Università di Pisa

Department of Computer Science

Master programme in Data Science and Business Informatics

Data Mining: Foundations
Project report
Spotify Tracks Dataset

Argento Pietro
Lattanzi Lorenzo
Montinaro Aldo

A.Y. 2023/2024

Contents

Data Understanding and Preparation

1.1 Dataset

The *Spotify Tracks Dataset* used in this study contains information about audio tracks available in the Spotify catalog. These tracks span 20 different genres, such as chicago-house, black-metal and breakbeat. Each track is described by essential details (track's name, artist, album name, ...) and other features like its level of popularity within the Spotify catalog. The dataset also contains audio-derived features representing various aspects like danceability, energy, key, and loudness. The variables are specified in the table below:

Name	**Description**	**Type**
`name`	Title of the track	`object`
`duration_ms`	Track length in milliseconds	`int64`
`explicit`	Whether or not the track has explicit lyrics (True = yes; False = no/unknown)	`bool`
`popularity`	Popularity of a track (0 to 100)	`int64`
`artists`	Artist(s) who performed the track	`object`
`album_name`	Album in which the track appears	`object`
`danceability`	How suitable a track is for dancing (0.0 to 1.0)	`float64`
`energy`	Perceptual measure of intensity and activity (0.0 to 1.0)	`float64`
`key`	Key of the track (standard Pitch Class notation)	`int64`
`loudness`	Overall loudness of the track in decibels (dB)	`float64`
`mode`	Modality (major or minor) of the track (major=1, minor=0)	`float64`
`speechiness`	Detects the presence of spoken words in the track	`float64`
`acousticness`	Confidence measure from 0.0 to 1.0 of whether the track is acoustic	`float64`
`instrumentalness`	Predicts whether a track contains no vocals	`float64`
`liveness`	Detects the presence of an audience in the recording	`float64`
`valence`	Musical positiveness conveyed by a track (0.0 to 1.0)	`float64`
`tempo`	Overall estimated tempo of a track in beats per minute (BPM)	`float64`
`features_duration_ms`	Duration of the track in milliseconds	`int64`
`time_signature`	Estimated time signature	`float64`
`n_beats`	Total number of time intervals of beats throughout the track	`float64`
`n_bars`	Total number of time intervals of bars throughout the track	`float64`
`popularity_confidence`	Confidence of the popularity of the song (0.0 to 1.0)	`float64`
`processing`	*no information available*	`float64`

This report contains the summary of the analysis performed on the dataset in four stages: Data Understanding and Preparation, Clustering, Classification, Pattern Mining.

1.2 Distribution of variables

The dataset contains:

- Continuous variables: duration_ms, popularity, features_duration_ms, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, n_beats, n_bars, popularity_confidence.
- Discrete variables: key, processing and time_signature.
- Categorical variables: mode and explicit (binary), name, artists, album_name and genre (these will be label encoded in order to perform the analysis with all numerical columns).

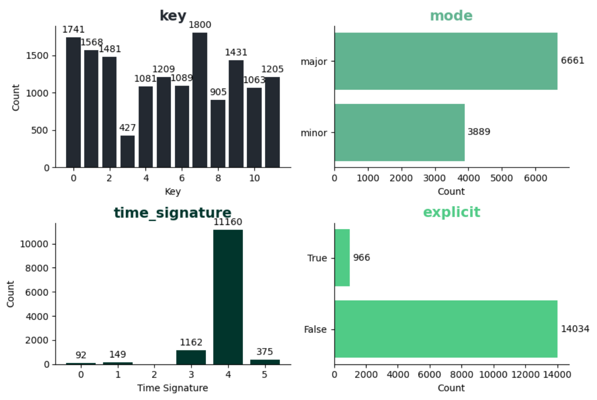


Figure 1.1: Distribution of discrete and binary variables.

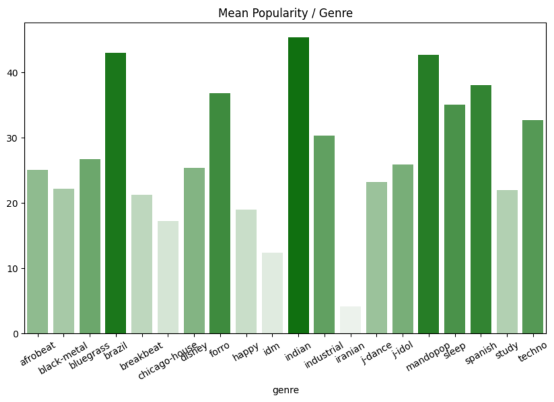


Figure 1.2: Mean popularity of tracks by genre.

Most of the tracks are in 4/4, and the keys C (0) and G (7) stand out, with a prevalence of major modes. Also noticeable is the sparse presence of songs with explicit content. By looking at the summary statistics of the continuous variables, we can calculate skewness and excess kurtosis. Excess kurtosis describes the shape of the tail of a distribution, while skewness measures the symmetry of the distribution. We can see that:

- `duration_ms` has a high kurtosis (161.5) and positive skewness (7.74), indicating that most values are concentrated to the left of the mean with a long tail to the right;
- `popularity`, `danceability`, `energy`, `acousticness`, `instrumentalness`, `valence`, `tempo`, and `processing` have negative excess kurtosis, indicating that the distribution is flatter compared to a normal distribution;

- `loudness`, `speechiness`, `liveness`, `n_beats`, and `n_bars` have positive excess kurtosis, indicating that the distribution has heavier tails and a sharper peak compared to a normal distribution;
- There is no sufficient available data for `popularity_confidence`.

1.3 Outliers detection

In order to check for possible outliers within the dataset, we can use some statistical tools, such as standard deviation and IQR (Interquartile Range), to measure the variability of data. The standard deviation method considers any data point that is a certain number of standard deviations away from the mean as an outlier; IQR is the range between the first and the third quartiles (namely $Q1$ and $Q3$): IQR $= Q3 - Q1$. The data points which fall below $Q1 - 1.5 * \text{IQR}$ or above $Q3 + 1.5 * \text{IQR}$ may be candidate outliers.
We identified which features in the dataset have potential outliers according to both methods. We can view boxplots related to some of these features:

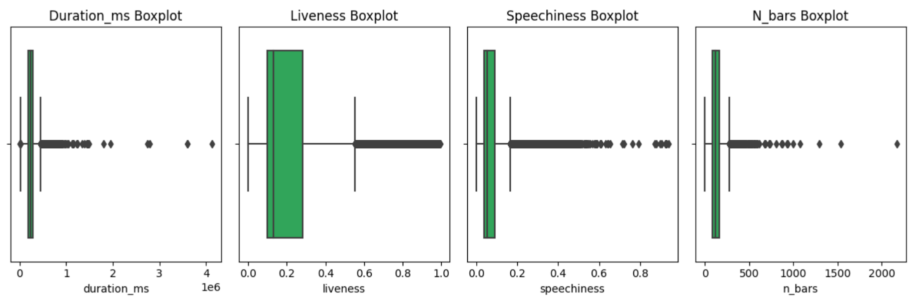


Figure 1.3: Some of the features with more presence of outliers according to the statistical methods used.

The large presence of outliers for features such as `duration_ms`, `liveness` and `speechiness` led us to check the details of the various candidate outliers: in fact, they actually refer to live recordings, DJ sets or compilations (like study background music). In this phase, we've decided not to remove any records, as doing so would result in a loss of approximately 44.59% of the samples.

1.4 Handling missing values

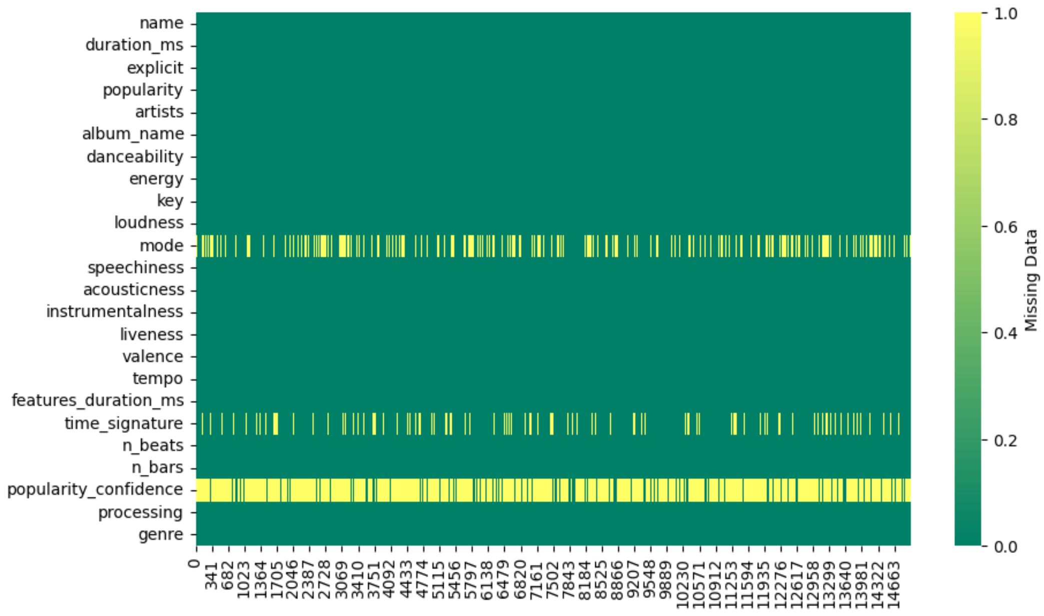


Figure 1.4: Heatmap of the data coded as boolean for "missing-ness" (1 is missing, 0 is not). Here we use the transposed boolean dataframe with `isna()` as input of the Seaborn's `heatmap()` function.

