UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

DATA MINING 2 PROJECT

Autori:

Lorenzo Cascone (581465)

Emiliano Marrale (545552)

Salvatore Puccio (547518)

ANNO ACCADEMICO 2023/2024

Contents

1 Data understanding and preparation

We dispose of two datasets, one is tabular while the other one is a time-series. We are going to analyze them in separate sub-sections, performing both data understanding and data preparation separately in those sub-sections.

1.1 Tracks Dataset

1.1.1 Description

We dispose of two datasets, the first one contains tracks from Spotify extended (from the previous used during data mining 1) with additional features and is made up of over 100.000 records.

1.1.2 Assessing data quality

We began our analysis of the track dataset by examining for NaN values, finding none. Subsequently, we conducted a check for duplicated rows, initially focusing on the ID column. This examination revealed 33,674 rows featuring duplicated IDs. Additionally, we investigated for duplicated song names, uncovering 24,074 rows with duplicated names but distinct IDs.
Handling duplicated IDs: We observed that records with duplicated IDs feature different genres. This discrepancy could arise from songs being associated with multiple genres rather than being exclusively categorized under one genre. To address this, we will merge these records, consolidating the genres into a list format.
Handling duplicated song names: We also noticed that records with duplicated name may have different album name, disc number, popularity and track because for instance a song, can be published in a compilation such as relaxing rock music, and in another compilation called for example general rock music. Also the name of the song can be duplicated as two artists may release a song with the same name. So the dataset is filled with "duplicated" songs because of that, the id also changes between those songs, so for our analysis we are going to rely on the id, song name and artist name to uniquely identify songs. We then dropped all the duplicates songs with same name and same artist, keeping the record with highest popularity. We end up with a clean dataset composed of 81225 songs.
Dimensionality Reduction: We are going to drop the following features listed below:

- **track_number**, **disc_number**, **album_type**, **album_release_date_precision**, **album_total_tracks** as we don't see any use to it for our analysis.
- **n_bars** as it is highly correlated with **n_beats**.
- **feature_duration_ms** as it is highly correlated with **duration_ms**.

We also created a new feature named **overall_confidence** that aggregates all the confidence measures, calculated by summing them all and then diving by their number.

Conclusions: At the end of our analysis we can affirm that the best way to uniquely identify a song is by using as key the features id, name, and artist.

1.1.3 Distribution Analysis

After examining the **genre** distribution in our dataset, we found a total of 114 unique genres among the tracks. Most genres are represented approximately 1000 times. However, some genres deviate from this trend: "British" appears 576 times, "Disco" 625 times, "Death Metal" 660 times, "Dancehall" 700 times, "Dub" 830 times, "Cantopop" 877 times, and "Disney" 894 times.
Among the tracks we have the presence of 31479 unique artist. Below we can see the top 10 occurrences of artists 1:

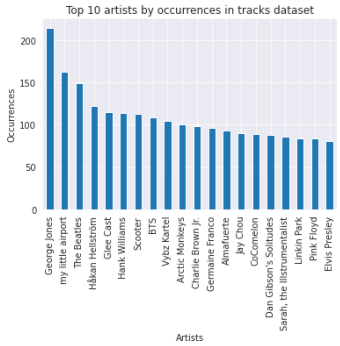


Figure 1: Top 10 artists

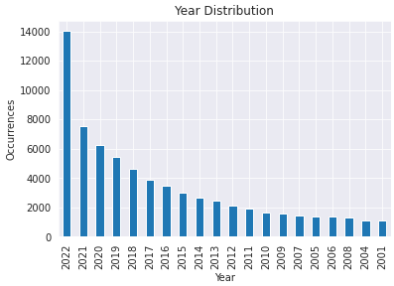


Figure 2: Year Distribution

The songs in the dataset are published in 100 unique years in our dataset. Here's 2 the distribution of the top 20 years. Notably, the majority of our songs are recent, with 70,459 out of 81,225 tracks published after 2000. Looking at the features **duration_ms** and **feature_duration_ms** we can say that they are both very similar, the second one may be a variant of the first. Without more context it's hard to say exactly why both exist, but their practical differences seem negligible for most purposes. The duration_ms feature likely represents the duration of tracks in milliseconds, in our dataset the median is 3 minutes and 35 seconds per song, with the shortest track of around 8.6 seconds long and the longest of around 1 hour and 8 minutes. The **explicit** feature is highly imbalanced with 91% of the tracks being non-explicit. In our dataset we are dealing with songs with a **popularity** between 0 and 95, with a median of 33. We have 7249 tracks with a popularity of 0. Features like **danceability** and **energy** exhibited distributions skewed towards higher values, underscoring a collection leaning towards more danceable and energetic tracks, we also noticed, looking at the **loudness** distribution a concentration of songs at "high volume" as explained by spotify in this article [7]. The **acousticness** and **instrumentalness** values displayed a wide-range of tracks with minimal to high levels of voice and instrumental content, showcasing the diversity in musical styles and approches within the dateset, with the great majority of the songs being probably not acoustic and not instrumental. The **speechiness** feature shows a right skew, with an average of 0.089, highlighting a musical predominance (typical songs) and not talk show, audio books or podcasts. The majority of the songs are studio recordings as highlighted by the **liveness** features. Through the analysis of **valence**, with an average of 0.463, we can say that the songs in our dataset represents a balanced emotional spectrum, from uplifting to darker tones.

