University of Pisa
Data Mining II
A.Y 2023-2024

# Data Mining II Project

**Submitted to:**
Prof. Riccardo Guidotti

**Submitted by:**
Alessandro Carella
Sara Hoxha
Rafael Urbina

October 20, 2025

# Contents

# 1 Data Understanding & Preparation

## 1.1 Data semantics

The developed project has two tabular datasets with the following features:

**Tracks dataset**

| Name | Type | Description |
|---|---|---|
| id | Nominal | Unique identifier for each record in the dataset. |
| genre | Nominal | Categorization of the music based on its style. |
| name | Nominal | The title or name of the track. |
| disc_number | Discrete | The disc number in a multi-disc album. |
| duration_ms | Continuous | The duration of the track in milliseconds. |
| explicit | Binary | Indicates whether the track contains explicit content (1 for true, 0 for false). |
| popularity | Ratio | A measure of the track's popularity. |
| track_number | Ratio | The track number within an album. |
| artists | Nominal | The artists or contributors involved in creating the track. |
| album_type | Nominal | The type of album (e.g., album, single, compilation). |
| album_name | Nominal | The title or name of the album. |
| album_release_date | Nominal | The release date of the album. |
| album_release_date _precision | Ordinal | Precision level of the album release date (e.g., day, month, year). |
| album_total_tracks | Discrete | The total number of tracks in the album. |
| danceability | Continuous | Measure of how suitable a track is for dancing. |
| energy | Continuous | Measure of the intensity and activity of the track. |
| key | Ratio | The key signature of the track. |
| loudness | Continuous | The overall loudness of the track in decibels. |
| mode | Ratio | Modality of the track (major or minor). |
| speechiness | Continuous | Measure of the presence of spoken words in the track. |
| acousticness | Continuous | Measure of how acoustic or non-electronic the track is. |
| instrumentalness | Continuous | Predicts whether a track is instrumental based on various musical features. |
| liveness | Continuous | Indicates the presence of an audience in the recording. |
| valence | Continuous | Measure of the musical positiveness of the track. |
| tempo | Ratio | The tempo of the track in beats per minute. |
| features_duration_ms | Continuous | Duration of a specific feature in milliseconds. |
| time_signature | Ratio | The number of beats in each bar of music. |
| start_of_fade_out | Continuous | The time in milliseconds where the fade-out of the track begins. |
| tempo_confidence | Ratio | Confidence level in the accuracy of tempo estimation. |
| time_signature _confidence | Ratio | Confidence level in the accuracy of time signature estimation. |
| key_confidence | Ratio | Confidence level in the accuracy of key estimation. |
| mode_confidence | Ratio | Confidence level in the accuracy of mode estimation. |
| n_beats | Discrete | The number of beats detected in the track. |
| n_bars | Discrete | The number of bars detected in the track. |

Table 1: Data Description

**Artists dataset**

| Name | Type | Description |
|---|---|---|
| id | Nominal | Unique identifier for each artist in the dataset. |
| name | Nominal | The name of the artist or group. |
| popularity | Ratio | A measure of the artist's popularity. |
| followers | Ratio | The number of followers the artist has on the platform. |
| genres | Nominal | The genres associated with the artist's music. |

To run the analysis, the two datasets were combined to exploit the columns **popularity** and **followers** present in the artists' dataset by adding 2 columns, **artists_popularity_mean** and **artists_followers_mean**, to the tracks' dataset. The two new columns represent the mean values of the artists present in the relative track, reported in the Artists' dataset. A new variable (**BPM**) was created which depends on the duration and beats of the song. This way, the information in just one attribute is preserved as opposed to having two, which slightly reduces the dimension of the dataset. It is important to note that some of the **ids** were **repeated** and all the values in the repetitions matched, with the exception of the genre feature. Because this would cause issues for the other tasks to be performed further into the project, the repeated records were merged by combining the two values in the genre column and concatenating the values with a ";" character. Furthermore, the values in the Artists' genres column that were not present in the tracks dataset, were deleted from the Artist's dataset. Finally, a few artists which were in the artists dataset but not present in the tracks dataset were filtered out. The final version of the tabular dataset includes the tracks dataset with the Artists dataset mean value for followers and popularity, without the missing artists, with the BPM column, and with the genre column populated in a way that there are no repeated records.

## 1.2 Distribution of the variables and statistics

While examining the distributional characteristics within the tabular dataset, diverse shapes across the attributes can be observed. Notably, danceability exhibits a skewed normal distribution, while valence demonstrates a distribution closely resembling uniform. Additionally, certain attributes display a pronounced concentration of observations towards either end of the distribution. Overall, the dataset exhibits a well-behaved nature, with distributions that are sufficiently spread out and devoid of excessive concentration at specific points. Noteworthy is the coherence between the continuous nature of continuous attributes, as evident in histograms, and the discrete nature of discrete attributes, as portrayed in bar charts. In the figure 1 one can observe some of the distributions in the present tabular dataset.

## 1.3 Measurement of Central Tendency

| Column | Mean | Mode | Median |
|---|---|---|---|
| duration_ms | 228517.08 | 180000 | 213230.0 |
| popularity | 31.46 | 0 | 31.0 |
| danceability | 0.56 | 0.631 | 0.576 |
| energy | 0.63 | 0.961 | 0.677 |
| loudness | -8.50 | -5.956 | -7.185 |
| speechiness | 0.09 | 0.0324 | 0.049 |

| | | | |
|---|---|---|---|
| acousticness | 0.33 | 0.995 | 0.187 |
| instrumentalness | 0.17 | 0.0 | 5.84e-05 |
| liveness | 0.22 | 0.111 | 0.132 |
| valence | 0.47 | 0.961 | 0.457 |
| tempo | 122.07 | 0.0 | 122.015 |
| features_duration_ms | 228507.62 | 180000 | 213221.0 |
| start_of_fade_out | 220.72 | 120.47093 | 205.55466 |
| tempo_confidence | 0.44 | 0.0 | 0.413 |
| time_signature_confidence | 0.88 | 1.0 | 0.987 |
| key_confidence | 0.49 | 0.0 | 0.507 |
| mode_confidence | 0.51 | 0.0 | 0.521 |
| n_beats | 458.27 | 352.0 | 419.0 |
| n_bars | 117.44 | 97.0 | 106.0 |
| artists_popularity_mean | 48.00 | 51.0 | 49.0 |
| artists_followers_mean | 2271812.49 | 1136992.0 | 184897.0 |

Table 3: Central Tendency of Data

### 1.4 Measurement of Data Dispersion

| Column | Min | Max | Range | Variance | Std |
|---|---|---|---|---|---|
| duration_ms | 8586.0 | 4120258.0 | 4111672.0 | 10356444000.72 | 101766.62 |
| popularity | 0.0 | 95.0 | 95.0 | 463.38 | 21.53 |
| danceability | 0.0 | 0.985 | 0.985 | 0.03 | 0.18 |
| energy | 0.0 | 1.0 | 1.0 | 0.07 | 0.26 |
| loudness | -49.531 | 4.532 | 54.063 | 27.30 | 5.23 |
| speechiness | 0.0 | 0.965 | 0.965 | 0.01 | 0.11 |
| acousticness | 0.0 | 0.996 | 0.996 | 0.11 | 0.34 |
| instrumentalness | 0.0 | 1.0 | 1.0 | 0.10 | 0.32 |
| liveness | 0.0 | 1.0 | 1.0 | 0.04 | 0.20 |
| valence | 0.0 | 0.995 | 0.995 | 0.07 | 0.26 |
| tempo | 0.0 | 243.372 | 243.372 | 907.19 | 30.12 |
| features_duration_ms | 8587.0 | 4120258.0 | 4111671.0 | 10356110182.92 | 101764.98 |
| start_of_fade_out | 8.58667 | 4106.339 | 4097.75233 | 10057.72 | 100.29 |
| tempo_confidence | 0.0 | 1.0 | 1.0 | 0.09 | 0.30 |
| time_signature_confidence | 0.0 | 1.0 | 1.0 | 0.05 | 0.22 |
| key_confidence | -1.0 | 1.454 | 2.454 | 0.06 | 0.25 |
| mode_confidence | -1.0 | 1.0 | 2.0 | 0.03 | 0.18 |
| n_beats | 0.0 | 7348.0 | 7348.0 | 53768.70 | 231.88 |
| n_bars | 0.0 | 2170.0 | 2170.0 | 3798.33 | 61.63 |
| artists_popularity_mean | 0.0 | 100.0 | 100.0 | 322.42 | 17.96 |
| artists_followers_mean | 0.0 | 114163489.0 | 114163489.0 | 55461329704547.18 | 7447236.38 |

Table 4: Summary Statistics

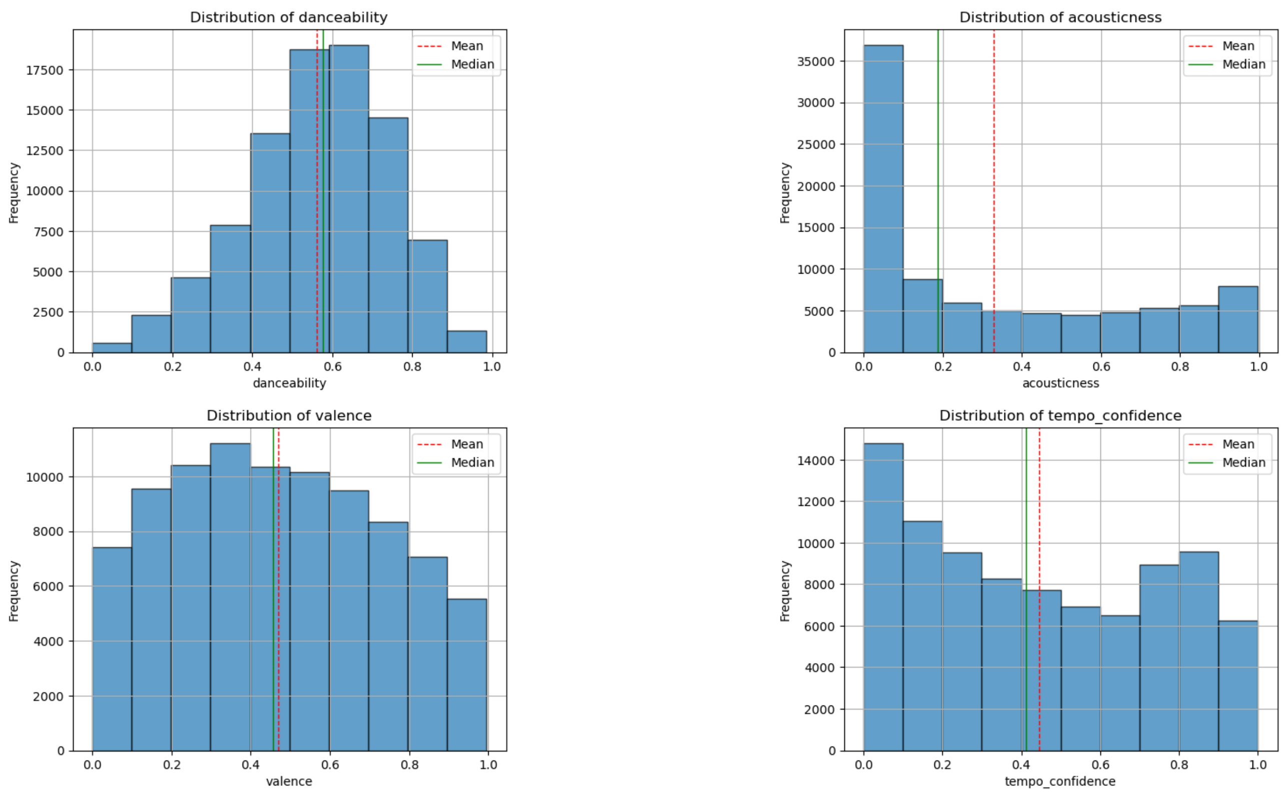


Figure 1: Some relevant features histograms

## 1.5 Assessing data quality

### 1.5.1 Semantic Inconsistencies

Regarding semantic inconsistencies, which means any strange or incorrect values due to human mistakes in the dataset, two cases were taken into consideration: albums with similar names from the same artists, as well albums with the same name from similar artists. For the first case, we set a rule that if the album names were 85% similar, we would flag them as potential inconsistencies. After some research, there were a few cases where albums by the same artists had the exact same name, except for some small changes like one letter being uppercase or lowercase: "Eye of the Storm" vs "Eye Of The Storm". Then, the inconsistent values were replaced with the correct album names. For the second case, the same threshold was used so that if the artist names were 85% similar, they would be flagged as potential inconsistencies. However, no such instances were found.