- `popularity_confidence`: since 12783 values (85.22% of the total) turn out to be missing, we therefore decided to drop the column from the original dataset, because it may not contain any information about the data.

- `mode`: the column contains binary values. Of these, 63.14% are 1 and 36.86% are 0. We decided to replace the null values by randomly inserting 0 or 1 while maintaining the percentage of the two classes. Although it might be formally wrong to assign 'minor' or 'major' even to track types such as live recordings or compilations, we noticed that among the nonmissing records it had already been done, so we simply chose to keep the percentage of classes. In this way we prevented the deletion of records that could be used in the next analysis and at the same time maintained the distribution of classes without altering the dataset.

- `time_signature`: we checked the Spotify Web API documentation in order to get more specific information about the variables. In the "Tracks" section, time_signature is described as an estimated time signature ranging from 3 to 7, indicating time signatures from "3/4" to "7/4" . Time signature (or meter) is a notation convention for specifying the number of beats in each measure (or measure). So we may decide to reject the variable, as it can be calculated from two other variables in the dataset (n_beats and n_bars). In order to verify this choice, we tested the calculation on a copy of the dataset, filtered with only non null values. The accuracy is calculated as the percentage of recalculated values corrected (i.e., with the value equal to the true value of time_signature). The result is: `Accuracy Rate:` 95.90%. We can therefore consider the formula reliable and drop the column as the feature can be derived.

1.5 Dependencies and correlations

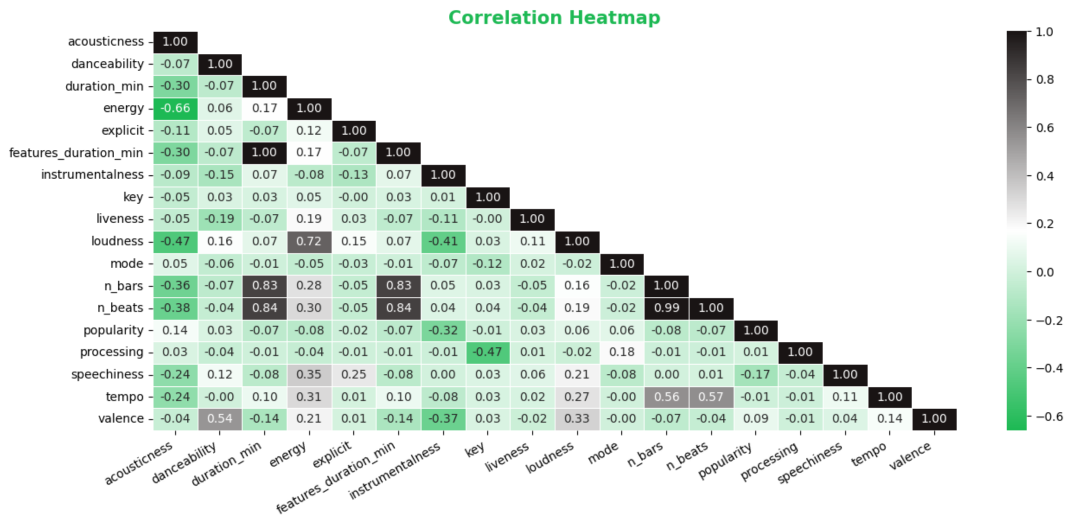


Figure 1.5: Spearman's correlation heatmap of features.

After dropping duration_ms and features_duration_ms because durations in minutes were introduced, we can see that the correlation matrix confirms that longer songs typically have more beats and bars. Energetic songs tend to be louder, while acoustic songs have lower energy. Instrumental tracks generally have lower energy and loudness. Valence is positively related to danceability and tempo, suggesting happier songs are more danceable and faster-paced. Faster-tempo songs are also more danceable. We can also drop the "features_duration_min" column because it has virtually maximum positive correlation with "duration_min".

Once we had filtered the dataset by selecting and standardizing (using the StandardScaler from `sklearn.preprocessing`) only continuous variables, we decided to implement a PCA (from `sklearn.decomposition`) to look for additional relationships between variables and identify the linear combinations of original variables that explain most of the variance in the data. After visualizing the scree plot, considering the elbow rule, we can rerun the PCA with 3 components and interpret the results. The results of the PCA, and the loadings of the original variables on the first three principal components are organized into new dataframes. This procedure leads to the following results:

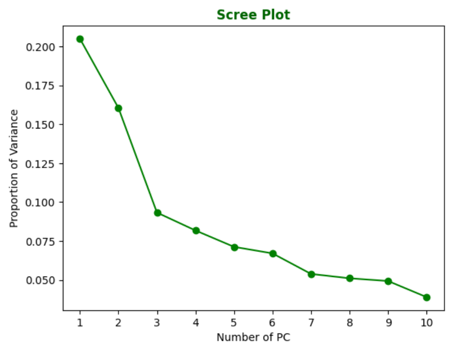


Figure 1.6: The scree plot shows how much each principal component contributes to the total variance of the data.

Feature	**PC1**	**PC2**	**PC3**
acousticness	-0.36	0.14	0.03
danceability	0.15	**-0.33**	0.00
duration_min	0.31	**0.39**	0.02
energy	**0.39**	-0.23	-0.02
explicit	0.06	-0.14	-0.04
instrumentalness	-0.14	0.31	-0.15
key	0.05	-0.03	**-0.62**
liveness	0.03	-0.03	0.04
loudness	0.36	-0.33	0.04
mode	-0.04	0.02	0.36
n_bars	**0.40**	**0.38**	0.04
n_beats	**0.41**	0.36	0.04
popularity	-0.04	-0.11	0.13
processing	-0.03	0.02	**0.65**
speechiness	0.06	-0.15	-0.08
tempo	0.31	-0.00	0.05
valence	0.15	**-0.37**	0.06

Table 1.2: Principal Component Loadings

PC1 is influenced by n_beats, n_bars, and energy, so it might be viewed as a measure of the rhythm intensity of the track. PC2 is positively influenced by duration_min and n_bars, and negatively influenced by valence and danceability: it might be seen as contrasting measure of song duration and rhythm structure with the mood and danceability of the music. PC3 is influenced by processing and negatively influenced by key, it might be viewed as a measure of the key's impact on the processing of the music. This is an interpretation of the results with the understanding that PCA is a "black box": since we do not know the procedure behind the algorithm's choice of PCs, we are giving a hypothesis that can later be confirmed or discarded during. Before continuing, we can further select the variables in the dataset: as we dropped time_signature, by the same reasoning we can drop n_beats, because it can be derived from the product `duration_min` $*$ `tempo` (which is in fact expressed in beats per minute, BPM). Entering the second phase of the analysis, we decided to normalize all variables using Z-Score normalization (`StandardScaler` class from the `sklearn.preprocessing` module), since is less sensitive than Min-Max to the presence of outliers. This is the description of the dataset at the end of the data understanding phase:

index	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
name	15000	7499.5	4330.271	0	3749.75	7499.5	11249.25	14999
explicit	15000	0.064	0.245	0	0	0	0	1
popularity	15000	0	1	-1.475	-0.722	-0.184	0.784	3.582
artists	15000	1987.648	1755.411	0	486	1437	3178	6256
album_name	15000	4084.496	2832.827	0	1569.75	3673	6399	9819
danceability	15000	0	1	-2.837	-0.567	0.149	0.741	2.208
energy	15000	0	1	-2.482	-0.667	0.2	0.862	1.3
key	15000	0	1	-1.475	-0.917	-0.08	0.757	1.593
loudness	15000	0	1	-6.766	-0.29	0.265	0.632	2.007
mode	15000	0.628	0.483	0	0	1	1	1
speechiness	15000	0	1	-0.966	-0.536	-0.378	0.056	9.863
acousticness	15000	0	1	-0.922	-0.893	-0.452	0.817	2.1
instrumentalness	15000	0	1	-0.749	-0.749	-0.741	1.194	1.863
liveness	15000	0	1	-1.11	-0.609	-0.439	0.324	3.98
valence	15000	0	1	-1.576	-0.869	-0.075	0.819	2.013
tempo	15000	0	1	-3.856	-0.726	0.034	0.591	3.051
n_bars	15000	0	1	-1.709	-0.604	-0.152	0.407	27.181
processing	15000	0	1	-1.197	-0.848	-0.38	0.948	1.54
genre	15000	0	1	-1.648	-0.824	0	0.824	1.648
duration_min	15000	0	1	-1.861	-0.522	-0.148	0.329	30.264