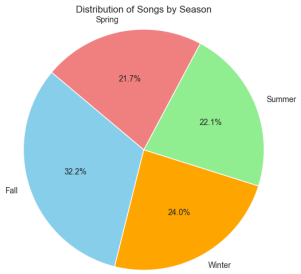


Figure 3: Distribution of songs publishing

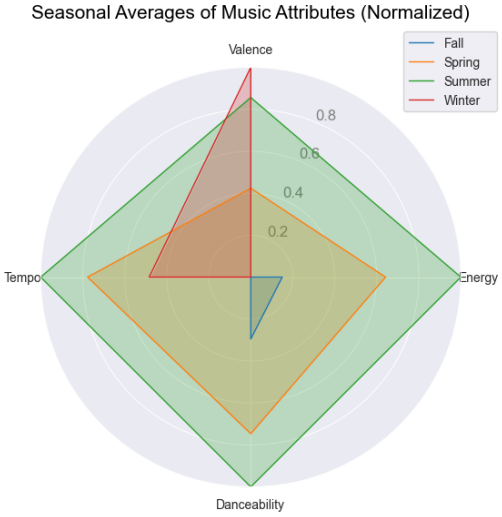


Figure 4: Changes in values between seasons

It's clear, by looking at image 3 that there is a tendency of publishing songs in Fall, this may be

due to the dates of the major music awards (i.e. MTV Awards, Billboard Music Awards, etc) [5]. We also digged more in detail within the data and found out that songs released in summer tend to have higher values in features like energy, danceability and tempo. The **mode** feature in the dataset, telling us whether a song is in major or minor mode, is slightly unbalanced towards the positive class, with 63% of the songs being in major mode. We also notice, looking at the **tempo** variable, that some songs have 0 tempo, for tracks like "Clean White Noise", "Ocean waves sounds", etc, it can be possible. For real songs, it's not, so these are outliers or malformed data for sure, and we will handle them in the preparation phase. Between the not malformed tempo values we can say that statistics suggest a broad diversity of musical songs within the dataset, reflecting a wide array of genres and styles. The range from minimum to maximum tempo indicates an inclusive selection of music, from the very slow and measured to the extremely fast-paced. The vast majority of tracks are in 4/4 time, which is expected as it's the most common time signature in popular and contemporary music. Other **time_signatures** are significantly less common, with 3/4 being the second most frequent, followed by 5/4, 1/4, and lastly, a very small percentage of tracks with a time signature recorded as 0, which may indicate missing or special cases. There are probably outliers present in both **n_beats** and **n_bars** distributions, as indicated by the long tails extending to the far right. These outliers could represent exceptionally long tracks or tracks with a very dense rhythmic structure. Their distribution is right-skewed.

1.1.4 Correlation Analysis

The correlation analysis of the dataset reveals several interesting relationships between the features as shown in the heatmap 5.

