Preliminary project
Project 20: Song recommender

Group 16:
Seniv Volodymyr 309358
Oleg Kim 309025

Task description:

Develop a song recommender system from datasets of song attributes. Students may choose one or a combination of datasets from the references below. Feel free to choose any approach to develop a solution, however, it is encouraged to use more than one method for comparing performance.

The goal of this project is as follows:

-Perform exploratory data analysis (EDA) on the dataset to understand data distribution, statistical significance of each feature and observe trends. Develop hypotheses and reasoning that can help when building a model.

-Perform data preprocessing to prepare the data before feeding into the model, e.g.: feature cleaning, selection and rescale.-Split the dataset to create separate training and test datasets.

-Train and compare regression models (may use ML libraries such as Scikit-Learn). Evaluate the model's performance metric and assess the impact of preprocessing strategy (e.g., contribution of features).

Theoretical background:

Clustering model is the main concept around which are solution is based.

Changes from initial plan:

Initially we have chosen content-based filtering and matrix factorization to recommend songs to users. We concluded that it would be more effective to use clustering algorithms for the purpose of the task.

Clustering Model:

A clustering model is a type of unsupervised learning algorithm that aims to group similar data points together into clusters based on their inherent patterns or similarities. It does not rely on predefined labels or target variables but rather identifies patterns and structures in the data itself.

Algorithms:

K-means clustering algorithm – K-means clustering is an iterative algorithm used to partition a dataset into K clusters. It works by iteratively assigning data points to the nearest centroid and updating the centroids based on the mean of the assigned points. The algorithm requires specifying the number of clusters, K, in advance. It aims to minimize the sum of squared distances between data points and their assigned centroids. K-means clustering is computationally efficient and widely used for its simplicity and effectiveness in finding spherical-shaped clusters.

Hierarchical clustering algorithm – Hierarchical clustering, on the other hand, is an algorithm that creates a hierarchy of clusters. It starts with each data point as an individual cluster and gradually merges or splits clusters based on their similarity. The algorithm does not require specifying the number of clusters in advance and provides a dendrogram, a tree-like structure that visually represents the clustering hierarchy. Hierarchical clustering can be agglomerative (bottom-up) or divisive (top-down) in nature. Agglomerative clustering, which is commonly used, starts with individual data points as clusters and progressively merges the most similar clusters until all data points belong to a single cluster. Hierarchical clustering is advantageous when the underlying

structure of the data is not well-defined or when exploring different levels of granularity in clustering.

Dataset Analysis

In the dataset we are provided with 16 columns where we have 13 of which are song attributes, 1 is a song name, 1 is for artist, and a “target” column which represents the preference of the author of the dataset, where “1” is preferred and “0” not preferred.

Continuous features: 'acousticness', 'danceability', 'duration_ms', 'energy', 'liveness', 'loudness', 'tempo', 'valence', 'speechiness', 'instrumentalness'

Discrete features: 'key','mode','time_signature','target'

Here are the 13 track attributes:

Acousticness - A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic. (>= 0 ,<= 1)

Danceability - Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.

Duration_ms - The duration of the track in milliseconds.

Energy - Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

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 contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.

Key - The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1. (>= -1 , <= 11)

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. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.

Mode - 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.

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 to 1.0 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.

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.

Time_signature – An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4". (>= 3, <= 7)

Valence - A measure from 0.0 to 1.0 describing 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). (>= 0, <= 1)

Mean values:

valence mean: 0.496815022

target mean: 0.505701537

tempo mean: 121.6032717

speechness mean: 0.092664254

loudness mean: -7.085624194

liveness mean: 0.190844026

intrumentalness mean: 0.133285529

energy mean: 0.681577144

duration_ms: 246306.1973

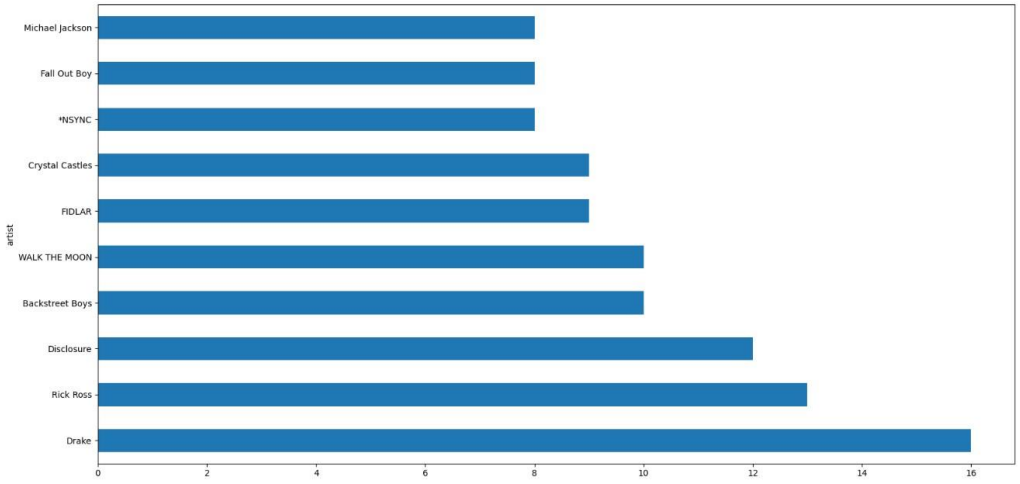
danceability mean: 0.618421914

acousticness mean: 0.187590034

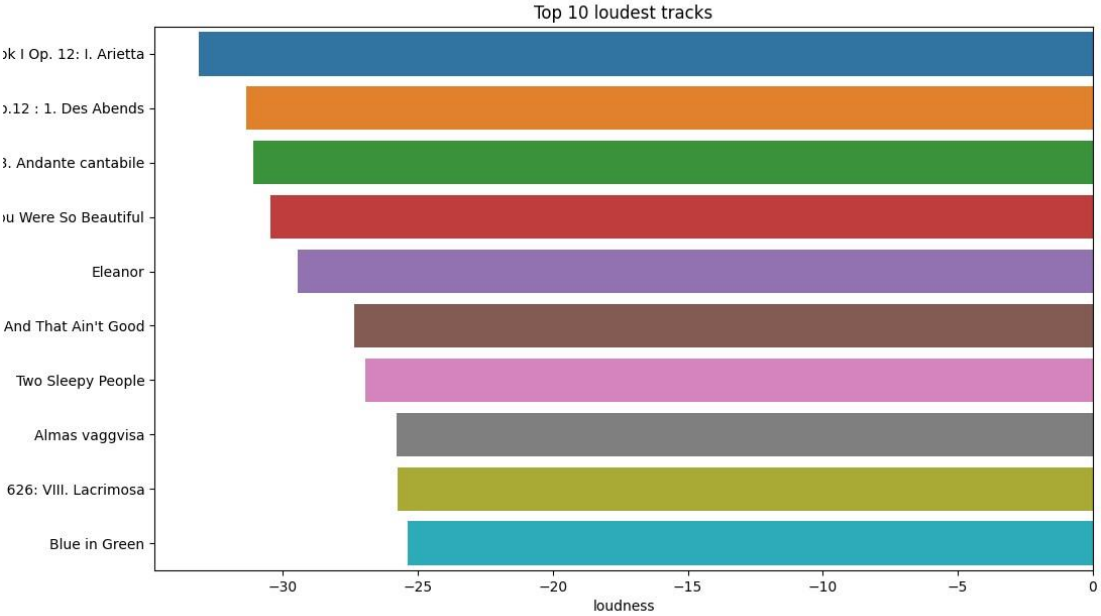
Intrumentalness, acousticness, energy, liveness, valence, speechiness, danceability are in range form 0-1, representing percentages.

Plots:

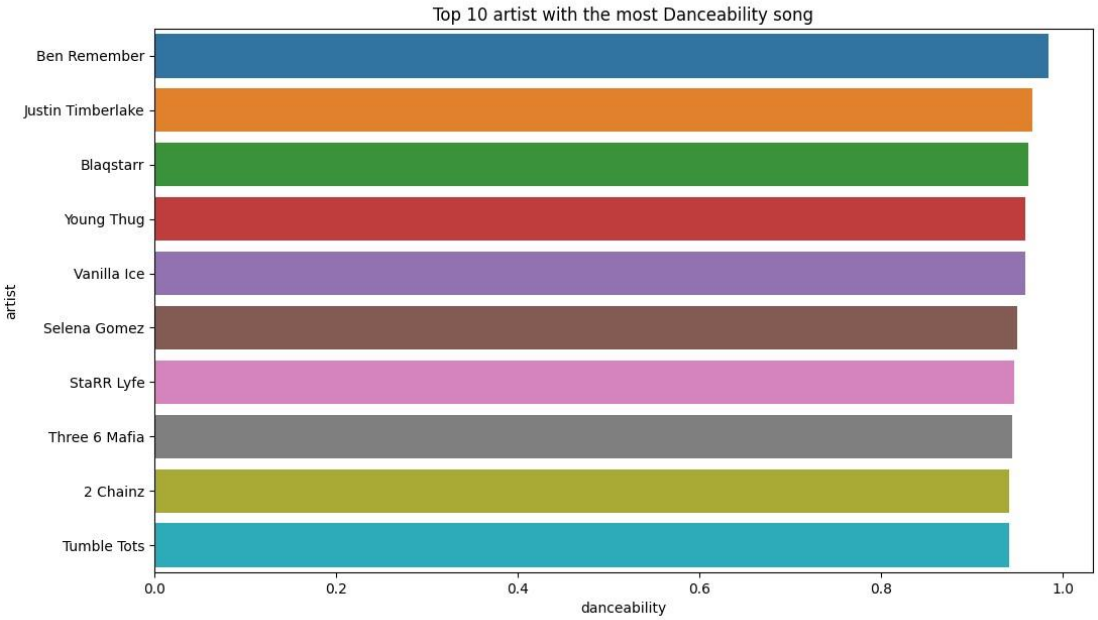
Top 10 artists:



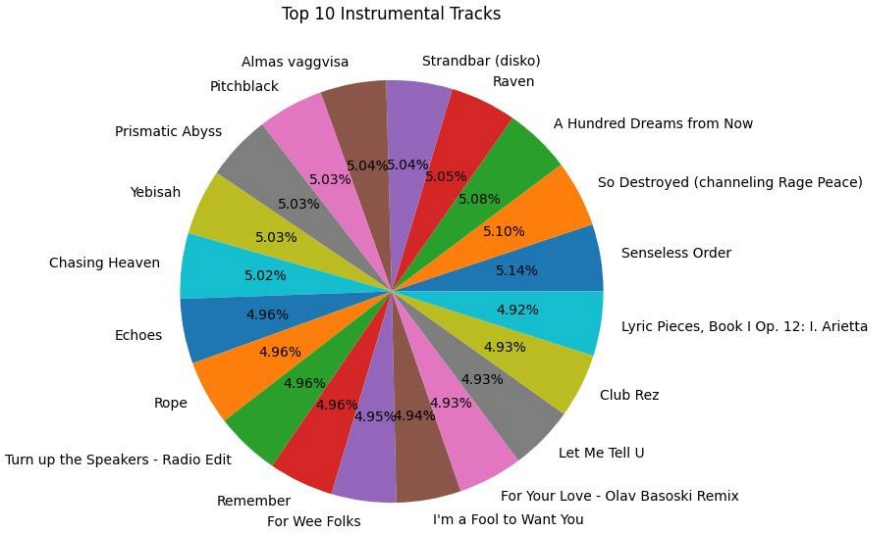
Top 10 loudest tracks:



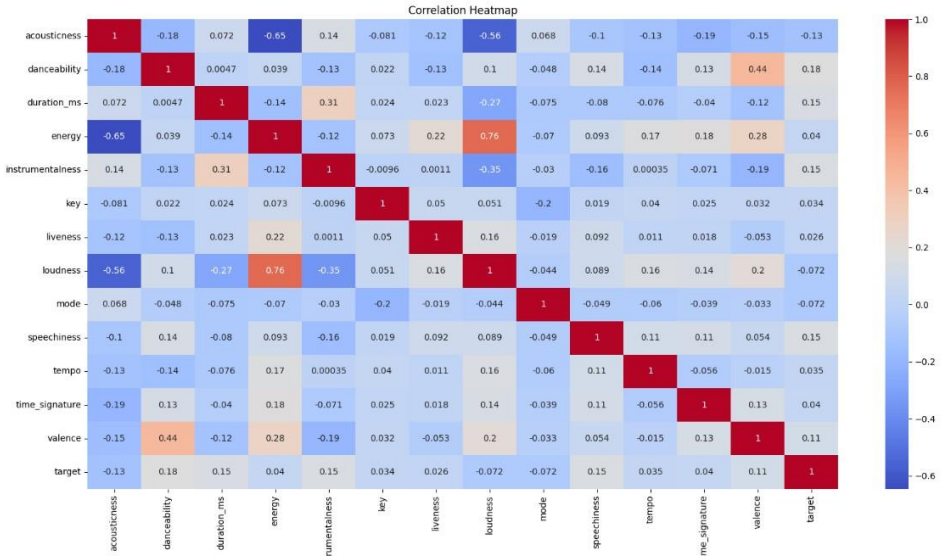
Top 10 artists by danceability:



Top 10 instrumental Tracks:

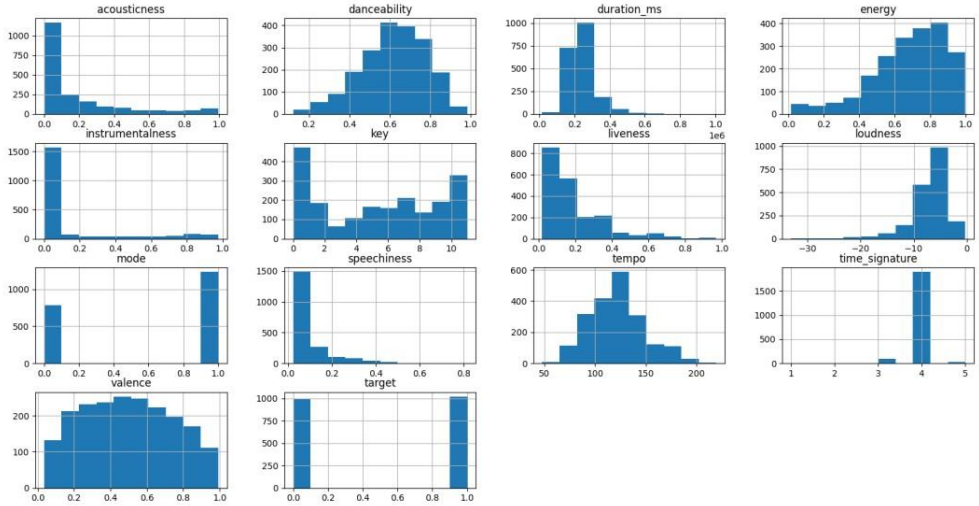


Heatmap:



Positive correlations are indicated by warm colors (reds) while negative correlations are indicated by cool colors (blues). The numerical values in the cells can also provide information about the magnitude of the correlations. For example, a value close to 1 or -1 indicates a strong correlation, while a value close to 0 indicates little to no correlation. Here we can see that energy and loudness have strong direct dependency, danceability and valence, instrumentalness and duration_ms, also have direct dependency while energy and acousticness, loudness and acousticness have strong inverse dependency.

Histogram:



All the graphs are asymmetrical but can be distributed by these categories:

Normal distribution: danceability, valence and tempo

Right-skewed distribution: liveness, acousticness, speechiness

Left-skewed distribution: energy, loudness

Symmetrical-bimodal distribution: target

Non-symmetrical-bimodal distribution: mode, time_sigbature, instrumentalness, key

df.head():

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	target	song_title	artist
0	0.0102	0.833	204600	0.434	0.021900	2	0.1650	-8.795	1	0.4310	150.062	4.0	0.286	1	Mask Off	Future
1	0.1990	0.743	326933	0.359	0.006110	1	0.1370	-10.401	1	0.0794	160.083	4.0	0.588	1	Redbone	Childish Gambino
2	0.0344	0.838	185707	0.412	0.000234	2	0.1590	-7.148	1	0.2890	75.044	4.0	0.173	1	Xanny Family	Future
3	0.6040	0.494	199413	0.338	0.510000	5	0.0922	-15.236	1	0.0261	86.468	4.0	0.230	1	Master Of None	Beach House
4	0.1800	0.678	392893	0.561	0.512000	5	0.4390	-11.648	0	0.0694	174.004	4.0	0.904	1	Parallel Lines	Junior Boys

```
PS C:\Users\-\Desktop\Spotify>
```

df.isna().sum():

```
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
key                 0
liveness            0
loudness            0
mode                0
speechiness         0
tempo               0
time_signature      0
valence             0
target              0
song_title          0
artist              0
dtype: int64
PS C:\Users\-\Desktop\Spotify>
```

df.info():