Clustering

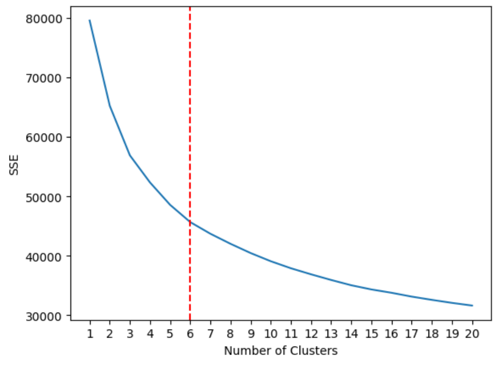
Before starting the cluster analysis, we selected the variables in the dataset, dropping nominal ones, i.e., `name`, album_name, artists and categorical/discrete ones (like `key` and `genre`). We decided to also drop the `processing` column because, observing the number of unique values and the distribution, is likely to be discrete (like key and time signature); moreover we don't have an accurate description of the variable, so it would be difficult to give an interpretation to its possible contribution in a cluster (although from the PCA there was a glimpse of a contribution in relation to key). The remaining features are 12: ['duration_min', 'popularity', 'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'n_bars'].
The clustering techniques were applied to two datasets: one with 15000 records and 12 features, and another with 10575 records and 12 features, excluding outliers (rejected based on the criteria described in Data Understanding, IQR and Standard Deviation, see 1.3); the cutoff this time is smaller (about 29% of the samples) because we dropped some columns that contained the outliers that were cut in the first case. Henceforth we will call **F** the full dataset with all samples and **C** the cut dataset without outliers. The best performing dataset was chosen for each algorithm. The process involved three steps: using all features, using only selected features, and evaluating performance. The selected features were determined by a selection algorithm based on feature importance for cluster separation (see Fig. 2.1).

2.1 Centroid-based clustering

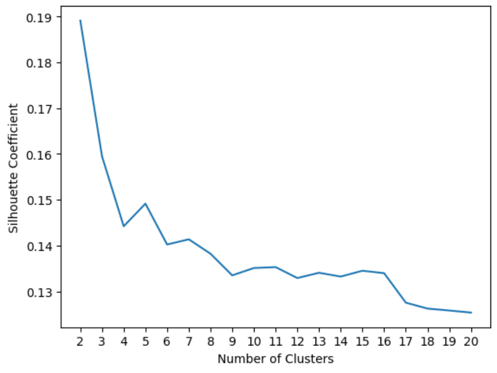
2.1.1 Choice of k

We applied the following techniques for choosing the optimal value of k, both methods provide heuristic approaches to determine the number of clusters.

Elbow Method: involves running the k-means algorithm for a choosen range of values of k. For each value of k, the Sum of Squared Errors (SSE) is calculated and stored in a list. The SSE tends to decrease toward 0 as we increase k (i.e., as the number of clusters increases, the distance from each data point to its closest centroid gets smaller). The "elbow" in the plot of SSE versus k is considered as an indicator of the appropriate number of clusters. This "elbow" is the point representing the value of k beyond which the decrease of SSE is no longer evident. When the elbow point is not distinguishable, we can use `KneeLocator` from the library `kneed`.



Silhouette Method: measures (range -1 to 1) how similar an object is to its own cluster compared to other clusters; a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. In this analysis the silhouette is calculated using a precomputed euclidean distance matrix (we used `pdist`, `squareform` from `scipy.spatial.distance`).



2.1.2 Evaluation techniques

Clustering evaluation is first performed by calculating **SSE** and **silhouette score**, for both datasets. For further validation we implemented a code that provides a thorough evaluation of the KMeans clustering results, taking into account the possibility of obtaining similar results by chance:

1. **Compute Correlation:** correlation (using the Pearson correlation coefficient) between the distance matrix of the data and the ideal similarity matrix, that is a binary matrix where 1 indicates that two points belong to the same cluster and 0 otherwise.

2. **KMeans Evaluation:** we then evaluate the significance of the clustering result by comparing it with the results of clustering on randomized datasets. This is done by: randomizing the data, computing the KMeans clustering for the randomized data, computing SSE and correlation for the randomized data, storing the results and repeating the steps for a choosen number of permutations (200 for SSE evaluation, 100 for Silhouette).

3. **Plots:** histogram and Kernel Density Estimation (KDE) plot of SSE and correlations for the randomized datasets. The original values of SSE and correlation are represented by a dashed vertical line.

If the original SSE is significantly lower than the permuted ones, it suggests that the clusters found by K-means are meaningful and not a result of random chance. On the other hand, if the original SSE is not much lower than the permuted ones, it suggests that the K-means solution might not be capturing any meaningful structure in the data. Then, if the original correlation is significantly different from the mean value given by permutations, it suggests that the clusters found by K-means are meaningful and not a result of random chance.

2.1.3 K-Means

Based on the assumptions above, for all the 12 features the results are: $k = 4$ for **F**-dataset and $k = 5$ for **C**-dataset. The one that performed slightly better (but still poor, sil$= 0.21$) is **F**, with the following parallel chart:

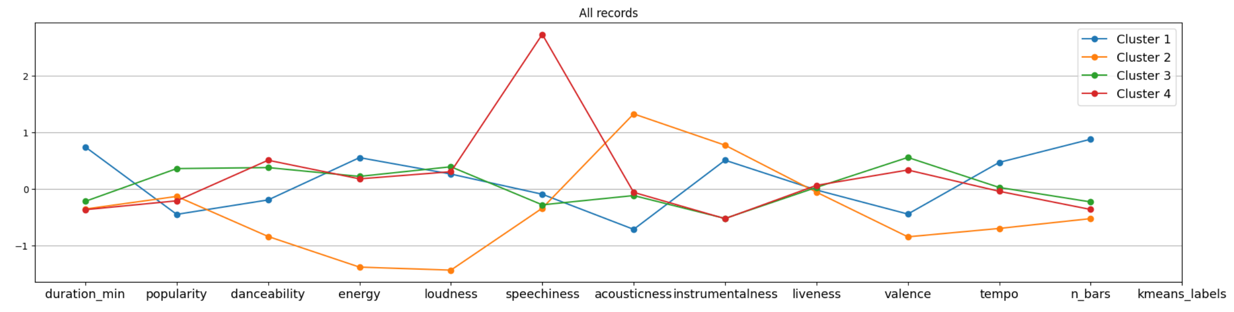


Figure 2.1: Parallel Line Chart; full dataset with all features. Points represents centroids with respect to the features.

We decided to try to select only those features that generate more pronounced separation between centroids (either based on the parallel chart display or by calculating them with an appropriate algorithm). For the two datasets they were found to be more "important":

```
F -> ['speechiness', 'acousticness', 'energy']
C -> ['instrumentalness', 'acousticness', 'energy']
```

Combinations with the 6, 5, or 4 most important features have been tested, but the results are poor. So we repeat the analysis using only the 3 most important features for each dataset. From the study of the SSE and silhouette score we obtain $k_{\mathbf{F}} = 3$ and $k_{\mathbf{C}} = 4$.
After fitting the two kmeans, the best silhouette score is 0.51 (**F**-dataset). As specified above, for further validation, correlations and SSE are calculated through permutations of the same data.

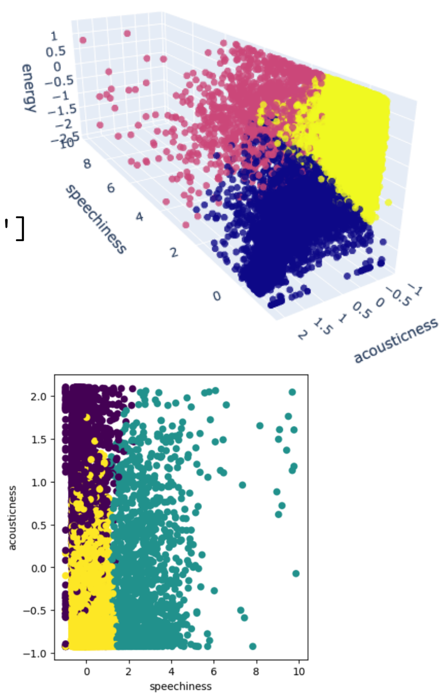


Figure 2.2: 2D and 3D plots: full dataset with selected features.

