	112.322	0
21708	0.425	0.683	0.416	154667	0.2300	0	0.000000	4	0.220	-15.853	1	The Great Pretender	26	0.0292	105.624	0
21707	0.186	0.861	0.255	158893	0.2340	0	0.020000	2	0.113	-13.506	1	Mr. Lonely	18	0.0298	172.227	0
21706	0.434	0.966	0.560	281253	0.0514	0	0.008730	3	0.116	-25.435	1	Hi-Lili, Hi-Lo	21	0.0414	140.117	0

6.2) Names of those 5 songs

```
[263] print(df_test['name'])
```

```
0                  Danny Boy
21709     The Side of a Hill
21708    The Great Pretender
21707              Mr. Lonely
21706         Hi-Lili, Hi-Lo
Name: name, dtype: object
```

These are the best 5 songs recommended when the recommender received input of index 5. The index 5 song which is Gati Mardika is so close to all these 5 songs distance wise, hence our Recommendation system worked.

CONCLUSIONS:

1) Data Preprocessing has been done so as to ensure the correct form of data is going as input to our algorithm. Standard Scaling along with PCA have been performed to reduce the dimensionality.
2) From the Elbow method, we found out that the optimal number of clusters are 6, we have fit our K Means algorithm to our data with K=6 and we stored the labels in our dataset
3) Then, we have built our recommendation system which takes input as an index, finds its appropriate K Means cluster and then does a cosine distance nearest neighbor search to give the closest songs possible.

The code of our project is attached, please make use of this system to your best and **“HAPPY LISTENING”**