# 2 Time Series data exploration

After analyzing the guide data in the shared Jupyter notebooks, the decision to trim the length of the time series was made. Initially, the minimum (830), mean (1279.9533), and median (1280) lengths were calculated. Given the close proximity of mean and median values, the distribution was inspected and found that 9,998 out of 10.000-time series were of length 1280, with the remaining 2 being of different lengths (830 and 1263). Consequently, the two time series with atypical lengths were removed from the dataset ("**0DVMaexfdXDz19zUv1zKej_kids.npy**" and "**0P2DLw3xEcOgyYI7Ye4whM_honky-tonk.npy**"). This was followed by the pre-processing of the time series dataset. The files were first organized into 20 different folders, one for each genre. Additionally, a tabular dataset mirroring the time series records by matching IDs was created. However, the IDs were repeated in the tabular dataset and the time series dataset, with 136 repeated IDs identified after inspection. As such, half of the time series with repeated IDs were removed to ensure uniqueness. Before this, it was verified if the time series files with matching IDs were indeed identical. Both full and filtered tabular datasets were updated to reflect the changes, resulting in 9,860 records in the latter. The outcome is a distribution of time series files across 20 folders, as outlined in Table 5. As evident from the table, the time series distribution is no longer uniform. Despite certain genres having approximately 70 missing songs, the dataset appears to maintain a reasonable balance.

| | | | |
|---|---|---|---|
| emo (500) | folk (433) | goth (469) | happy (500) |
| heavy-metal (500) | honky-tonk (500) | j-idol (499) | kids (500) |
| minimal-techno (500) | mpb (496) | new-age (468) | opera (500) |
| piano (497) | progressive-house (500) | salsa (500) | sertanejo (500) |
| sleep (500) | songwriter (500) | synth-pop (500) | world-music (500) |

Table 5: Amount of files for each genre for the time series dataset

### 2.1 Approximation on time series

During classification and clustering tasks, computational challenges became apparent due to the heavy computational load of the time series data. To address this, we opted to implement approximation methods introduced in class were implemented for both clustering and classification purposes.

**PAA**: PAA approximation was conducted by testing different fractions of the total length (1280) of the time series. Upon analysis, it was observed that splitting the time series total length into 3 and 4 intervals (resulting in 320 and 256 intervals, respectively) provided satisfactory results, effectively capturing peaks and lows. For efficiency, the decision was made to proceed with the time series utilizing 256 intervals.

**SAX**: Similar to PAA, SAX was implemented with the same intervals. However, an alphabet size list was iterated through, testing values from 3 to 7. After evaluation, a decision was made to use an alphabet size of 5 and continued with 256 intervals, maintaining consistency with PAA. While SAX graphs may not closely follow the original time series line, this behavior is expected as SAX aggregates points into straight lines.

## 3 Time Series Classification

The time series data underwent scaling post-approximation, utilizing the StandardScaler class. K-fold cross-validation was not employed due to the already computationally demanding task and the absence of pre-made methods to perform it on this classification task. Instead, all train-test splits were conducted using an 80/20 ratio. For the classification task, models were developed to predict the **genre** column of the dataset, as well as various continuous features. Continuous variables were discretized into five bins based on their respective columns' minimum and maximum values. As such, the classification was conducted on the following columns: **genre**, **popularity**, **tempo**, **liveness**, **speechiness**, **acousticness**, **artists_popularity_mean**, and **time_signature**.

## 3.1 Derivative Dynamic Time Warping

The additional distance metric employed, Dynamic Derivative Time Warping (DDTW), is specifically designed to compute nonlinear alignments between two-time series efficiently. Unlike simpler metrics such as Euclidean distance, DDTW accounts for temporal discrepancies like delays or advances. This algorithm is particularly beneficial when dealing with sequences that share similar component shapes but lack perfect alignment in the time domain. To achieve enhanced alignment, it becomes necessary to "deform" the time axis of one or both sequences. Although DTW is used for this purpose, it may produce sub-optimal results by overlooking clear sequence alignment. DDTW addresses this limitation by focusing solely on the local derivatives of the data, rather than the raw data itself, often resulting in superior alignments. DDTW introduces a modification that disregards the Y-values of data points, prioritizing the higher-level feature of "shape." This adjustment allows for accommodating variations in sequence shapes while minimizing the influence of specific data points. Figure 2, extracted from the relevant paper, illustrates the alignment process using DDTW, which facilitates shape-based alignment.

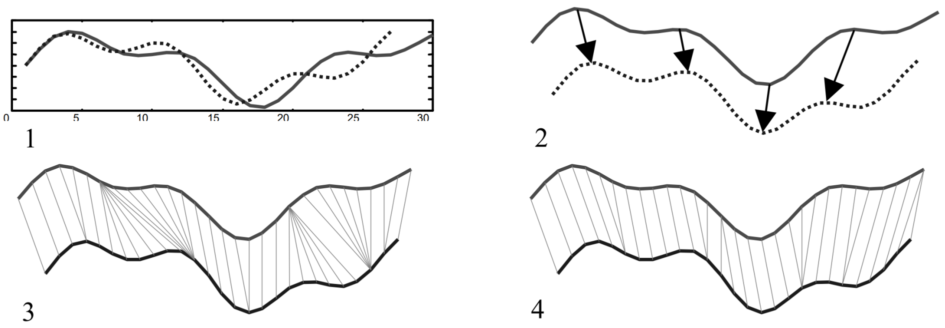


**Figure 6: 1)** Two artificial signals. **2)** The intuitive feature to feature warping alignment. **3)** The alignment produced by classic DTW. **4)** The alignment produced by DDTW.

Figure 2: DDTW paper figure for showing differences in the

The decision to utilize DDTW is prudent, particularly when time series shifting is absent. By prioritizing shape-based alignment, DDTW offers a more robust approach capable of accommodating variations in underlying patterns.

## 3.2 K Neighbors Classifier

For the KNN implementation, the KNeighborsTimeSeriesClassifier class was selected and a range of values for k was chosen, including [5, 7, 9, 17, 20, 23, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97]. The initial values were intentionally low; some extra values were added around the number of genres (20), and another array of subsequent values was centered around the square root of the number of samples in the final time series dataset (9860). To facilitate these computations, pre-computed matrices of distances were used both within X_train and between X_train and X_test. This preparation step enabled swift calculations. Given 2 approximations, 3 distance metrics (Euclidean, DTW, and DDTW), 8 target columns, 16 values of k, and 4 evaluation metrics (accuracy, precision, recall, and F1 score), we obtained a total of 768 models and 3072 metric results. Due to the comprehensive results and page space constraints, we decided to report only a summary of the best three results we obtained from this model for each of the target variables and approximation methods.

**acousticness**

For the approximation method PAA, the top three results were achieved with $k$ values of 81, 83, and 85, all using DTW as the distance metric. The accuracy ranged around 63.8%, with precision and recall around 53-53.4% and 33.4%, respectively. For the approximation method SAX, the top three results were achieved with $k$ values of 83, 79, and 81, all using DTW as the distance metric. The accuracy ranged around 55.1-55.4%, with precision and recall around 38.1-39.4% and 27-27.2%, respectively.

**artists_popularity_mean**

The top three results for both PAA and SAX approximation methods had relatively lower accuracies, ranging around 45-45.4%, with precision and recall values varying across different $k$ values.

**genre**

The accuracy was relatively low for both PAA and SAX approximation methods, ranging around 18-29.1%, with precision and recall values stable across different $k$ values, all around 30%. Only in the case of the genre as the target variables, are those results more promising than the ones obtained for the other target variables since, contrary to the other target variables, this class has 20 possible labels.

**liveness**

The top three results for the PAA approximation method were achieved with $k$ values of 9, 17, and 20, all using DTW as the distance metric. The accuracies ranged around 59.8-63.2%, with precision and recall around 53.9-54.8% and 21-21.5%, respectively. For the SAX approximation method, the top three results were achieved with $k$ values of 97, 95, and 93, all using DDTW as the distance metric. The accuracies were consistently around 65%, with precision and recall around 42.3% and 16.7%, respectively. These results are interesting since, when changing the approximation method the best distance changes and best values of k change dramatically. The results obtained are the same anyway.

**popularity**

The accuracy of PAA and SAX approximation methods was relatively low, ranging around 39.4-43.8%, with precision and recall values varying across different $k$ values.

**speechiness**

For the PAA approximation method, the top three results were achieved with $k$ values of 17, 97, and 95, using DTW and DDTW as the distance metrics. The accuracies were high, around 93.9-94%, with precision and recall around 88.2-88.9% and 17.9%, respectively. For the SAX approximation method, the top three results were achieved with various $k$ values and distance metrics, with accuracies consistently around 93.9%, precision around 88.6-88.9%, and recall around 17.9%.

**tempo**

For both PAA and SAX approximation methods, the accuracies ranged around 51.7-53.2%, with precision and recall values varying across different $k$ values. It is also noticeable that, when using the PAA approximation the distance that yelds the best results is dtw but when using the SAX approximation the better distance is the euclidian one.

**time_signature**

For both PAA and SAX approximation methods, the accuracies were relatively high, consistently around 85.9%, with precision and recall values varying across different $k$ values, since the best results for the SAX approximation were given by a k of 23 using dtw as distance but the remaining ones with higher values and using ddtw; in the meanwhile for PAA the best results were all obtained using a high value of k and dtw as distance.

## 3.3 Other approaches

### 3.3.1 Rocket

The experimentation involved two parameters: minirocket and multirocket. A consistent random state was set during classifier creation. As for the K neighbors classifier here is a summary of the results:

**Acousticness**: For PAA approximation, both minirocket and multirocket show similar performance. However, under SAX approximation, multirocket outperforms minirocket across all metrics even if not with that big of a difference.