Silhouette Score	
F	0.51
C	0.49

Correlation	
F	- 0.70
C	- 0.72

SSE	
F	15859.78
C	6848.65

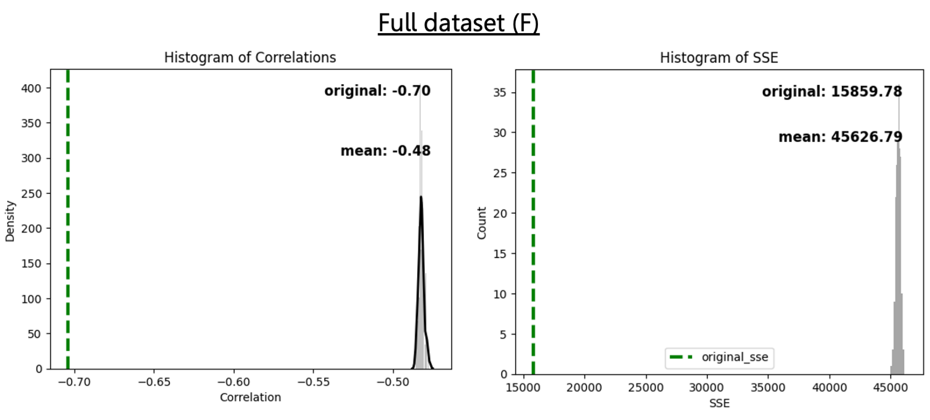


Figure 2.3: Results of KMeans evaluation, the histogram plot refers to the full dataset. F: full dataset, C: dataset without outliers.

Both datasets show a negative correlation because it's calculated between the distance matrix and the ideal similarity matrix. We can try to give an interpretation to the clusters generated by the analysis:

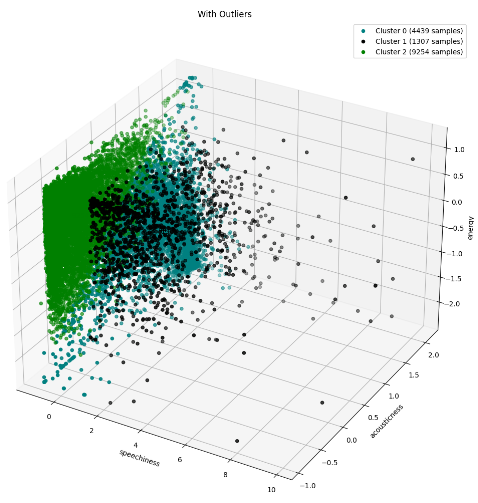


Figure 2.4: Teal: Cluster 0, Black: Cluster 1, Green: Cluster 2

- Cluster 0: **4439** samples; tracks with low energy and speechiness and high acousticness, it could contain mainly instrumental low-energy acoustic songs.
- Cluster 1: **1307** samples; tracks with ascending speechiness, high energy and low acousticness, maybe containing rap songs.
- Cluster 2: **9254** samples; tracks with low speechiness and acousticness and high energy, it may contain dance music.

2.1.4 Bisecting K-Means

Bisecting k-means is a hybrid approach between Hierarchical Clustering and K-means Clustering. Instead of partitioning the data set into k clusters in each iteration, bisecting k-means algorithm splits one cluster into two sub clusters at each bisecting step (by using k-means) until k clusters are obtained.

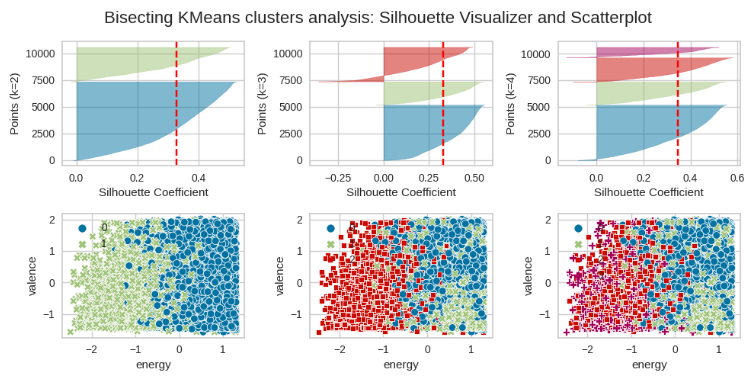


Figure 2.5: Silhouette Visualizer, from $k = 2$ to $k = 4$.

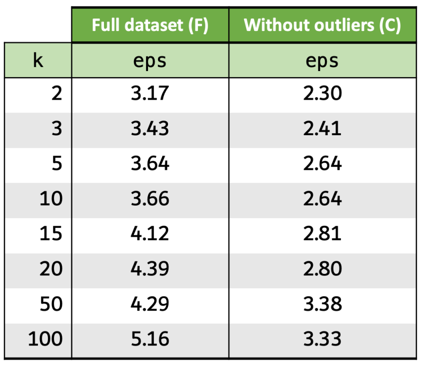
In this section we performed the analysis with $k_{\mathbf{F}} = 3$ and $k_{\mathbf{C}} = 4$ and a varying number of features, from 12 down to 3 (based on importance), computing SSE and Silhouette Coefficient for each set of features. From the results, it appears that the **C** dataset (without outliers) performs better (but still gives bad performances) in terms of both SSE (53130.08) and Silhouette (0.26) for 4 features: ['acousticness', 'energy', 'loudness', 'valence'] for **F** and ['instrumentalness', 'acousticness', 'energy', 'valence'] for **C**. Although the results are bad (SSE greater than 50000), to attempt further visualization we used Silhouette Visualizer from the yellowbrick library, which generates a series of silhouette plots and scatterplots for a variable number of clusters (in this case 2 to 4, Fig. 2.5).

2.2 Density-based clustering

Density-based clustering algorithms like DBSCAN can identify clusters of arbitrary shapes in datasets. This is an advantage over other clustering algorithms that assume clusters are spherical or have other predefined shapes. Additionally, DBSCAN can handle noise and outliers in the data, and does not require the number of clusters to be specified in advance.

2.2.1 DBSCAN

Again we run the algorithms on both the full dataset (**F**) and the dataset without outliers (**C**) to compare performance, both scaled with StandardScaler. Both datasets are tested with all features and then with the previously chosen features for kmeans analysis. The other feature selections did not affect the final results much, so we assumed that using the same attributes might have led to a better comparison between the two algorithms.
In order to choose the best parameters, we determined the optimal value of epsilon (eps), by plotting the *k*-th nearest neighbor distances for each data point in a given dataset, and then using the "elbow method" to find the optimal eps value (using again `KneeLocator` from `kneed`).



	Full dataset (F)	Without outliers (C)
k	eps	eps
2	3.17	2.30
3	3.43	2.41
5	3.64	2.64
10	3.66	2.64
15	4.12	2.81
20	4.39	2.80
50	4.29	3.38
100	5.16	3.33

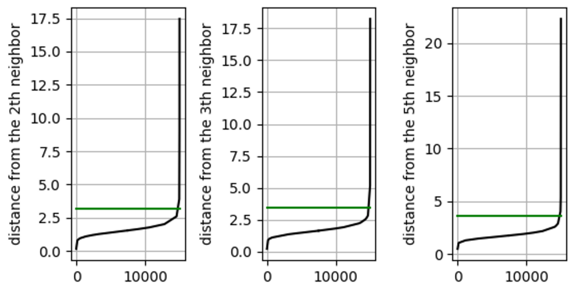


Figure 2.6: Values of *k* and `eps` for the datasets with all features.

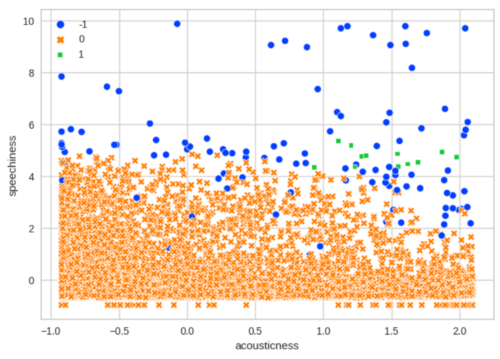


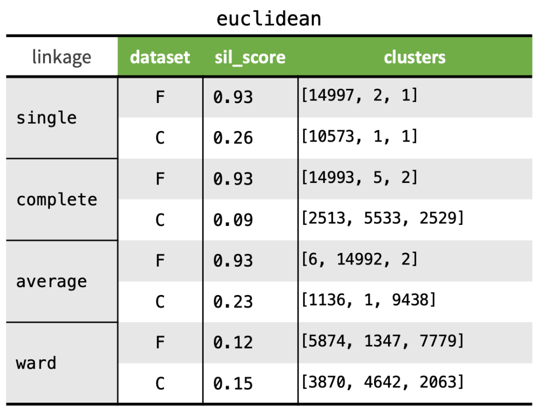
Figure 2.7: DBSCAN: Full Dataset, eps=0.75, minPts=12, sil_score=0.59

Because of the poor results with these configurations, we tried choosing a single eps value (average value of the two lists of values seen before, so 3.72 for **F** and 2.10 for **C**) testing a wider range of k (2 to 51) to find the winning combination. The best results are: `Full Dataset, eps=3.72, minPts=2, silhouette=0.45`.
Then we tried the same procedure by selecting, for each dataset, the same features selected for the kMeans (so we could make a better comparison); the results are: `Full Dataset, eps=0.75, minPts=12, silhouette=0.59`.