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2017 entries, 0 to 2016
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   acousticness      2017 non-null   float64
 1   danceability      2017 non-null   float64
 2   duration_ms       2017 non-null   int64
 3   energy            2017 non-null   float64
 4   instrumentalness  2017 non-null   float64
 5   key               2017 non-null   int64
 6   liveness          2017 non-null   float64
 7   loudness          2017 non-null   float64
 8   mode              2017 non-null   int64
 9   speechiness       2017 non-null   float64
 10  tempo             2017 non-null   float64
 11  time_signature    2017 non-null   float64
 12  valence           2017 non-null   float64
 13  target            2017 non-null   int64
 14  song_title        2017 non-null   object
 15  artist            2017 non-null   object
dtypes: float64(10), int64(4), object(2)
memory usage: 252.2+ KB
None
```

df.describe():

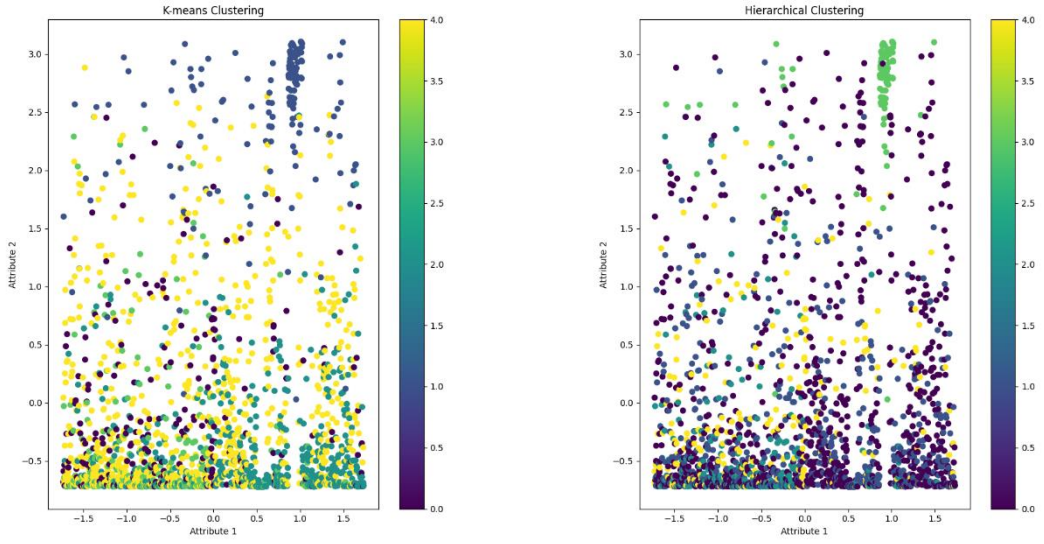
	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	target
count	2017.000000	2017.000000	2.017000e+03	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000
mean	0.187590	0.618422	2.463062e+05	0.681577	0.133286	5.342588	0.190844	-7.085624	0.612295	0.092664	121.603272	3.968270	0.496815	0.505702
std	0.259989	0.161029	8.198181e+04	0.210273	0.273162	3.648240	0.155453	3.761684	0.487347	0.089931	26.685604	0.255853	0.247195	0.500091
min	0.000003	0.122000	1.604200e+04	0.014800	0.000000	0.000000	0.018800	-33.097000	0.000000	0.023100	47.859000	1.000000	0.034800	0.000000
25%	0.009630	0.514000	2.000150e+05	0.563000	0.000000	2.000000	0.092300	-8.394000	0.000000	0.037500	100.189000	4.000000	0.295000	0.000000
50%	0.063300	0.631000	2.292610e+05	0.715000	0.000076	6.000000	0.127000	-6.248000	1.000000	0.054900	121.427000	4.000000	0.492000	1.000000
75%	0.265000	0.738000	2.703330e+05	0.846000	0.054000	9.000000	0.247000	-4.746000	1.000000	0.108000	137.849000	4.000000	0.691000	1.000000
max	0.995000	0.984000	1.004627e+06	0.998000	0.976000	11.000000	0.969000	-0.307000	1.000000	0.816000	219.331000	5.000000	0.992000	1.000000

```
PS C:\Users\-\Desktop\Spotify>
```

df.columns:

```
Index(['acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence', 'target',
       'song_title', 'artist'],
      dtype='object')
PS C:\Users\-\Desktop\Spotify>
```

Scatter plots:



From these plots we can see how the algorithms group the data, where groups are represented by huge number of dots of different colors.

Steps of Implementation:

The whole project will be implemented on python.

Step 1) Data preprocessing:

- Read the Spotify dataset into a pandas DataFrame.
- Preprocess the data by normalizing or scaling the song attribute values. You can use the StandardScaler from the sklearn.preprocessing module to standardize the numerical attributes.

Step 2) Dataset splitting:

- Split the preprocessed dataset into a training set and a validation set using the train_test_split function from the sklearn.model_selection module.
- Use an 80-20 split, where 80% of the data is used for training and 20% for validation.

Step 3) K-means clustering

- Apply K-means clustering on the training dataset using the KMeans class from the sklearn.cluster module.
- Specify the desired number of clusters, K, and fit the K-means model to the training data.
- Obtain the cluster labels for the training and validation data using the predict method of the trained K-means model.

Step 4) Hierarchical clustering

- Apply Hierarchical clustering on the training dataset using the AgglomerativeClustering class from the sklearn.cluster module.
- Fit the Hierarchical clustering model to the training data.
- Obtain the cluster labels for the training and validation data using the fit_predict method of the trained Hierarchical clustering model.

Step 5) Silhouette Score calculation (Evaluation method):

- Calculate the silhouette score for both K-means and Hierarchical clustering using the silhouette_score function from the sklearn.metrics module.
- Pass the feature matrix (song attributes) and the corresponding cluster labels (obtained from K-means and Hierarchical clustering) to the silhouette_score function.

Step 6) Visual Inspection (Evaluation method):

- Plot the resulting clusters using data visualization libraries like Matplotlib or Seaborn.
- Use scatter plots or other visualization techniques to visually inspect the clusters and assess their quality, coherence, and separation.

How program works:

Brief program flow:

1. Program must work in the following way:
2. The user enters the song he prefers (this song should be from the data set we are working with).
3. User should also input the name of the algorithm he wants to use.
4. The data set file is opened and stored in the variable.
5. The program performs the appropriate preprocessing of the data before building a model.
6. The evaluation metric is calculated for the chosen algorithm and then displayed in the console.
7. Then the clustering is applied on the data set. The clustering type depends on the type of algorithm chosen.
8. Based on the clustering data, the model is build.
9. The array of ten songs is returned to the user and printed in the console.

More detailed description:

1. The required libraries are imported, including pandas for data manipulation, scikit-learn for clustering algorithms, matplotlib for plotting, and numpy for numerical operations.
2. The preprocessing_the_data function takes a pandas DataFrame df as input and preprocesses the data by normalizing or scaling the song attribute values using the StandardScaler from scikit-learn. It returns the scaled attributes as a numpy array.
3. The kmeans_alg function takes the scaled attributes as input and applies the K-means clustering algorithm with n_clusters=5 (5 clusters) and a random state of 42. It returns the cluster labels assigned by K-means.
4. The hierarchical_alg function takes the scaled attributes as input and applies the hierarchical clustering algorithm with n_clusters=5 (5 clusters). It returns the cluster labels assigned by hierarchical clustering.
5. The scores_kmeans function calculates the silhouette score for the K-means clustering results. The silhouette score measures the quality of the clustering, indicating how well each sample fits within its cluster.
6. The scores_hierarchical function calculates the silhouette score for the hierarchical clustering results.
7. The plots_kmeans function creates a figure with two subplots, one for the scatter plot of the data points colored by the K-means cluster labels and the other for the scatter plot of the data points colored by the hierarchical cluster labels. The function uses the matplotlib library to create the plots.
8. The recommend_songs function takes a song title, cluster labels, algorithm name, and the number of recommendations as input. It retrieves the cluster label of the input song and recommends songs from the same cluster. It returns a list of recommended songs.
9. In the main part of the program (__name__ == "__main__"), the Spotify song dataset is loaded into a pandas DataFrame (df) from a CSV file.
10. The kmeans_labels variable is assigned the cluster labels obtained from applying the K-means algorithm to the preprocessed data.
11. The hierarchical_labels variable is assigned the cluster labels obtained from applying the hierarchical clustering algorithm to the preprocessed data.
12. The plots_kmeans function is called to create a visualization of the clustering results using scatter plots.
13. The scores_kmeans function is called to calculate and display the silhouette score for the K-means clustering results.
14. The scores_hierarchical function is called to calculate and display the silhouette score for the hierarchical clustering results.
15. The program enters a while loop where the user can input a song name. The recommend_songs function is called to recommend songs similar to the input song using both K-means and hierarchical clustering algorithms. The recommendations are printed to the console.
16. The loop continues until a valid song name is entered, and the recommendations are displayed.

Libraries:

NumPy: NumPy is a popular library for scientific computing in Python and is often used for matrix factorization algorithms.

pandas: pandas is a powerful data manipulation library that is widely used in Python for data analysis tasks, including data preprocessing for recommender systems.

scikit-learn: scikit-learn is a popular machine learning library in Python that provides a range of algorithms for both supervised and unsupervised learning tasks, including matrix factorization.

NLTK: NLTK is a natural language processing library that can be used for text preprocessing tasks, such as tokenization, stemming, and stopword removal.