**Artists Popularity Mean**: Minirocket exhibits better performance compared to multirocket for PAA and the opposite is true for the SAX approximations.

**Genre**: Results for genre prediction are poor (all metrics are around 20%) for the SAX approximation, with no significant difference between minirocket and multirocket but, regarding the PAA approximation, the results are much more promising with all the metrics scoring around 40%.

**Liveness**: There is no meaningful difference between the multi rocket and mini rocket for both approximation methods, even tho, for both the approximations, we get high values of accuracy and precision but low values of recall.

**Popularity**: Popularity prediction results are low across all metrics, approximations, and transformations.

**Speechiness**: As for liveness, we get excellent results for accuracy and precision across the board, but we get low values for recall.

**Tempo**: There is no clear winner between mini rocket and multi rocket transformations for PAA and SAX approximations, as the results are very similar.

**Time Signature**: Multirocket shows slightly better results than minirocket for both PAA and SAX approximations, but the same problem with recall is also present here.

**In summary:** regarding acousticness, multirocket outperforms minirocket under SAX approximation, while minirocket performs better for PAA. Conversely, for artist popularity means, minirocket excels with PAA, while Multirocket performs better with SAX. Genre prediction yields poor results under SAX but promising results with PAA, albeit with no significant difference between the two transformations. Liveness and speechiness show high accuracy and precision but low recall, with no meaningful difference between the two transformations. Popularity prediction results are generally low. Tempo and time signature analysis show similar performance between minirocket and multirocket, with slightly better results for multirocket in the latter. However, recall remains a challenge across several metrics.

### 3.3.2 Canonical Interval Forest

In the Canonical Interval Forest (CIF) investigation for time series classification, a systematic approach was adopted to assess the impact of varying the number of estimators on model performance. Initially, a wide range of estimator values (100, 300, 500) was explored, with the genre variable as the primary target for evaluation. Upon observing promising outcomes with 100 estimators, further experimentation proceeded with 300 estimators. Model creation and fitting, a computationally intensive process, required approximately 30 minutes, whereas prediction on the test dataset demanded significantly more computational resources, approximately 12 hours compared to 7 hours for the 100 estimators. Given the marginal difference in results between 100 and 300 estimators, the exploration was not extended to 500 estimators. Instead, a more conservative range (25, 50, 75) was chosen for subsequent evaluations, both for the genre variable and other pertinent targets. Notably, the performance of models trained with these lower estimator values exhibited comparability to those trained with higher values, particularly evident in the genre column. Consequently, these lower values were deemed suitable for further exploration across other target variables. A concise summary of the findings is presented here:

**Acousticness:** For the PAA model, as the number of estimators increases, both accuracy and precision improve slightly while recall fluctuates. F1 score improves with the number of estimators up to a certain point and then slightly decreases, in turn, SAX model generally performs worse compared to PAA, with lower accuracy and precision. The F1 score also tends to be lower across different numbers of estimators.

**Artists Popularity Mean:** For PAA Model, both accuracy and precision increase as the number of estimators increases. However, recall and F1 score show less consistent patterns with varying numbers of estimators, The performance of the SAX model is generally similar compared to PAA, but with slightly lower accuracy, precision, recall, and F1 score.

**Genre:** Across different numbers of estimators, the PAA model shows moderate performance with varying but somewhat consistent accuracy, precision, recall, and F1 score, similar to other target variables, the SAX model performs worse compared to PAA with lower accuracy, precision, recall, and F1 score. The difference between the two approximations is much sharper here, with a difference of around 20% across all the metrics, and PAA comes out as a winner.

**Liveness:** The PAA model performs well with high accuracy and precision. However, recall is relatively low, indicating some difficulty in correctly identifying instances of liveness; the two approximations are strictly comparable.

**Popularity:** The PAA model exhibits moderate performance with varying but somewhat consistent accuracy, precision, recall, and F1 score across different numbers of estimators. The SAX model's performance is poorer compared to PAA, with lower accuracy, precision, recall, and F1 score.

**Speechiness:** The PAA model demonstrates high performance with consistently high accuracy, precision, and F1 score across different numbers of estimators, but the recall is very low. Surprisingly, the SAX model shows performance identical to the PAA model's accuracy, precision, recall, and F1 score.

**Tempo:** The PAA model generally exhibits moderate performance with varying but somewhat consistent accuracy, precision, and F1 score across different numbers of estimators but with the recall, as usual, being very low. Similar to other target variables, the SAX model performs worse compared to PAA, with slightly lower accuracy, precision, recall, and F1 score.

**Time Signature:** The PAA model shows relatively high accuracy and precision, but lower recall, indicating some difficulty in correctly identifying instances of time signature, The SAX model's performance is poorer compared to PAA, with slightly lower accuracy, precision, recall, and F1 score.

Overall, the PAA model tends to outperform the SAX model across all target variables, exhibiting higher accuracy, precision, recall, and F1 score. However, both models show varying performance depending on the target variable and the number of estimators used.

## 3.4 Overall results

Overall, the three models—K Neighbors Classifier, Rocket, and Canonical Interval Forest—demonstrate decent results in terms of accuracy and precision across multiple target variables and approximation methods. However, a notable trend emerges across all models: while accuracy and precision are often satisfactory, recall tends to be consistently low.

What's particularly striking is that these models achieve these results solely based on the time series data of the songs, without any additional information. This underscores the complexity of the task and highlights the models' ability to extract meaningful patterns from temporal data alone.

Across the K Neighbors Classifier, Rocket, and Canonical Interval Forest models, the evaluation across different target variables and approximation methods reveals consistent challenges with recall. This indicates a common difficulty in correctly identifying instances of certain attributes, despite achieving overall reasonable accuracy and precision.

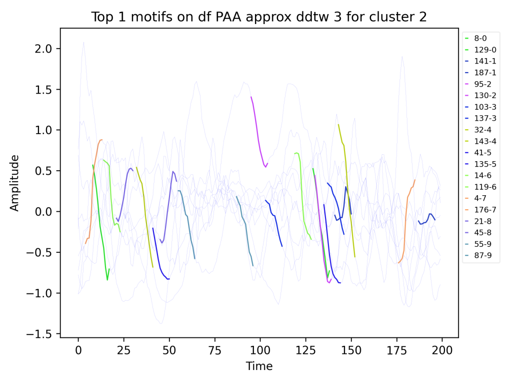
## 3.5 Motifs & Shapelets

Motif discovery plays a crucial role in time series analysis. It facilitates the extraction of knowledge from time series data, enabling the formulation of association rules, classification models, and anomaly detection techniques. In this study, motif discovery and discord detection were applied to a dataset of time series songs pre-processed using Piecewise Aggregate Approximation (PAA). Specifically, the focus on subgroups identified through clustering algorithms, allowing for the analysis of motifs and discords within sets of similar songs. This approach facilitated the identification of highly similar motifs across different songs, revealing recurring patterns within the data.

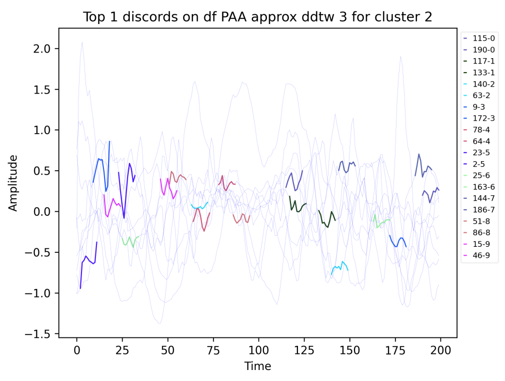
Given the potential for a significant increase in the number of motifs and discords per time series, leading to cluttered visualizations, the decision was made to extract only the top motif and discord for each time series. This resulted in motif-discord pairs for further analysis. While the visualization of potential neighbors for these motifs and discords was an option, it was decided to set their values to zero for visualization purposes to maintain clarity. The window size for analysis was chosen to be 10. This window size encompasses approximately 5% of the data and strikes a balance between avoiding overly large windows that capture minimal patterns and overly small windows that identify almost any shape as a motif.

The results reveal a consistent behavior for motifs within this subgroup. Motifs generally exhibit a steep positive or negative slope and tend to occur at the beginning or end of the time series. Discords, on the other hand, manifest as more erratic subsections with frequent changes in direction and are distributed throughout the time series, as evident from the visualizations. These findings illuminate the characteristic behaviors of motifs and discords within this subgroup, highlighting both the similarities and the deviations in time series shapes. It is noteworthy to mention shapelets as a subsequent approach that leverages similar time series patterns for model fitting in classification tasks.

When extracting shapelets, the goal was to find subsequences that are discriminative for different categories within the time series. The time series utilized underwent a preprocessing step involving Piecewise Aggregate Approximation (PAA) for dimensionality reduction, ensuring that duplicates were eliminated from the dataset. The dataset was separated into training and testing sets, with a test size of 20%. To define the shapelet Parameters and extract them the 'tslearn' library was utilized, specifically the 'LearningShapelet' and 'grabocka_params _to_shapelet_size_dict' functions. After fitting the model to the training data, seven shapelets were observed to be extracted. These shapelets were then aligned to the matching time series and visualized on plots. Due to paper space constraints, only the 7th shapelet is visualized in Figure 4.



(a) Top motifs cluster 2



(b) Top discords cluster 2

Figure 3: Motifs and discords in cluster 2

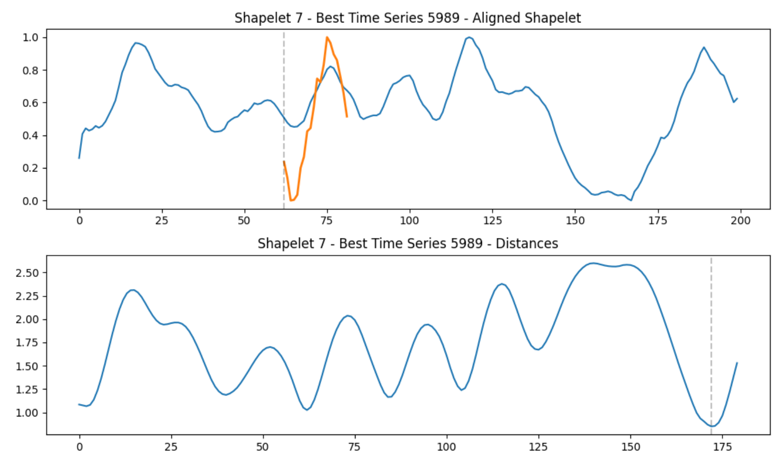


Figure 4: 7th Shapelet

Based on the time series that exhibited the closest match, matching the ts index to the labels, it was determined that the genres represented by the shapelets were respectively: **Synth-pop**, **Songwriter**, **Kids**, **Emo**, **Heavy-Metal**, **Opera**, **Kids**.

In essence, Motifs and Shapelets are methodologies designed to encapsulate the inherent temporal dynamics within a time series dataset. While both techniques endeavor to delineate the principal features of the data, they diverge in their utilization of the extracted insights. Motifs afford a more accessible avenue for visual analysis and knowledge extraction, facilitating interpretability through the direct identification of recurring patterns. Moreover, motifs lend themselves well to clustering analyses, wherein discovered motifs are spatially represented, thus facilitating the discernment of overarching trends.