Despite having a convincing silhouette value, we discovered that the clustering is really very unbalanced (Fig. 2.7), since almost all the points are in the first cluster (orange), very few points are in the second cluster (green) and the others are all noise points (blue).

2.3 Hierarchical clustering

We decided to perform hierarchical clustering with Euclidean and Manhattan distances (precomputed) as metrics, and `single`, `complete`, `ward` and `average` linkages. By setting `n_clusters`$= 3$ (in order to compare results with KMeans analyisis), we used an algorithm that tested different combinations of distance-linkage returning a silhouette score. Also for the hierarchical clustering analysis, we tested the procedure first on the datasets with all features and then selecting the same features used for the second part of the KMeans. The best results were given by the configuration: dataset **F**, `euclidean` distance, `single` linkage, `sil_score` $= 0.93$. The high value of silhouette score is unfortunately due to the large imbalance of the clusters: in fact, it is noticeable that there is a cluster that contains almost all the points in the dataset and then smaller clusters formed by single points. So we tried to see if the other linkages, although with lower silhouettes, could generate better separation.



euclidean

linkage	dataset	sil_score	clusters
single	F	0.93	[14997, 2, 1]
	C	0.26	[10573, 1, 1]
complete	F	0.93	[14993, 5, 2]
	C	0.09	[2513, 5533, 2529]
average	F	0.93	[6, 14992, 2]
	C	0.23	[1136, 1, 9438]
ward	F	0.12	[5874, 1347, 7779]
	C	0.15	[3870, 4642, 2063]

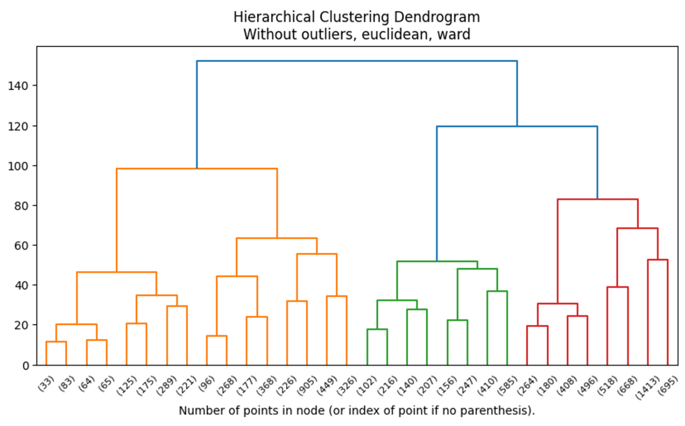


Figure 2.8: Silhouette Scores of different combinations of dataset/linkage using euclidean distance.

The C-euclidean-ward combination creates more balanced datasets, at the expense of a very low silhouette value.

2.4 Final discussion

Having reached this point, we can conclude that: DBSCAN is unable to provide optimal clustering, despite having tested several choices of eps and minPts, because it results mainly in large clusters that include almost the entire dataset, then only noise points; even the hierarchical methods produce highly unbalanced clusters.
K-Means, applied to a dataset with **selected features**, proved to be the only algorithm capable of separating some clusters in a balanced way with an acceptable silhouette value.

Classification

This section contains the methods applied to two different classification problems: the first target chosen is the variable `genre`, which contains 20 different classes; next we tried to apply the same algorithms to `popularity`, a continuous variable discretized into three classes (low, medium, high). For this analysis, the data were standardized using `StandardScaler` (from `sklearn.preprocessing`), and in a first step we chose to use all features in the dataset before making a selection: *explicit*, *popularity*, *danceability*, *energy*, *key* (one-hot encoded), *loudness*, *mode*, *speechiness*, *acousticness*, *instrumentalness*, *liveness*, *valence*, *tempo*, *n_ bars*, *processing*, *genre*, *duration_ min*. Which could be an expected accuracy value for the classification of `genre`? A starting point might be a comparison with a random model: a model that randomly assigns classes would have an expected accuracy of 5% (1 in 20); therefore, any model with significantly higher accuracy than this could be considered an improvement, even if it did not achieve an accuracy close to 1. The same is true for classification with three classes, where any model with accuracy greater than 1/3 can be considered acceptable (for purely analytical purposes) although perhaps not realistically usable.
For each classification algorithm presented, the work process was carried out in several successive stages: initially we apply a basic model with default parameters by measuring its accuracy, so as to have a starting value to be improved with appropriate choice of parameters; in fact, the second stage concerns the tuning of the hyperparameters using `GridSearchCV` by `sklearn.model_selection`; once the model with the optimal parameters is obtained, we proceed with the evaluation. All the analysis presented were performed on the entire dataset used for the previous steps, appropriately split into `train` and `test` subsets. For some further tests, an external test dataset (never seen by the models), provided by the lecturer, was also used in order to have the three sets train, validation and test.

Evaluation metrics and techniques

- Accuracy and Overall Error Estimate: fraction of predictions got right by our model and relative error (1-accuracy).
- Precision-recall curves: trade-off between the rate of true positives and the positive predictive value for a predictive model using different probability thresholds.
- F1 score: measures the performance of a model that combines precision and recall. It ranges from 0 to 1, where 1 indicates perfect precision and recall.
- ROC-AUC score: performance of a classification model at all classification thresholds. The area under the curve (AUC) measures the 2D area under the ROC curve, providing an aggregate measure of performance across all classification thresholds.

All the results provided refer to the performance measure on the test set.

3.1 Target: `genre`

3.1.1 KNN

The base model returns an accuracy of 0.37. For hyperparameter tuning, we start by looking at the trend of accuracy as the number of neighbors changes (from 1 to 100): the algorithm generates an error bar graph showing the average scores and their standard deviations for each number of neighbors and highlights the number of neighbors that obtained the highest accuracy, based on its cross-validation performance. The configuration `{'metric': 'cityblock', 'n_neighbors': 26, 'weights': 'distance'}` performs best according to GridSearch, with a final accuracy of about 0.48.

The model does not behave in the same way on the various classes, in particular it is better able to recognize tracks in the genre of "black metal" (class 2), "sleep" (class 8) and "study" (class 13).

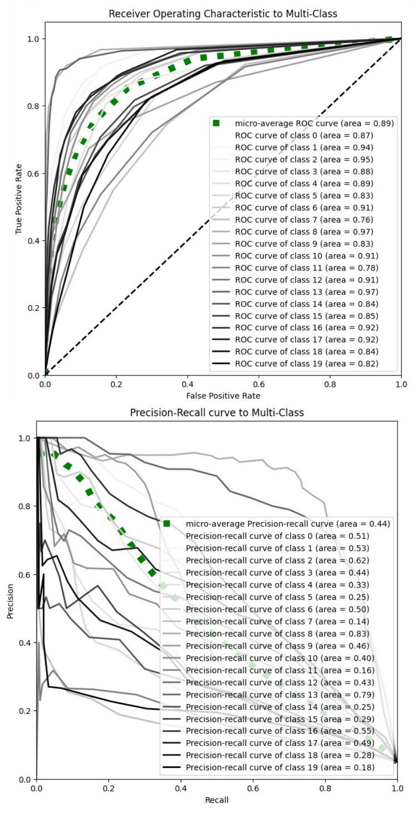


Figure 3.1: ROC-AUC and Precision/Recall for KNN model.

3.1.2 Naive Bayes

The Naive Bayes classifier is a classification algorithm based on Bayes' Theorem with the assumption that features are conditionally independent given the class label. In this study we distinctly use two types of NB: first, `Gaussian Naive Bayes`, a variant applicable to continuous variables in which attributes are assumed to follow a Gaussian distribution; it predicts the output variable based on each parameter independently. We then use `Categorical Naive Bayes`, suitable for discrete features that are categorically distributed.

Classifier	Accuracy	Precision	Recall	F1
GaussianNB	0.37	0.37	0.37	0.33
CategoricalNB	0.40	0.38	0.40	0.37

3.1.3 Decision Tree

The base model returns a train accuracy of 1.0 and a test accuracy 0.426, with an obvious presence of overfitting. Before applying GridSearch we choose the ranges of some parameters based on the behavior of the model by plotting accuracy against different values: in particular we tuned `max_depth`, `min_samples_split` and `min_samples_leaf`. The configuration with the optimal parameters turns out to be `{'splitter': 'best', 'min_samples_split': 56, 'min_samples_leaf': 10, 'max_depth': 11, 'criterion': 'gini'}`, with an average accuracy of 0.45. To visualize the tree we used `tree.export_graphviz` from `sklearn` (below the graph is cut down to `max_depth` = 2 to make its content more understandable).

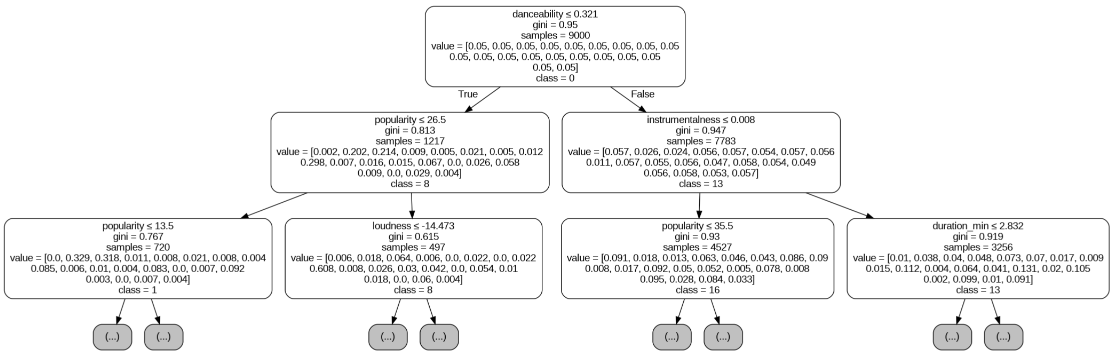

danceability ≤ 0.321
gini = 0.95
samples = 9000
value = [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05 0.05, 0.05]
class = 0
True
False
popularity ≤ 26.5
gini = 0.813
samples = 1217
value = [0.002, 0.202, 0.214, 0.009, 0.005, 0.021, 0.005, 0.012 0.298, 0.007, 0.016, 0.015, 0.067, 0.0, 0.026, 0.058 0.009, 0.0, 0.029, 0.004]
class = 8
instrumentalness ≤ 0.008
gini = 0.947
samples = 7783
value = [0.057, 0.026, 0.024, 0.056, 0.057, 0.054, 0.057, 0.056 0.011, 0.057, 0.055, 0.056, 0.047, 0.058, 0.054, 0.049 0.056, 0.058, 0.053, 0.057]
class = 13
popularity ≤ 13.5
gini = 0.767
samples = 720
value = [0.0, 0.329, 0.318, 0.011, 0.008, 0.021, 0.008, 0.004 0.085, 0.006, 0.01, 0.004, 0.083, 0.0, 0.007, 0.092 0.003, 0.0, 0.007, 0.004]
class = 1
loudness ≤ -14.473
gini = 0.615
samples = 497
value = [0.006, 0.018, 0.064, 0.006, 0.0, 0.022, 0.0, 0.022 0.608, 0.008, 0.026, 0.03, 0.042, 0.0, 0.054, 0.01 0.018, 0.0, 0.06, 0.004]
class = 8
popularity ≤ 35.5
gini = 0.93
samples = 4527
value = [0.091, 0.018, 0.013, 0.063, 0.046, 0.043, 0.086, 0.09 0.008, 0.017, 0.092, 0.05, 0.052, 0.005, 0.078, 0.008 0.095, 0.028, 0.084, 0.033]
class = 16
duration_min ≤ 2.832
gini = 0.919
samples = 3256
value = [0.01, 0.038, 0.04, 0.048, 0.073, 0.07, 0.017, 0.009 0.015, 0.112, 0.004, 0.064, 0.041, 0.131, 0.02, 0.105 0.002, 0.099, 0.01, 0.091]
class = 13
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

Figure 3.2: Decision Tree with `genre` as target variable.

We can also see the importance of features according to the model used: the most important is `popularity` (0.23), followed by `acousticness` (0.13), `danceability` (0.13), `duration_min` (0.12) and so on; along with some features that have a very low contribution (like `liveness` or `processing`), we see that `key`, `explicit` and `mode` result useless (according to this model) in determining the genre of the track. We can therefore assume that the model relies mainly on how popular, acoustic and danceable a track is to identify its genre. Although predicting genre using popularity might be counterintuitive, removing it from features drops performance dramatically.
In a Decision Tree, when the parameter `ccp_alpha` increases, more of the tree is pruned, thus creating a decision tree that generalizes better. So we implemented a procedure that could find the optimal value of the parameter. The result is [`Best ccp_alpha value is: 0.000386`], but the performance of the model has not improved noticeably (accuracy about 0.46).
We anticipated above that for an initial classification study we used all the features in the dataset. After seeing how all the models behaved, we decided to make a selection of the variables to be used: we used the correlation coefficient (the highest the correlation with target the highest the importance) and `mutual_info_classif` from `sklearn.feature_selection`, that measures the reduction in entropy about one variable given the knowledge of another; in other words, it measures the dependency between each feature and the target variable. We used these two tools to obtain a combined score of each variable, so that we could select those with the highest score. From the results comes confirmation of the features identified by the decision tree; in fact, by selecting the 5 features with the highest score, we can repeat the analysis using only the variables `danceability`, `popularity`, `acousticness`, `energy`, `valence`. To recapitulate, the overall results are:

Decision Tree	Train acc	Test acc	precision	recall	F1
base model	1.0	0.43	0.43	0.43	0.43
GridSearch	0.55	0.45	0.47	0.46	0.46
ccp_alpha	0.55	0.46	0.47	0.46	0.46
Feature selection	0.54	0.46	0.47	0.46	0.46

3.1.4 Results

- KNN: [`accuracy`: 0.48, `roc auc`: 0.89, `precision/recall auc`: 0.44]. Although we have improved the basic model and we are above the expected value of an accuracy of 1/20, for pure analytical purposes we can consider the model acceptable but not usable in a real-world context, given the high error rate: about half of the data are not classified correctly.

- Naive Bayes: In this case the error increases to about 60%, which means that the two models Gaussian and Categorical (on different feature groups, continuous and categorical) still perform worse than KNN.

- DecisionTree: The accuracy of the model does not exceed 0.46, even after appropriate parameter tuning. However, we can still study the behavior of the model and how it was able to capture relationships between variables based on the importance given in the training phase.

3.2 Target: `popularity`

As anticipated, we also decided to test the studied techniques on another target variable: we discretized popularity using `pd.cut` into three classes, `[Low (0-50), Medium (50-80), High (80-100)]`. We followed the same process used for the target `genre`, that is, we started with a basic model (for this study we used only the Decision Tree) and then tuned its parameters to improve its performance. In this case, the optimal configuration is `{'splitter': 'best', 'min_samples_split': 94, 'min_samples_leaf': 58, 'max_depth': 5, 'criterion': 'entropy'}` with an average accuracy of 0.87. The most important features are `genre`, `energy` and `instumentalness`.
In this case, however, there is a problem of class imbalance: in fact, tracks with low popularity cover almost the entire dataset, mediums are on the order of hundreds, and highs are a few dozen. So we implemented a process for optimizing class weights in the classifier, in order to handle class imbalance. The code generates a range of weights for the classes, calculates 10-fold cross-validated F1 scores for each set of weights, identifies the weights that yield the highest score, and trains a final model using these optimal weights. We then again applied the procedure for the ccp_alpha parameter and feature selection. The results are:

Decision Tree	Train acc	Test acc	precision	recall	F1
base model	0.99	0.81	0.44	0.48	0.45
GridSearch	0.87	0.87	0.48	0.35	0.34
Class weights	0.36	0.35	0.39	0.47	0.25
ccp_alpha	0.36	0.42	0.39	0.49	0.29
Feature selection	0.44	0.43	0.39	0.49	0.29

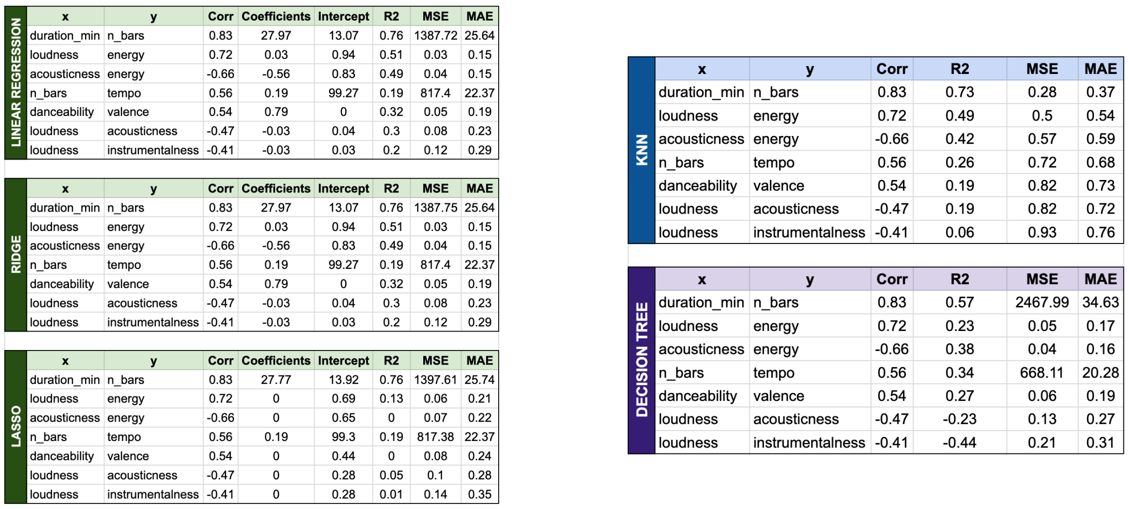
We can conclude that while the results were promising at first, if we go to consider the weights of the various classes (due to imbalance), the model loses its ability to generalize by a large margin.