Conversely, Shapelets primarily find application in classification endeavors, leveraging their discriminative power to categorize instances within the dataset effectively. As such, Shapelets are adept at discerning subtle distinctions between different classes or categories embedded within the temporal data.

In conclusion, both Motifs and Shapelets harness the inherent shapes and visual cues present within the data to distill meaningful insights and deliver actionable results. While Motifs excel in facilitating visual exploration and pattern recognition, Shapelets specialize in discerning class boundaries, thereby underpinning their respective utility in distinct analytical contexts.

# 4 Time series clustering

For the clustering task, the same approximations used for classification (SAX and PAA) were employed.

The two algorithms chosen were K-means on both the approximations. Additionally, the same distances used for classification (Euclidian, DTW and DDTW). Hierarchical clustering with different parameters was also performed on the dataset composed of the features extracted from the time series dataset.

## 4.1 K-means

For the K-means algorithm, six values of K were selected. Among them, three are considered low values, namely 3, 9, and 20 (where 20 corresponds to the total number of genres). The remaining three values were chosen around the square root of the total number of time series, which is 9860, namely 89, 91, and 93. Following the execution of the algorithm, attempts were made to generate basic plots (cross-tab plots) based on the columns of the parallel dataset, specifically "genre," "explicit," "key," "mode," and "time signature." However, the outcomes did not yield satisfactory results. Subsequently, efforts were directed towards evaluating the silhouette and sum of squared errors (SSE) scores of the algorithm. Regrettably, the application of K-means for high values of K, particularly when utilizing dynamic time warping (DTW) and derivative dynamic time warping (DDTW) as distance metrics, proved unfeasible. As evidenced by the Euclidean results presented in Table 6, outcomes for high values of K are unpromising, with particularly discouraging results observed for the symbolic aggregate approximation (SAX) technique.

| Approximation | Method | 3 | 9 | 20 | 89 | 91 |
|---|---|---|---|---|---|---|
| PAA approx | Euclidean | 0.41 | 0.28 | 0.15 | 0.03 | 0.00 |
| | DTW | 0.42 | 0.28 | 0.16 | - | - |
| | DDTW | 0.11 | 0.11 | 0.11 | - | - |
| SAX approx | Euclidean | 0.10 | 0.02 | -0.00 | -0.05 | -0.04 |
| | DTW | 0.02 | - | - | - | - |
| | DDTW | - | - | - | - | - |

Table 6: Knn silhouette score results comparison

The results in Table 6 are approximated and the ones that are not reported have either not been calculated since they required too long or all the samples had been clustered in the same cluster and the silhouette score was not calculated.

Regarding SSE score we had the results reported in Table 7.

| Approximation | Method | 3 | 9 | 20 | 89 | 91 |
|---|---|---|---|---|---|---|
| PAA approx | Euclidean | 93504074.64 | 81038068.98 | 78379879.66 | 76410356.42 | 76338671.97 |
| | DTW | 925855102003.38 | 757139496386.88 | 722912173097.75 | - | - |
| | DDTW | 396392077726.64 | 396392077726.64 | 396392077726.64 | - | - |
| SAX approx | Euclidean | 191931.36 | 188192.28 | 185822.66 | 180698.60 | 180644.99 |
| | DTW | 3451185.12 | - | - | - | - |
| | DDTW | 2101292.04 | 2101292.04 | 2101292.04 | 2101292.04 | 2101292.04 |

Table 7: Knn SSE score results comparison

As we can see from Table 6 and Table 7, the most promising results were obtained when using the PAA approximation with distance either Euclidean or DTW and a k of either 3 or 9.

## 4.2 Feature based clustering

For the feature-based clustering, the class sktime.transformations.panel.tsfresh. TSFreshFeatureExtractor was used, and as a default feature calculator, the parameter comprehensive. This calculation extracted a total of 783 features from the original time series files (no approximations). To reduce the dimensionality of the dataset and eliminate some irrelevant features, the total number of features was filtered down to the top 100 based on variance. This was done, particularly, to remove features that were constant for all rows of the dataset. After obtaining the dataset with 100 features, a combination of hierarchical clustering algorithms was employed, and the results are presented in Table 8. The data was scaled using the class sklearn.preprocessing.StandardScaler.

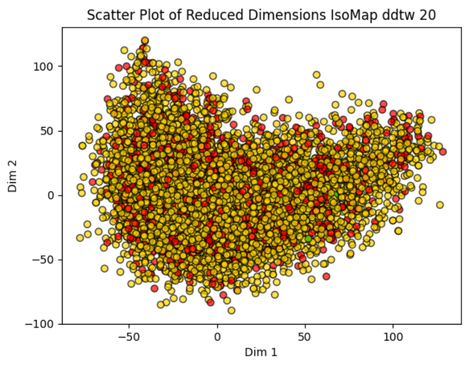
| **Linkage method** | **criterion** | **5** | **25** | **50** | **75** | **100** | **125** | **150** | **175** | **200** | **225** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Centroid | Distance | -0.212 | -0.744 | -0.542 | -0.401 | -0.401 | -0.401 | -0.366 | -0.308 | -1 | -1 |
| | MaxClust | -0.401 | -0.647 | -0.751 | -0.827 | -0.873 | -0.879 | -0.878 | -0.876 | -0.871 | -0.878 |
| Complete | Distance | -0.157 | -0.775 | -0.546 | -0.416 | -0.399 | -0.400 | -0.364 | 0.292 | 0.292 | 0.292 |
| | MaxClust | -0.400 | -0.649 | -0.696 | -0.771 | -0.788 | -0.833 | -0.832 | -0.797 | -0.795 | -0.794 |
| Single | Distance | -0.226 | -0.704 | -0.528 | -0.401 | -0.401 | -0.308 | -1 | -1 | -1 | -1 |
| | MaxClust | -0.407 | -0.649 | -0.747 | -0.780 | -0.860 | -0.860 | -0.866 | -0.865 | -0.863 | -0.860 |
| Average | Distance | -0.162 | -0.721 | -0.546 | -0.405 | -0.401 | -0.401 | -0.366 | -0.308 | -1 | -1 |
| | MaxClust | -0.401 | -0.632 | -0.704 | -0.845 | -0.860 | -0.865 | -0.870 | -0.874 | -0.869 | -0.872 |

Table 8: Feature based clustering silhouette results comparison

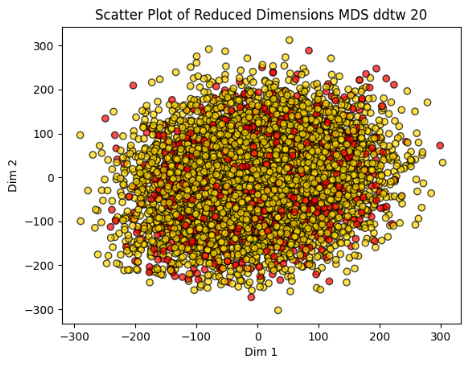
As usual, the values -1 in the table are the ones where all the samples are put into the same cluster. As observed, the clustering results, particularly when considering the silhouette score, were notably poor. Consequently, it was decided not to pursue further investigation into these clustering results.

## 4.3 Dimensionality Reduction

The objective of dimensionality reduction for clustering time series was to reduce the number of features in the data while preserving the most relevant information. To achieve this, various methodologies were employed, specifically Multidimensional Scaling (MDS), Isometric Mapping (IsoMap), and t-Distributed Stochastic Neighbor Embedding (t-SNE). These algorithms were chosen as they are more compatible with our previously computed dissimilarity matrices obtained through DTW. For each technique, two versions were tried: one using DTW and another using Sakoe-Chiba constrained DTW, all displayed in Figure 5. The results obtained from these techniques were not satisfactory. As such, attempts were made to perform dimensionality reduction in three dimensions, yet the resulting shape remained consistent with the 2D representation, indicating a lack of improvement. When analyzing these results, it was not surprising, as even when evaluating clusters through entropy and silhouette, the results were not the most satisfactory.



(a) IsoMap



(b) MDS

Figure 5: Dimensionality Reduction Techniques for Time Series Clustering

## 5 Sequential Pattern Mining

In this section, the methodologies employed in the analysis of time series data through the application of the Generalized Sequential Patterns (GSP) algorithm are explained. The primary objective of this endeavor is to discretize the time series to identify recurring sequential patterns within the dataset and to gain insights into underlying trends in the data. The time series utilized underwent a pre-processing step involving Piecewise Aggregate Approximation (PAA) for dimensionality reduction, ensuring that duplicates were eliminated from the dataset, as well as a conversion mechanism from float to string was implemented for the time series to conform to the input requirements of the GSP algorithm. The algorithm was run by specifying the following parameters: minsup $\rightarrow$ 0.85, maxgap $\rightarrow$ 2 and mingap $\rightarrow$ 1. These parameters were chosen for the following reasons:
1.**Minsup**: It was decided that patterns must occur with a minimum frequency of 85% across the dataset to be deemed significant, as setting it to an even higher value would be a very strict criterion for pattern significant.
2.**Maxgap**: The value of 2 was chosen so that there can be at most two-time steps of interruption or gap between consecutive elements within a pattern. This is a moderate value that allows to capture sequential patterns that exhibit temporal dependencies while accommodating minor temporal variations within the data.
3.**Mingap**: The value of 1 was chosen so that there must be at least one-time step of continuity between consecutive elements within a pattern. This ensures that the elements within a pattern are adjacent in time, promoting the detection of continuous sequences, not overly fragmented, in the data.
After retrieving the sequential patterns from GSP, sequences containing only one element were filtered out to focus on more meaningful patterns. The results, including the sequences and their corresponding support counts, are displayed in Figure 6.

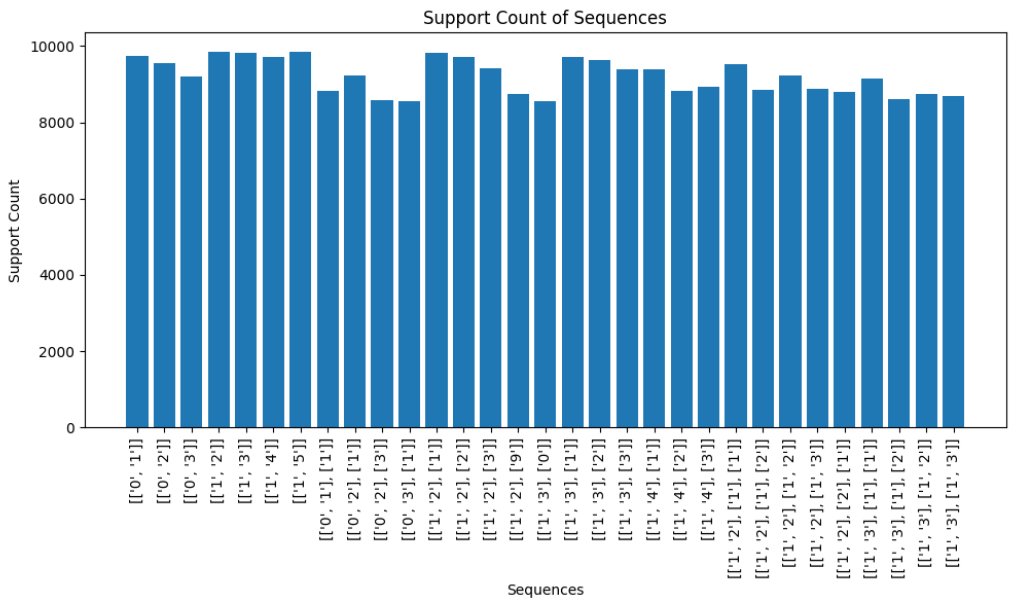


Figure 6: GSP Sequences

Then, the time series was transformed from integer representations back to their original float values, followed by plotting these time series data to visualize the sequential patterns present within them. It was decided to only display patterns that had 3 or more values, as shown in Figure 7.