Pattern Mining and Regression

4.1 Regression

Simple Regression

We created a function that performs simple linear regression using the provided regressor (that will be Linear, Ridge or Lasso) on the given training and testing datasets. It returns a dictionary containing the fitted model, its coefficients and intercept, and the R^2, MSE, and MAE of the model on the test data. Then, for each regressor, our code performs a series of operations to identify pairs of features in a dataset that have a strong positive or negative correlation, and then applies simple linear regression to these pairs. After sorting the feature pairs based on the absolute value of their Spearman correlation coefficients, it filters the pairs to only include those with a correlation coefficient greater than 0.4 or less than -0.4 (the threshold was chosen based on the values displayed in the correlation matrix). For each of these selected pairs, the code fits the model on the training data and uses it to predict the dependent variable in the test data. The list of results is converted into a pandas DataFrame that provides a comprehensive summary of the relationships between different features in the dataset and the performance of the model. In addition to the linear methods, we also used on the same pairs of features two previously seen models, KNN and DecisionTree, this time as regressors.



LINEAR REGRESSION

x	y	Corr	Coefficients	Intercept	R2	MSE	MAE
duration_min	n_bars	0.83	27.97	13.07	0.76	1387.72	25.64
loudness	energy	0.72	0.03	0.94	0.51	0.03	0.15
acousticness	energy	-0.66	-0.56	0.83	0.49	0.04	0.15
n_bars	tempo	0.56	0.19	99.27	0.19	817.4	22.37
danceability	valence	0.54	0.79	0	0.32	0.05	0.19
loudness	acousticness	-0.47	-0.03	0.04	0.3	0.08	0.23
loudness	instrumentalness	-0.41	-0.03	0.03	0.2	0.12	0.29

RIDGE

x	y	Corr	Coefficients	Intercept	R2	MSE	MAE
duration_min	n_bars	0.83	27.97	13.07	0.76	1387.75	25.64
loudness	energy	0.72	0.03	0.94	0.51	0.03	0.15
acousticness	energy	-0.66	-0.56	0.83	0.49	0.04	0.15
n_bars	tempo	0.56	0.19	99.27	0.19	817.4	22.37
danceability	valence	0.54	0.79	0	0.32	0.05	0.19
loudness	acousticness	-0.47	-0.03	0.04	0.3	0.08	0.23
loudness	instrumentalness	-0.41	-0.03	0.03	0.2	0.12	0.29

LASSO

x	y	Corr	Coefficients	Intercept	R2	MSE	MAE
duration_min	n_bars	0.83	27.77	13.92	0.76	1397.61	25.74
loudness	energy	0.72	0	0.69	0.13	0.06	0.21
acousticness	energy	-0.66	0	0.65	0	0.07	0.22
n_bars	tempo	0.56	0.19	99.3	0.19	817.38	22.37
danceability	valence	0.54	0	0.44	0	0.08	0.24
loudness	acousticness	-0.47	0	0.28	0.05	0.1	0.28
loudness	instrumentalness	-0.41	0	0.28	0.01	0.14	0.35

KNN

x	y	Corr	R2	MSE	MAE
duration_min	n_bars	0.83	0.73	0.28	0.37
loudness	energy	0.72	0.49	0.5	0.54
acousticness	energy	-0.66	0.42	0.57	0.59
n_bars	tempo	0.56	0.26	0.72	0.68
danceability	valence	0.54	0.19	0.82	0.73
loudness	acousticness	-0.47	0.19	0.82	0.72
loudness	instrumentalness	-0.41	0.06	0.93	0.76

DECISION TREE

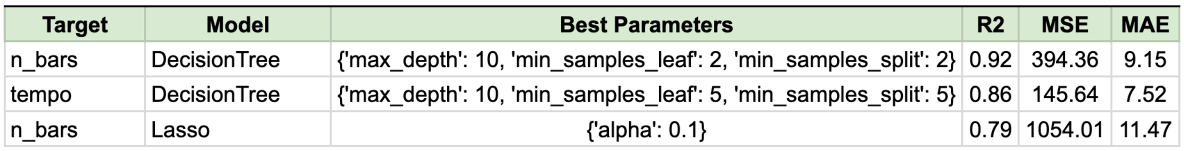
x	y	Corr	R2	MSE	MAE
duration_min	n_bars	0.83	0.57	2467.99	34.63
loudness	energy	0.72	0.23	0.05	0.17
acousticness	energy	-0.66	0.38	0.04	0.16
n_bars	tempo	0.56	0.34	668.11	20.28
danceability	valence	0.54	0.27	0.06	0.19
loudness	acousticness	-0.47	-0.23	0.13	0.27
loudness	instrumentalness	-0.41	-0.44	0.21	0.31

Figure 4.1: Results of linear and non-linear regression, performed on different pairs of features.

The best performing model is the one between duration_min and n_bars. This confirms what we expected, because the length of the song increases as the number of bars it contains increases. One of the worst performing models is the one between n_bars and tempo. In fact, the number of bars in a song does not significantly influence its tempo, at least not in a linear way. The tempo and the number of beats of a song are largely independent musical elements: knowing the number of beats of a song does not provide much information about its tempo. The tempo of a piece is often chosen based on the mood or musical style, while the number of beats is typically determined by the structure and length of the piece.

Multiple Regression

We created a procedure that iterates over the list of targets containing only continuous variables of the dataset: the code, for each target, splits the dataset into train/test, standardizes the data, tests the different types of models seen above (performing a gridsearch based on the grid of parameters chosen) and then saves the results in a dataframe, sorting them by descending value of R^2.

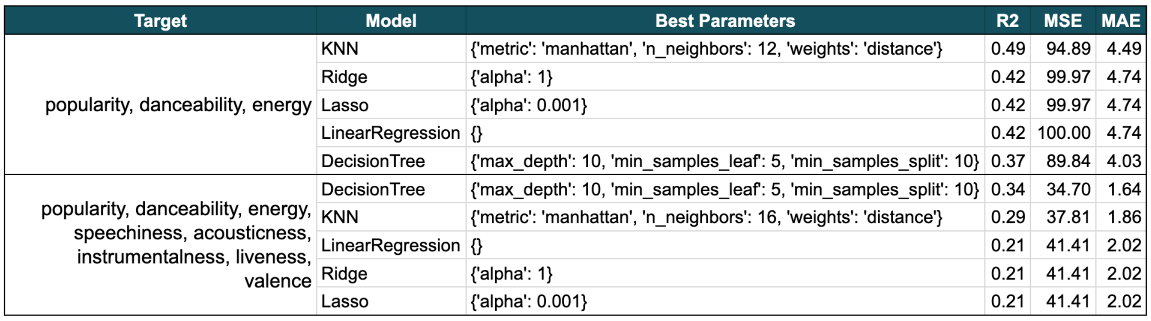


Target	Model	Best Parameters	R2	MSE	MAE
n_bars	DecisionTree	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2}	0.92	394.36	9.15
tempo	DecisionTree	{'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 5}	0.86	145.64	7.52
n_bars	Lasso	{'alpha': 0.1}	0.79	1054.01	11.47

Figure 4.2: Part of the results of multiple regression analysis: here we can see the 3 best combinations target-model.

Multivariate Regression

As targets this time we have chosen: list `[popularity, danceability, energy]`, which can have implications in the music business, aiding in music recommendation, playlist creation, artist guidance, and advertising; then, after checking correlation to avoid multicollinearity, we added `[speechiness, acousticness, instrumentalness, liveness, valence]`, so that we can see performances with more targets.



Target	Model	Best Parameters	R2	MSE	MAE
popularity, danceability, energy	KNN	{'metric': 'manhattan', 'n_neighbors': 12, 'weights': 'distance'}	0.49	94.89	4.49
	Ridge	{'alpha': 1}	0.42	99.97	4.74
	Lasso	{'alpha': 0.001}	0.42	99.97	4.74
	LinearRegression	{}	0.42	100.00	4.74
	DecisionTree	{'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 10}	0.37	89.84	4.03
popularity, danceability, energy, speechiness, acousticness, instrumentalness, liveness, valence	DecisionTree	{'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 10}	0.34	34.70	1.64
	KNN	{'metric': 'manhattan', 'n_neighbors': 16, 'weights': 'distance'}	0.29	37.81	1.86
	LinearRegression	{}	0.21	41.41	2.02
	Ridge	{'alpha': 1}	0.21	41.41	2.02
	Lasso	{'alpha': 0.001}	0.21	41.41	2.02

Figure 4.3: Results of the multivariate regression analysis.

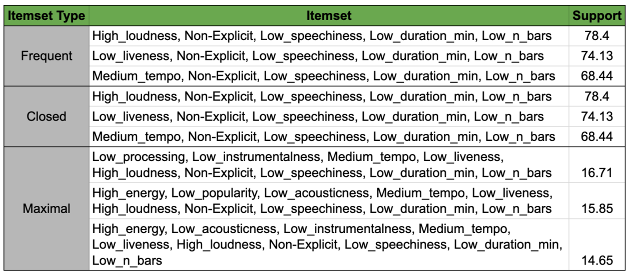
4.2 Pattern Mining

Before starting the analysis, we prepared the data: we mapped the categorical features to the actual values, for example, by substituting each key value for the corresponding Pitch Class Notation value (1 represents C pitch, 2 represents $C\sharp/D\flat$, and so on); then we discretized the continuous variables, each in 3 bins, using `pd.cut`.

4.2.1 Frequent Patterns

Apriori

Before running the Apriori algorithm, we looked for the optimal values of supp (minimum support of an item set) and zmin (minimum number of items per item set). The code defines a grid of parameters for the Apriori algorithm: `supp` values are generated using numpy's linspace function to create 5 evenly spaced values between 10 and 50; `zmin` values are generated using numpy's arange function to create a range of values from 5 to the number of columns in the dataframe. Then it runs the Apriori algorithm for each combination of `supp` and `zmin` values in the parameter grid, computing, for each combination, the number of frequent, closed, and maximal itemsets. Results are sorted by the number of frequent, closed, and maximal itemsets in descending order. The best configuration founded is `[supp=10, zmin=5]`. In fact, by setting 2, 3 or 4 as the minimum of the `zmin` range, we realized that the patterns found were just a simple confirmation of some information already found in data understanding: for example, in the dataset most of the tracks have a fairly short duration (the distribution is positively skewed). By setting a value of zmin equal to 5 instead, we can already begin to get more useful patterns.



Itemset Type	Itemset	Support
Frequent	High_loudness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	78.4
	Low_liveness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	74.13
	Medium_tempo, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	68.44
Closed	High_loudness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	78.4
	Low_liveness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	74.13
	Medium_tempo, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	68.44
Maximal	Low_processing, Low_instrumentalness, Medium_tempo, Low_liveness, High_loudness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	16.71
	High_energy, Low_popularity, Low_acousticness, Medium_tempo, Low_liveness, High_loudness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	15.85
	High_energy, Low_acousticness, Low_instrumentalness, Medium_tempo, Low_liveness, High_loudness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	14.65

Figure 4.4: Highest support values for the frequent, closed, and maximal itemsets derived from the Apriori algorithm.

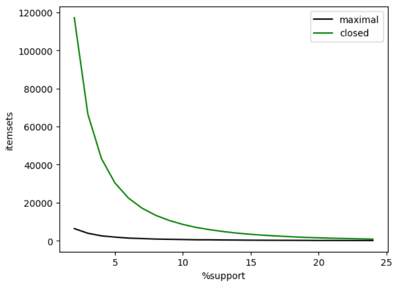


Figure 4.5: Number of maximal and closed itemsets (y-axis) for different support thresholds (x-axis).

Our dataset predominantly consists of non-explicit tracks with high volume, low speechiness, a duration range that remains below 22 minutes, and a low number of bars. Tracks with medium tempo and low speechiness also have a significant presence. The presence of liveness feature among the most frequent patterns also suggests that dataset contains lots of tracks that have a low "live" feel.

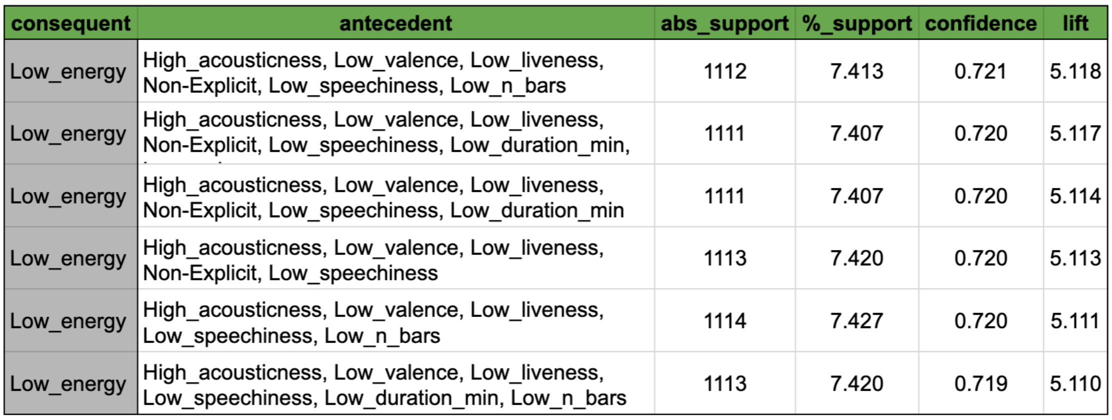
FP-Growth

We implemented the same procedure used for apriori, with the goal of finding the optimal values of supp and zmin, which are again `[supp=10, zmin=5]`. These are the three frequent patterns found with the highest support:

Frequent Itemset	Support
High_loudness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	78.4
Low_liveness, Non-Explicit, Low_speechiness, Low_duration_min, Low_n_bars	74.13
Low_liveness, High_loudness, Low_speechiness, Low_duration_min, Low_n_bars	67.97

4.2.2 Association Rules

We implemented a code that runs the Apriori algorithm for association rule mining over a range of confidence values (from 50 to 95) with a step of 5. For each `conf` value we then compute the average lift for the rules generated. We chose as optimal `conf` value the confidence level with the highest average lift. Finally, the code runs the Apriori algorithm again with the best confidence level and displays the resulting association rules sorted by lift in descending order.



consequent	antecedent	abs_support	%_support	confidence	lift
Low_energy	High_acousticness, Low_valence, Low_liveness, Non-Explicit, Low_speechiness, Low_n_bars	1112	7.413	0.721	5.118
Low_energy	High_acousticness, Low_valence, Low_liveness, Non-Explicit, Low_speechiness, Low_duration_min,	1111	7.407	0.720	5.117
Low_energy	High_acousticness, Low_valence, Low_liveness, Non-Explicit, Low_speechiness, Low_duration_min	1111	7.407	0.720	5.114
Low_energy	High_acousticness, Low_valence, Low_liveness, Non-Explicit, Low_speechiness	1113	7.420	0.720	5.113
Low_energy	High_acousticness, Low_valence, Low_liveness, Low_speechiness, Low_n_bars	1114	7.427	0.720	5.111
Low_energy	High_acousticness, Low_valence, Low_liveness, Low_speechiness, Low_duration_min, Low_n_bars	1113	7.420	0.719	5.110

Figure 4.6: Association rules sorted by lift.

We can see the following information:

- Consequents: in our case it's mainly related to 'energy'.
- Confidence: measures how often the rule has been found to be true. For example, a confidence of 0.72 for the first rule means that in about 72% of the transactions containing [High_acousticness, Low_valence, Low_liveness, Non-Explicit, Low_speechiness, Low_n_bars] appear to have a low energy level.
- lift: ratio of the observed support to that expected if the antecedent and the consequent were independent. In our case, all the higher lifts are around 5, which means that the rules are quite significant.

The first rule can be read as: "If a relatively short song is acoustic, non-expicit and has low valence, liveness, speechiness, then it's likely to have a low energy". This rule has a confidence of about 72% and is about 5.1 times as likely to occur as would be expected if the conditions and result were independent. This pattern repeats for the other rules with slight variations in the conditions. All this also confirms the study of data understanding, in which we assumed such relationships based more or less only on correlations.