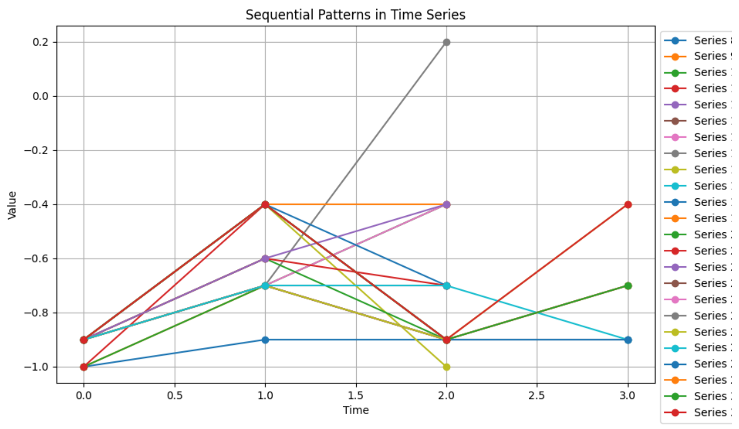


Figure 7: Sequential Patterns

From this plot, it can be understood that in the time series, patterns with a maximum of 4 values and with a maximum time span of 3 are observed. This indicates that the dataset captures sequential behaviors within a relatively short time frame.

## 6 Outliers & Anomalies

Regarding outliers and anomalies, the diversity of methods that can be used for handling them is very broad, with so many tools we wanted to implement a methodology of nested evaluation, where we apply a variety of methods with certain levels of outlier contamination (very low) and we try to match the results of the methods, to create a sort of "Ensemble" where only the outliers flagged by all the methods are actually defined as outlier observations for us, in this sense, we increase the contamination in each level to allow a higher number of observations to pass, now regarding the methods implemented we have: Elliptical Envelope (EE), Outlier scores with KNN based on distances (OKNN), Local outlier factor (LOF), Standard DBSCAN, Cluster based local outlier factor (CBLOF), Angle based outlier degree (ABOD), Feature bagging (FEABAG), Lightweight on-line detector of anomalies (LODA), Isolation forest (IF). In total, we implemented an ensemble of 9 outlier methods to determine the number of outliers we wanted to accept.

The number we were aiming for is around 1% of the total data, with this in mind we ran three rounds of outlier ensemble filtering, the first one with contamination factors of 1%, the second with 10%, and the third one with 20%, this high contamination factors account for the very few values that will be matched in the 9 methods inside the ensemble. The results in each of these rounds are as follows: with 1% contamination, we found 0 matching outliers flagged by the ensemble, with 10% contamination we found 163, and with 20% 715. With these results in mind, we decided that the final ensemble with 20% of contamination in each one had the best tradeoff since the total matching (715) is close to a 1% outlier percentage for the whole data set.

With the results of these ensembles, we proceed to remove them from the data set and create dimensionality reductions to analyze the new dataset and the relationship between the outliers deleted and the dataset. Since the data is quite complex due to a large number of attributes and an observation count close to 90,000, proceeding with a straight dimensionality reduction of MDS, IsoMap, or t-SNE, was practically impossible with the hardware we have, for this reason, we decided to cluster the data into a big number (k=5,000 and k=10,000) this was made to obtain the medoid of each cluster and represent the data in the best way possible for the dimensionality reduction, then we proceed to reduce the data and plot the representation of the data against the outliers, one can observe the results in 2D in Figure 8 for a reduction made with t-SNE since is the most graphically explainable found in our case. The visualization show the blue dots as the medoids and the green x's as the outliers that were trimmed, we can see that the main center of the data appears to be untouched by the outliers, as they remain mainly in the outskirts of the shape.

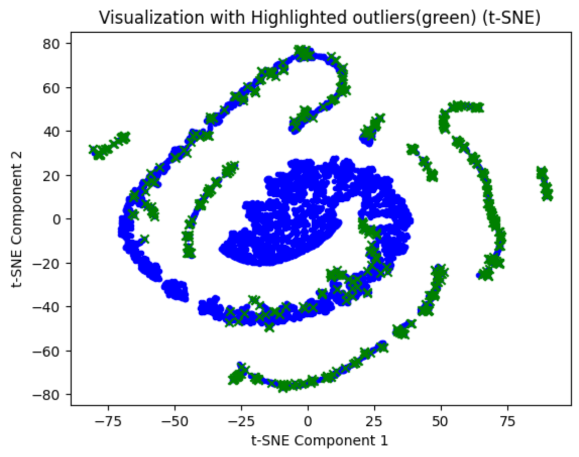


Figure 8: t-SNE Dimensionality Reduction on Kclustering analysis for outliers

## 7 Imbalanced learning

For the imbalanced learning task, we conducted experiments with various algorithms focusing on the target column. The initial distribution of the values for the feature key is illustrated in

Figure 9.

Here are the results after balancing the values using a preliminary 80/20 split for model evaluation:

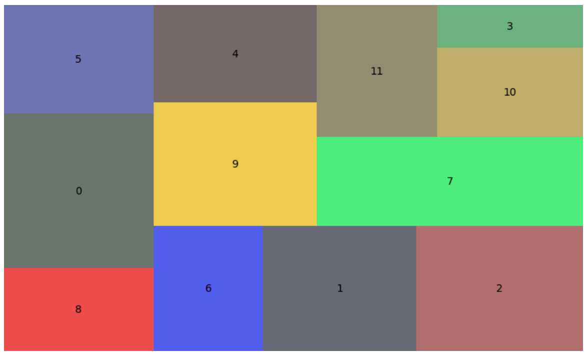


Figure 9: Initial distribution of values for the feature key (see footnote 0)

## 7.1 Undersampling

**Random Under Sampler:** Resampled dataset shape Counter({0: 2189, 1: 2189, 2: 2189, 3: 2189, 4: 2189, 5: 2189, 6: 2189, 7: 2189, 8: 2189, 9: 2189, 10: 2189, 11: 2189})

**Condensed Nearest Neighbour:** Resampled dataset shape Counter({7: 3202, 0: 3133, 2: 3069, 9: 2979, 1: 2822, 5: 2634, 4: 2617, 11: 2549, 6: 2305, 10: 2299, 3: 2189, 8: 2138})

**Tomek Links:** Resampled dataset shape Counter({7: 4911, 0: 4811, 2: 4233, 9: 4147, 1: 4100, 5: 3314, 11: 3204, 4: 3128, 6: 2764, 10: 2685, 8: 2569, 3: 2189})

**Edited Nearest Neighbours:** Resampled dataset shape Counter({3: 2189, 0: 461, 7: 397, 5: 323, 1: 302, 9: 297, 11: 289, 6: 258, 8: 257, 2: 201, 10: 192, 4: 152})

**Cluster Centroids:** Resampled dataset shape Counter({0: 2189, 1: 2189, 2: 2189, 3: 2189, 4: 2189, 5: 2189, 6: 2189, 7: 2189, 8: 2189, 9: 2189, 10: 2189, 11: 2189})

## 7.2 Oversampling

**Random Over Sampler:** Resampled dataset shape Counter({7: 8369, 5: 8369, 0: 8369, 10: 8369, 1: 8369, 6: 8369, 11: 8369, 9: 8369, 4: 8369, 2: 8369, 8: 8369, 3: 8369})

**SMOTE:** Resampled dataset shape Counter({7: 8369, 5: 8369, 0: 8369, 10: 8369, 1: 8369, 6: 8369, 11: 8369, 9: 8369, 4: 8369, 2: 8369, 8: 8369, 3: 8369})

## 7.3 Results

After balancing the dataset with all algorithms, we trained decision trees with different parameters to find the best-performing model and used the 20% dataset from the previous split to evaluate them.

The features used for training models are: ['mode', 'energy', 'acousticness', 'loudness', 'danceability', 'valence', 'speechiness', 'n_beats', 'time_signature']. We experimented with the decision tree parameter **class_weight** set to balanced, and in different instances, with the **adjusted predict method** using a range of thresholds ([0.1, 0.3, 0.5, 0.7, 0.9]). The general parameters explored are (the just mentioned ones were added on top of those): Criteria: ['entropy', 'gini'], Max Depths: list(np.arange(2, 3)) + [None], Min Samples Leaf: [0.1, 0.2, 1, 2, 3, 4, 5, 6], Min Samples Split: [0.05, 0.1, 0.2], CCP Alphas: list(np.arange(0.01, 0.1)). The results were collected in a dictionary (parameter combinations: metrics), and sorted to see which parameters yielded the best metrics. Interestingly, the results remained, mostly, consistent across various balancing methods and parameters. The most effective parameters combination, or

[0]Class distribution: 8: 5510, 0: 10243, 5: 7231, 6: 6091, 1: 8486, 2: 9240, 9: 8915, 4: 7069, 7: 10461, 11: 7036, 10: 5807, 3: 2737. Min and max values: 2737, 10461

those closely related, were consistently: Criterion: entropy, Max depth: 2, Min sample leaf: 0.1, Min sample split: 0.05, CCP alpha: 0.01. Regarding evaluation metrics, results hovered around: Accuracy: 0.14, Precision: 0.045, Recall: 0.116, F1 Score: 0.067. While writing the code and testing it to run it on the full dataset we used a cut of the dataset with a hold-out method on just 5000 samples which yielded significantly better results, which motivated us towards further exploration to check its efficacy in the advanced classification task. To balance the dataset, we aimed to reach the mean of the minimum and maximum class values for the target variable ((2737 + 10461)/2 = 6599). Given the observed results and the improved performance with a smaller dataset during testing, we opted to use either the Cluster Centroids method or the RandomUnderSampler method, both reducing the number of samples for each key to 2189 (with 2737 being the minimum samples per class). We chose Random Undersampler due to its stochastic nature, which has shown efficacy in this kind of task.

## 8 Advanced classification

As suggested in the guidelines we decided to use the same target variable for the advanced classification tasks (key). To chose the features to use to train the various models we used the SelectKBest class from sklearn feature selection module. We found that the most influential ones are, in order: mode, energy, acousticness, loudness, danceability, valence, speechiness, n_beats, time_signature, etc. We decided to use the ones just reported for our classification. Since we are using the balanced dataset for train and validation the test dataset used for those models was obtained by the remaining indexes in the original dataset.

### 8.1 Logistic Regression

For the logistic regression model we initially used this list of parameters and all their combinations. penalties = ['l2', None], tolerances = [1e-4, 1e-3, 1e-2], Cs = [0.1, 1.0, 100.0], solvers = {'l2' : ['lbfgs', 'saga'], None : ['lbfgs', 'saga'] }, maxIters = [10, 100, 200]. We also decided to use the holdout method and to the various combinations of parameters, we added various percentages of the dataset ([0.1, 0.25, 0.5, 0.75, 1]). We did not notice meaningful differences in the results obtained with the various holdouts so we won't report on them in the next paragraphs. We also implemented a 10-fold cross-validation and we're reporting the mean values for the chosen metrics here. We obtained the best mean results on the validation dataset using the following parameters: penality: l2, tolerance: 0.01, C: 1.0, solver: saga, maxIter: 100 and the related results are: accuracy: 0.1257, precision: 0.1239, recall: 0.1287, f1Score: 0.1107. As one can notice the results are not very satisfactory due to their low values but considering that the target feature has 12 possible values we still get better results than random (1/12=0.08333). To perform parameters tuning we used values around the best previously obtained results to train new models on the train dataset. Those are the new parameters we used: penalty: l2 (we kept it fixed since it was, consistently, in the best results), Cs = [1, 0.8, 3], tolerances = [0.0001, 0.00008, 0.0003], solvers = ["saga", "lbfgs"] (since we got some mixed results for them in the train part), maxIters = [80, 100, 120] (this parameter is very divided since up until the 0.25 holdout it is better when it's 10 but since we are now using the whole dataset it's better to use the 100 value). After running those combinations of new parameters we found that the results were still very close to the previous ones even with the new parameters; more in detail: the penalty, the solver, and the tolerance always stay the same for the best results (respectively: l2, saga, 0.01). Instead, the maxIter and the C values are all over the place, the possible values we had for those are 3 each (6 in total) and in the first 10 best results they are all present with almost the same frequency (3, 3, 4 respectively). The best final results were obtained with the following list of parameters Penalty: l2, tolerance: 0.0003, C: 3, solver: lbfgs, maxIter: w1100 and the results on the test dataset are the following: accuracy: 0.1252, precision: 0.1561, recall: 0.1106, f1Score: 0.1129.

## 8.2 Support Vector Machines

In this section, we aimed to predict the feature "key" from various other features of our dataset using Support Vector Machines (SVM). The process involved dimensionality reduction through t-SNE and the use of grid search for hyperparameter tuning to optimize the SVM models. For both Linear and Non-Linear SVM, we applied t-SNE with two components to the training data, and the visualization is depicted in Figure 10.

### 8.2.1 Linear SVM

For Linear SVM, a grid search with 5-fold cross-validation was performed to find the best hyperparameters, where the parameter grid included these possible hyperparameter values: - **'C'**: [0.001, 0.01, 0.1, 1, 10, 100] - **'multi_class'**: ['ovr', 'ovo'] From the results, we found that the best model was the one with C: 0.001 and multi_class: OVR (one vs rest). The results suggest that for the data using a linear SVM model, a small C value (0.001) makes the decision surface smoother and less sensitive to the individual data points, implying a higher regularization strength, whereas the use of the 'ovr' strategy indicates that the problem was decomposed into multiple binary classifications, which can be simpler and more efficient than other multi-class strategies like'ovo'. However, when trying to predict the test data, the model yielded very unfavorable results as its accuracy was a mere 12.8%. Support vectors were identified using the decision function of the best model, and visualized using t-SNE as circles and a lighter color palette in Figure 12a.

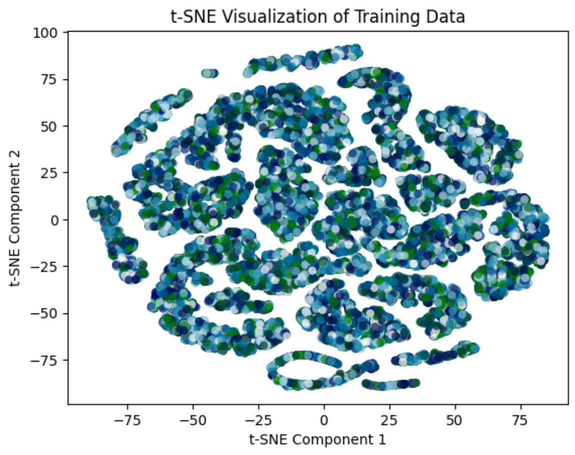
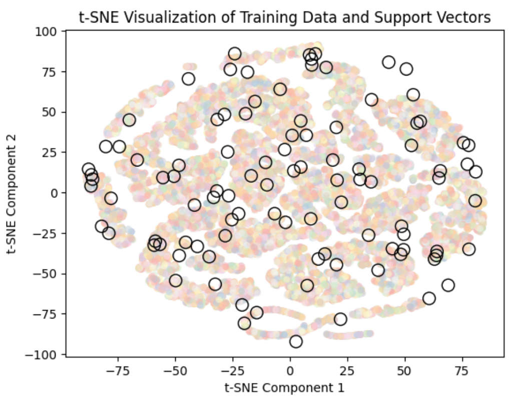


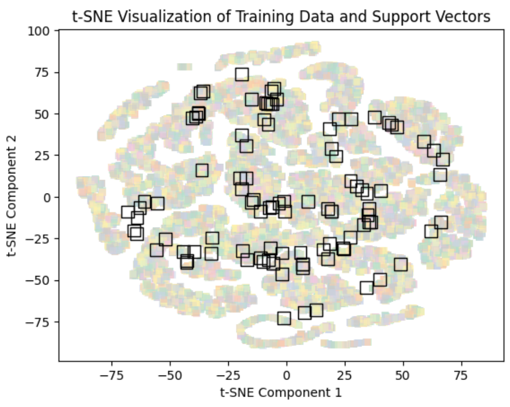
Figure 10: t-SNE Dimensionality Reduction on Training Data

### 8.2.2 Non-Linear SVM

For Non-Linear SVM,a grid search with 5-fold cross-validation was performed to find the best hyperparameters, where the parameter grid included these possible hyperparameter values: - **'C'**: [0.1, 1, 10, 100] - **'gamma'**: ['scale', 'auto', 0.1, 1, 10] - **'kernel'**: ['rbf', 'linear', 'poly'] - **'degree'**: [2, 3] (Only for the poly kernel) From the results, we found that the best model was the one with C:10, gamma: 10, and kernel: rbf. These findings suggest to us that the dataset is highly complex, as indicated by both a high C and a high gamma. This model fits the training data very closely and captures detailed patterns in the data, which are suggested to be non-linear by the choice of a rbf kernel. Although better than the Linear SVM, this model too yielded unfavorable results as its accuracy was 16% on the Test Set. Similarly, support vectors were identified using the decision function of the best model, and visualized using t-SNE as squares and a lighter color palette in Figure 12b.



(a) Linear SVM



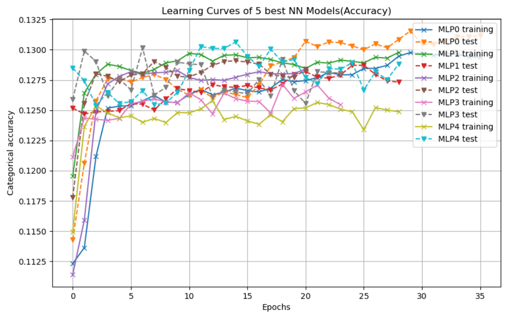
(b) Non Linear SVM

Figure 11: t-SNE Dimensionality Reduction on Training Data Non-Linear SVM

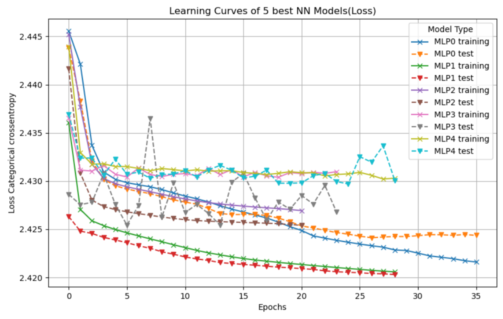
## 8.3 Neural Networks

For neural networks we decided to take a broad approach since the number of hyperparameters is big, we created a tuner where we ran 4 rounds of hyperparameter search, the ones accounted for in our experiment were: number of layers (fully connected), number of units, activation functions, optimizers, learning rates, momentum, and weight decay. The hyperparameters were used in three types of tuners, random search, Bayesian search, and hyperband search, we selected Bayesian and hyperband as our selected tuners since they take educated guesses in their trials, at the beginning we did the search with choices on each hyperparameter from a minimum to a maximum value, but in the final search we decided to give a small range in the hyperparameters and allow for float search in those smaller ranges.

In the end, we found that the models were overfitting the data since the model was converging in few epochs, so we decided to apply an additional search with less complex models; this is how we found the final selection of models, the final tuner search has the following hyperparameters in its grid. Number of layers: from 1 to 5, number of units: from 12 to 48, activation functions: sigmoid, tanhyperbolic, relu, possible optimizers: sgd or adam, learning rate: from 0.0001 to 0.1, momentum from 0.01 to 0.1, weight decay from 0.01 to 0.1. It's also worth noting that we used a fixed mini batch of 32 observations, as inputs we have 12 nodes, all of them from a range from 0 to 1, this is done in order to prevent a high bias in the weights, some of the 12 attributes are categorical one hot encoded feature, and others are normalized for a 0 to 1 scale. As output nodes we also have 12, this is because the attribute we want to predict is key, and this is a multi-categorical attribute with 12 features. So we one hot encoded the target and let the network decide on those 12 one hot encoded targets. As output activation function we implemented a softmax function, we were tempted on using also sigmoid, but we decided on softmax since it appears to be used more frequently in the literature for classification purposes. In terms of loss and accuracy we use categorical cross entropy loss and categorical accuracy. In the end we implemented early stopping and model check points to monitor the performance during the training phases, to conclude the set up of the model is worth noting that more searches were needed in the end and most of our models were too complex for the task at hand, this, in turn, generates models that have high variance and are intuitively not close to a good trade off in terms of bias and variance, things to keep in mind for the future would be to take a more parsimonious approach, even though we tried to aim for the later it was still not enough.



(a) Categorical accuracy



(b) Categorical crossentropy loss

Figure 12: Neural networks accuracy and loss for the best 5 models found

The 5 best models are shown in Figure 12, this is their performance in terms of loss and accuracy, all of them present very similar losses and accuracies and this is due to the fact that the models reached a limit with very few epochs.

The best five models accuracies are: Model 0 : 0.1320, Model 1 : 0.1270, Model 2 : 0.1275, Model 3 : 0.1280, Model 4 : 0.1290.

## 8.4 Ensemble Methods

In terms of ensemble methods, we implemented three main branches called Random forests, Bagging, and Boosting for predicting the labels of the attribute Key. We built ensemble classifiers, these three classifiers were used with different hyperparameters, it's worth noting that in general, we had major issues creating grid searches for ensemble methods (mainly bagging and boosting). This happened mainly because the classifiers are already too complex to add on top a high number of bagging estimators, or a high number of boosting machines. With this in mind, we remained mainly focused on what happens if we change the number of estimators in very limited steps. Now, Regarding random forests we experimented with: minimum samples to split, minimum samples in leaf, number of estimators, and maximum depth. For bagging we tried two different types of classifiers: Support vector machines and Random forests, we tested different numbers of bagging estimators, as 10, 20 and 30 estimators; we also tried different regularization values in SVM as 1(standard), 100, 1000. Trying different kernels generated major issues and so we could not test differences between linear, polynomial and radial basis function kernels, the final conclusion is that with a higher number of bagging estimators, the model tends to perform better in terms of F1-score, and accuracy.

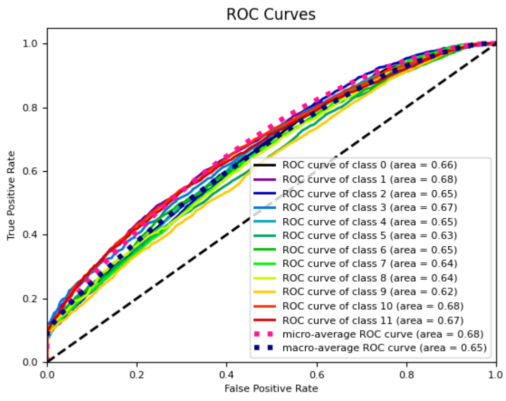


Figure 13: ROC curve for Adaboost ensemble method

Now regarding the boosting we implemented an Adaboost ensemble, with specifically random forests. The standard setting was used (trees of depth 1), we also tried to implement grid searches in this setting with hyperparameters: criterion as ‘entropy’ and ‘gini’, we also used the number of estimators for the random forests as 100, 200, 300, 400, 500. We used the number of estimators also for the boosting machines, using the same hyperparameters as in the random forests. In general terms the results were similar to bagging, giving better performance when the number of estimators was higher. Boosting results were the best, then random forests and then bagging. The ROC curve for the best adaboost ensemble is shown in Figure 13. It has 500 trees in the forest, and it has 500 boosting machines, the criterion is gini.

The best results for each ensemble branch are as follows. Random Forest Accuracy 0.2, F1-Score 0.19. Bagging(Random Forests) Accuracy 0.22, F1-Score 0.22. Boosting(Adaboost) Accuracy 0.22, F1-Score 0.22.

## 8.5 Gradient Boosting Machines

For the Gradient Boosting Classifier, we initially used this combinations of parameters: loss: ’log_loss’, learningRates: [0.01, 0.1, 0.2], n_esitmatorss: [25, 50], minSamplesSplitValues: [2, 4], minSampleLeafValues: [1, 3], maxDepths: [3, 2], tolerances: [1e-3, 1e-4]. We also decided to use the holdout method and to the various combinations, you have to add various percentages of the dataset ([0.1, 0.25, 0.5, 0.75, 1]). We did not notice meaningful differences in the results obtained with the various holdouts so we won’t report on them in the next paragraphs. We also implemented a 10-fold cross-validation and we’re reporting the mean values for the chosen metrics here. We initially obtained the best mean results on the validation dataset using the following parameters: loss: log_loss, learningRate: 0.1, n_estimator: 50, min_samples_split: 4, min_samples_leaf: 1 max_depth: 2 tolerance: 0.001 and the related results are accuracy: 0.1272, precision: 0.1297, recall: 0.1281, f1Score: 0.1222. The next best result, which had the same exact result metrics, only changed in tolerance using 0.0001 instead. We then decided to use values around the best previously obtained results which are:.

- loss = ”log_loss”
- learningRates = [0.2, 0.1, 0.05] (0.1 is better for bigger holdouts, adding 0.05 so maybe the results will improve since we were now using the full dataset)
- n_estimators = [50, 75] (we decided to add 75 since 50 was the most frequent in better results)
- min_samples_splits = [2, 3, 4] (3 out of 5 of the best results used 2, so we decided to add the middle value)
- min_samples_leafs = [3, 1, 2] (same reasoning as before)
- max_depths = [3, 5] (most of the best results used 3, so we thought that going higher might improve the results)
- tolerances = [0.01, 0.001] (all the best obtained results used 0.001, the initial parameters were 0.001 and 0.0001 so we thought that going higher might help)

In the results we obtained we noticed that the best learning rate was always 0.1, the best mean samples split also stabilized at 2 and the best tolerance was still 0.01 as we observed before. Instead using a higher number of estimators improved the results, also the mid value (2) for the min samples leaf helped with the improvement and the higher max depth improved the results too. We decided then to use an even higher number of estimators ([50, 75, 100, 125]) and an even higher number of max depth ([5, 10, 15]) while keeping the other parameters as they were. The best results we got were accuracy: 0.1827, precision: 0.20007, recall: 0.1673, f1Score: 0.1887, when using 125 as the number of estimators and 15 as the max depth. We also noticed that the next best results used just 75 estimators instead of 125 (and kept the number of max_depth as 15). Since the number of estimators really increases the training time we decided to fix the number of estimators to 75 and keep looking into the max_depth parameter; the values we tested are [15, 30, 50, 70, 100] so that we could compare the upgrade given by the various options. The new results we got were very close to the previous ones and actually, the best results were still the ones obtained with 15 as max_depth. The final parameters that we found were then: loss: log_loss, learningRate: 0.1, n_estimator: 75, min_samples_split: 2, min_samples_leaf: 2, max_depth: 15, tolerance: 0.01. When tested on the test dataset this classifier gets the following results: accuracy: 0.1807, precision: 0.1987, recall: 0.166, f1Score: 0.1872.

## 9 Advanced Regression

In this section, for the advanced regression we have decided to employ multiple regression techniques, specifically Gradient Boosting Regressor and Random Forest Regressor. The features we used were ['explicit', 'popularity', 'danceability', 'energy', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'BPM], with the target variable being the continous feature "artists_followers_mean" that we previously extracted from other features of the dataset. As per the dataset, it was divided into a split of 80% being Training Set and the remaining 20% being the Test Set. The section involves a detailed examination of each method individually, followed by a comparative analysis to assess their respective efficacy.

### 9.1 Random Forest Regressor

To optimize the hyperparameters of our RandomForestRegressor model for regression, we used a grid search technique, which employed cross-validation to assess model performance. The search was conducted over a defined parameter grid, which includes variations of 'max_depth', 'min_samples_split', and 'min_samples_leaf' values. The values we tried were the following:
- **max_depth**: Ranging from no maximum depth to depths incrementing from 2 to 20 with a step of 2. This varying depth allows the model to capture different levels of complexity in the data, as well as allowing us to explore the trade-off between model complexity and overfitting.
- **min_samples_split**: Values explored include 2, 5, 10. This allowed us to assess model robustness across different data granularities.
- **min_samples_leaf**: Values explored include 1, 2, 4, 8. This allowed us to control the size of the leaves and prevent the model from making over-specialized decisions.

After the grid search, the optimal model configuration was 'max_depth': **None**, 'min_samples_leaf': **2**, 'min_samples_split': **2**. The absence of a maximum depth suggests that our model benefits from complex decision boundaries. As for the minimum number of samples being 2, it suggests a preference for finer granularity in decision-making, potentially allowing the model to adapt closely to individual data points. Lastly, the minimum number of samples set to 2 indicates that the model is sensitive to even small variations in the data. The best tree from the Random Forest Model is depicted in Figure 14.

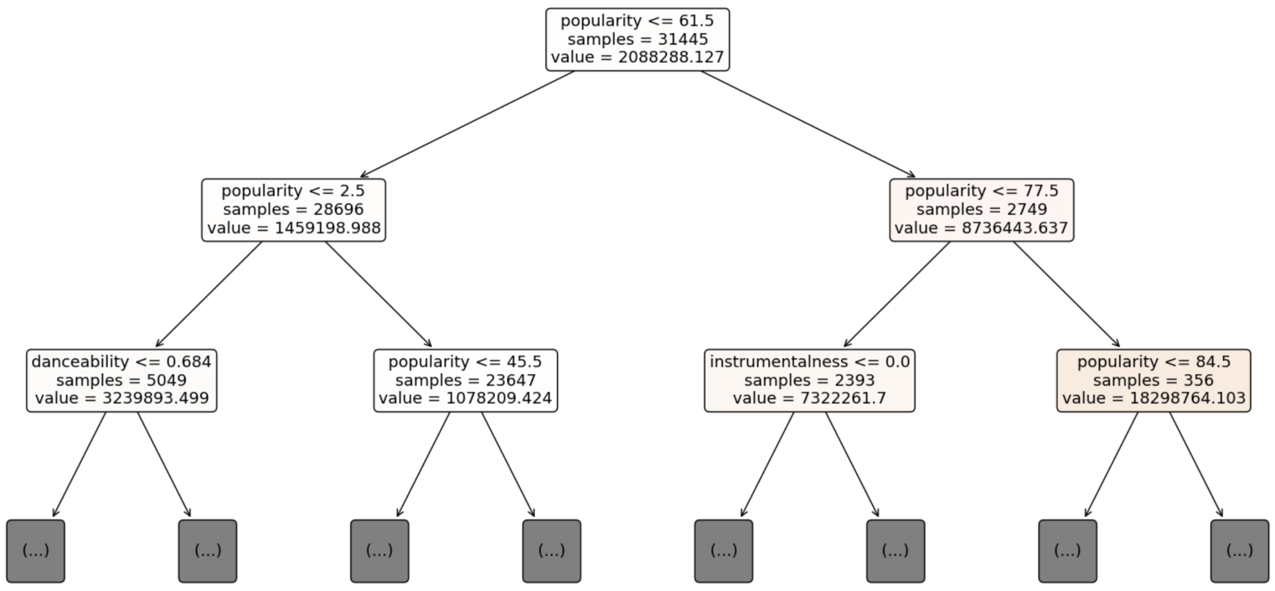


Figure 14: Random Forest Regressor Best Tree

## 9.2 Gradient Boosting Regressor

To optimize the hyperparameters of our Gradient Boosting Regressor model for regression, we used a grid search technique, which employed cross-validation to assess model performance. The search was conducted over a defined parameter grid, which includes variations of 'max_depth', 'min_samples_split', 'n_estimators', and 'learning_rate' values. The values we tried were the following:
- **learning_rate**: Values explored include 0.01, 0.1, and 0.2 as they reflect respectively low, moderate, and high rates. The trade-off lays between the speed and stability of model training.
- **n_estimators**: Values explored include 50, 75, and 100, which are chosen to manage the trade-off between performance and computational efficiency.
- **max_depth**: Ranging from no maximum depth to depths incrementing from 3 to 11. This varying depth allows the model to capture different levels of complexity in the data, as well as allowing us to explore the trade-off between model complexity and overfitting.
- **min_samples_split**: Values explored include 2, 4, and 6. This variety allowed us to determine the best balance between the purity of leaf nodes and the overall structure of the decision trees, helping to control overfitting by requiring more samples to split a node.

After the grid search, the optimal model configuration was 'max_depth': **11**, 'n_estimators': **100**, 'learning_rate': **0.1**, 'min_samples_split': **2**.A maximum depth of 11, which is a deep tree, indicates a relatively complex model. number of estimators being 100, it's a moderate value that allows for a strong ensemble while not being so large as to cause excessive computational overhead. The learning rate being 0.1 makes the learning process more gradual and stable, as well as it helps in reducing overfitting. The minimum number of samples split set to 2 indicates that the model is sensitive to even small variations in the data.

The best tree from the Gradient Boosting Model is depicted in Figure 15.

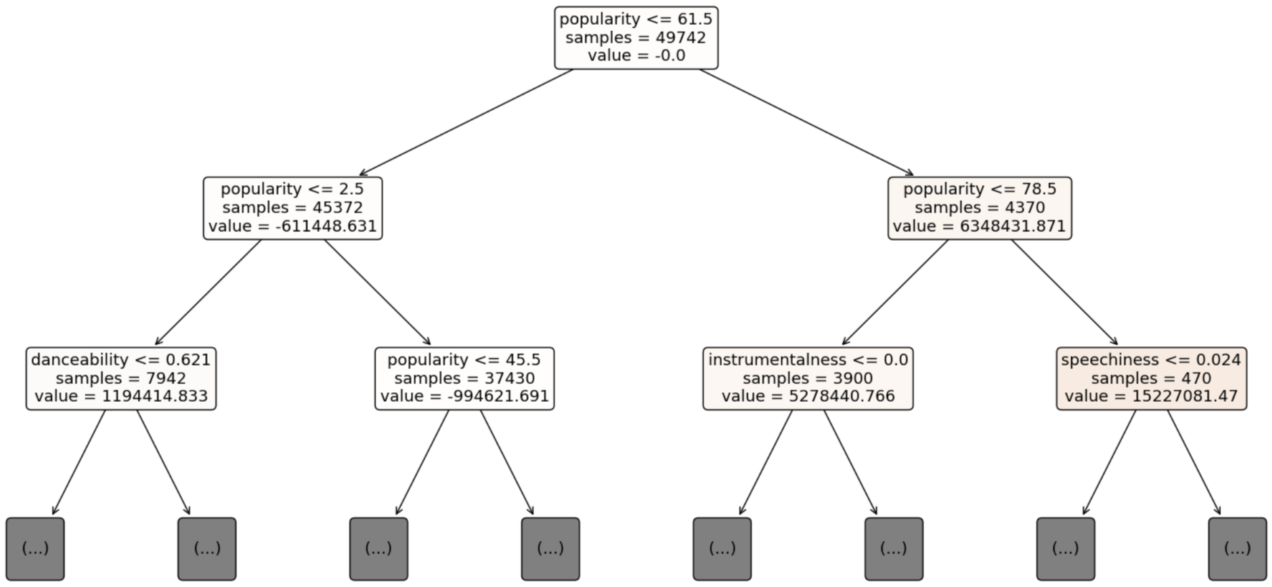


Figure 15: Gradient Boosting Regressor Best Tree

## 9.3 Comparison of Models

The table 9 compares the performance of our two advanced regression models, Random Forest and Gradient Boosting, on the Test Set using several metrics: $R^2$, MSE, RMSE, and MAE. As shown, the Random Forest model shows a better and higher $R^2$ value of 0.322 compared to 0.283 for Gradient Boosting, indicating marginally better explanatory power in regards to the variance of the target variable. Additionally, on all the other remaining metrics, the Random Forest Regressor shows better performance. It's important to note, that the results for the Random Forest model on the Test Set were more similar to those we got on the Validation Set, when compared to the performance of Gradient Boosting. This shows us that Random Forest provides a better fit and accuracy to our data, whilst not falling into overfitting. Nevertheless, the results captured by both models were not the most favorable, as they still had relatively low $R^2$ values, as well as quite strange high MSE error values which did not budge despite us trying out many different parameters through the grid search.

| **Model** | **$R^2$** | **MSE** | **RMSE** | **MAE** |
|---|---|---|---|---|
| Random Forest | 0.322 | 29205582280428 | 5404218 | 2132661 |
| Gradient Boosting | 0.283 | 30865975712682 | 5555715 | 2175858 |

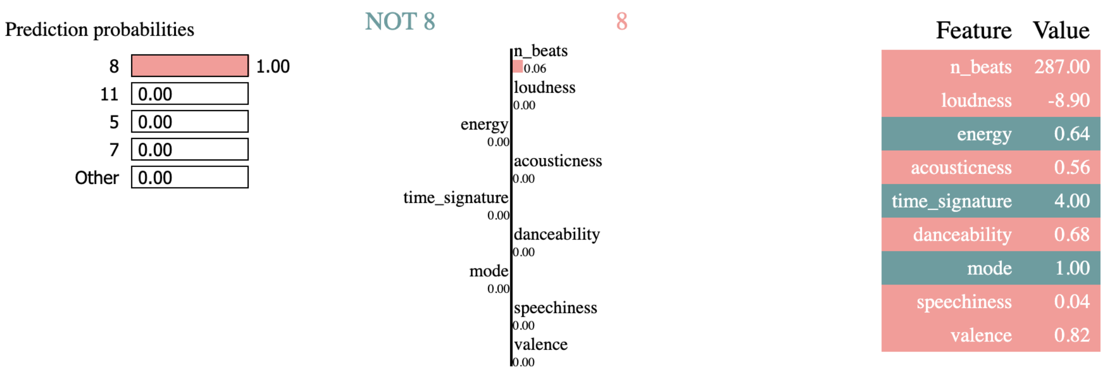
Table 9: Comparison of Advanced Regression models' performance metrics

## 10 Explainability

To shed light on how our Logistic Regression model makes its classification decisions, we've decided to delve into its inner workings using three different explanation methods: LIME, LORE, and SHAP. We have inspected one single randomly chosen instance from our test set, at index 112, which has been used by all the methods. This instance had a true class value of "8" and was correctly predicted by our model.

## 10.1 LIME

LIME was utilized to offer localized insights into the decision-making of the model. By concentrating on the single instance we choose, LIME perturbs the data in its vicinity, observes the model's reactions, and constructs a more simple model specific to that instance. The explanations of LIME can be seen in Figure 16. Among the features analyzed, "n_beats" emerged as the most influential factor in determining the model's prediction on the "key" feature, where a greater number of beats (287) can indicate a higher key value (in this case 8). Another interesting insight into how our model works was the fact when mode was 1, it presumed that the key would not be 8 and likely a lower key value.



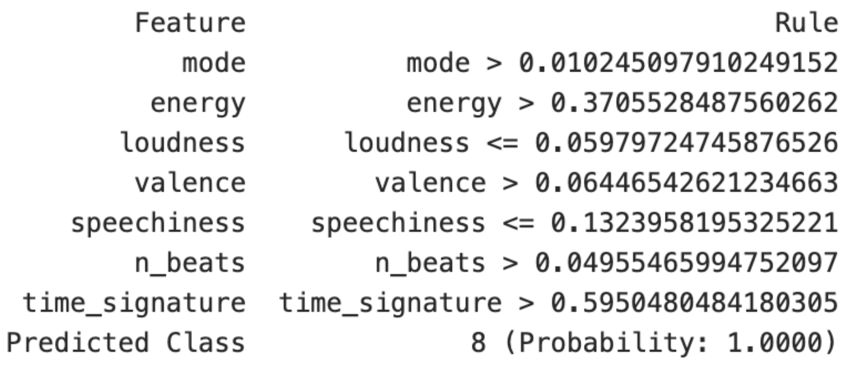
| Prediction probabilities | |
|---|---|
| 8 | 1.00 |
| 11 | 0.00 |
| 5 | 0.00 |
| 7 | 0.00 |
| Other | 0.00 |

| Feature | Value |
|---|---|
| n_beats | 287.00 |
| loudness | -8.90 |
| energy | 0.64 |
| acousticness | 0.56 |
| time_signature | 4.00 |
| danceability | 0.68 |
| mode | 1.00 |
| speechiness | 0.04 |
| valence | 0.82 |

Figure 16: LIME

## 10.2 LORE

We also utilized LORE to extract interpretable rules from our models' decision-making process. We applied the algorithm as follows: LORE extracts rules by analyzing the coefficients of the model's features. For instance, if a feature's coefficient is positive, LORE generates a rule stating that when that feature exceeds a certain threshold, it contributes to predicting a particular class. Conversely, if the coefficient is negative, the rule indicates that the feature's value falling below the threshold is associated with the predicted class. The threshold we set was at 0.6, to produce better results than the simple random model at 0.5. The rules generated for instance 112 are shown in Figure 17. For instance, the rule for the "mode" feature which shows that a higher value of mode than 0 is associated with key "8" is consistent with our findings from LIME. Moreover, these results suggest that high values of "time_signature" are associated with class "8".



```
        Feature                                 Rule
           mode            mode > 0.010245097910249152
         energy             energy > 0.3705528487560262
       loudness        loudness <= 0.05979724745876526
        valence          valence > 0.06446542621234663
    speechiness       speechiness <= 0.1323958195325221
        n_beats          n_beats > 0.04955465994752097
 time_signature   time_signature > 0.5950480484180305
Predicted Class                   8 (Probability: 1.0000)
```

Figure 17: LORE

## 10.3 SHAP

Lastly, we utilized SHAP to gain more insights into feature importance. Utilizing game theory principles, SHAP assigns each feature an importance score based on its contribution to the model's output for a specific prediction. From Figure 18, we can see that again "n_beats" is shown as an important feature for our model's decision making, which is consistent with LIME and LORE. Similarly, "loudness" is another important feature in the classification, which was indicated in LIME results. These findings are also further confirmed by the positive impact of these features, as shown in Figure 19. Nevertheless, a new insight from Figure 19is that "valence" feature is shown to have a negative impact on the predicted key value. This means that decreasing "valence" also decreases the predicted "key" value.

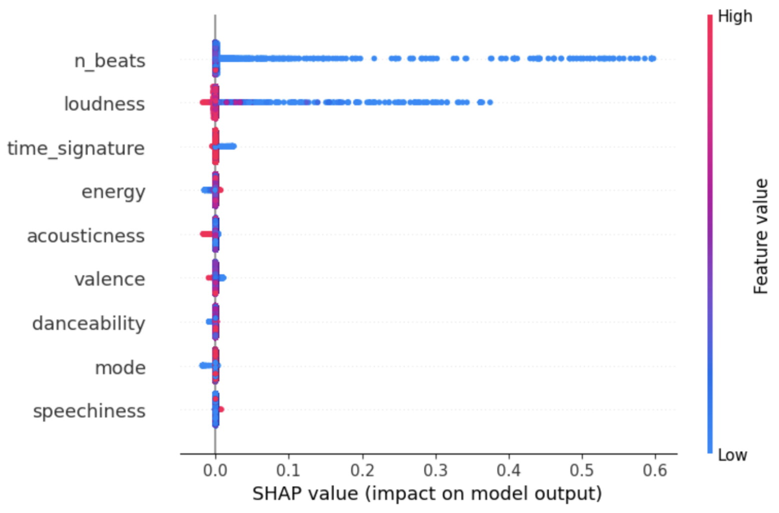


Figure 18: SHAP Summary

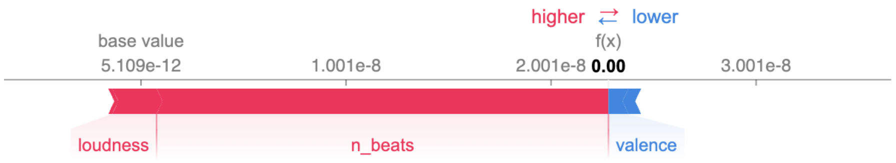


Figure 19: SHAP Plot