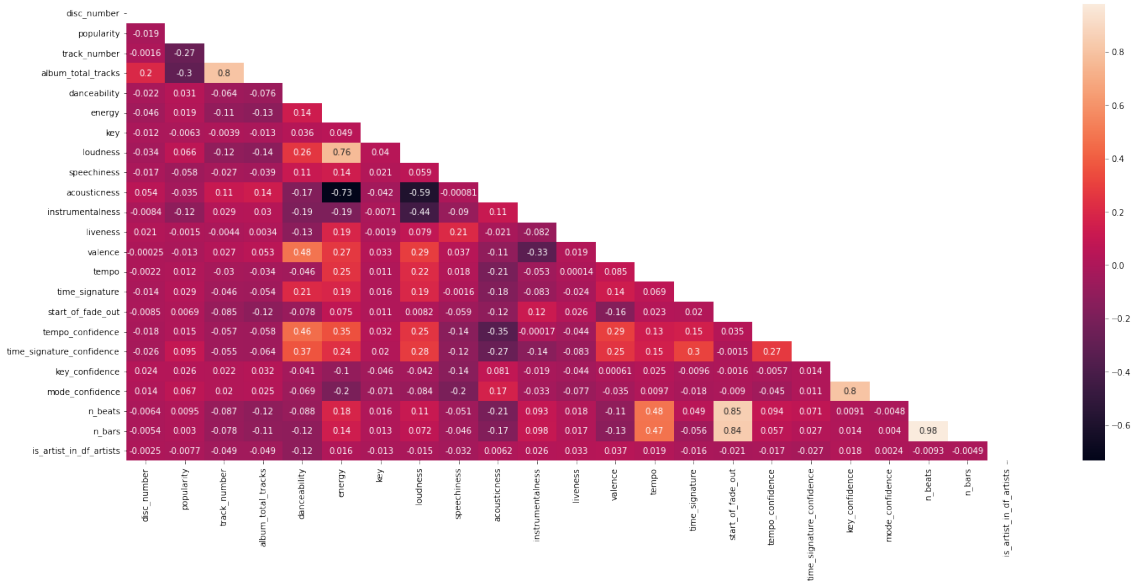


Figure 5: Tracks dataset heatmap

- **Duration and Beats/Bars**: There is a strong positive correlation between `duration_ms` and both **n_beats** (0.86) and n_bars (0.85), indicating that longer tracks tend to have more beats and bars, as expected.
- **Energy and Loudness**: A strong positive correlation exists between **energy** and **loudness** (0.76), suggesting that tracks with higher energy levels tend to be louder.
- **Danceability and Valence**: There's a moderate positive correlation between **danceability** and **valence** (0.49), indicating that tracks that are more danceable tend to have a more positive mood.

- **Acousticness and Energy**: There is a strong negative correlation between **acousticness** and **energy** (-0.73), indicating that tracks with higher acousticness tend to have lower energy levels.
- **Beats and Bars**: A very strong positive correlation between **n_beats** and **n_bars** (0.98), which is expected as both are measures of the track's rhythmic structure and duration.

1.2 Artists Dataset

1.2.1 Description

The second dataset is related to artists, contains over 30.000 records and the following features:

- **id:** The Spotify ID for the artist.
- **name:** The name of the artist.
- **popularity:** The popularity of the artist. The value will be between 0 and 100, with 100 being the most popular. The artist's popularity is calculated from the popularity of all the artist's tracks.
- **followers:** Information about the followers of the artist.
- **genres:** A list of the genres the artist is associated with. If not yet classified, the array is empty.

1.2.2 Assessing data quality

We began by examining the dataset for duplicated rows and NaN values. Upon analysis, it was revealed that the following columns contains NaN values: id (1 occurrence), name (2 occurrences), popularity (1 occurrence), followers (1 occurrence), and genres (1 occurrence). For these NaN values, we will exclusively remove records where the name is NaN, as such entries wouldn't be reliable for associating them with our tracks. Additionally, within the artists dataset, there are 435 records with duplicate names, and 4 records has duplicated id.
Handling duplicated IDs: We removed the records with duplicated id, keeping only one of them as they shown the same values for all features.
Handling duplicated names: After analyzing the dataset and acquiring additional information, we opted to remove duplicated artists who shared identical genres, keeping the one with highest popularity. This action suggests that such entries represented the same artist. Conversely, artists with identical names but different genres might still represent distinct individuals.

Further investigation was conducted to determine the number of relations established by linking the two datasets using the artist name. This analysis revealed that 81,479 records in the tracks dataset contain a matching artist name from the artists dataset.

1.2.3 Distribution Analysis

Looking at the distibution of the **popularity column** we can see that we are dealing with an almost symmetrical distribution, mainly with artists that have a popularity score between 33 and 44, followed by artists with a popularity score between 22 and 33, and with only 101 artists with a popularity greater than 80. Also we have many artists (1337) with a popularity ranging between 0 and 5. We also checked for outliers through the use of boxplots, the only values falling outside the 90th percentile were in reality real values of very famous artists. We did the same study for the **followers column**, we noticed that the majority of artists has a small number of followers, the veriable has a mean of approximately 416866 and a median of 15814 which suggests a right-skewed distribution which is also confirmed by the histrogram. We have 110 artists with 0 followers. Even for this variable the possibile outliers pointed by the boxplot actually indicates real values of famous artists.

The genres column is distribute as shown in the image 6, we can see a majority of artists that

produce sleep songs and as the second most frequent the genre 'filmi' namely music soundtracks that are produced for India's mainstream motion picture industry.

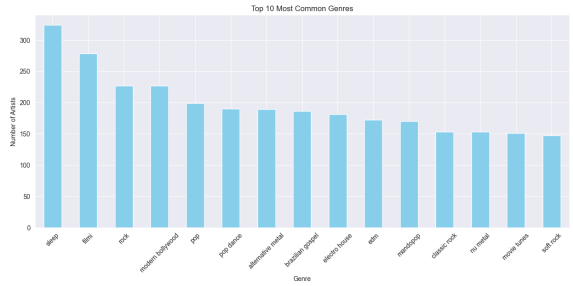


Figure 6: Distribution of genres in artists dataset

1.2.4 Correlation Analysis

After the normalization of the variables followers and popularity we plotted this scatter plot 7 that shows a heavy concentration of points around the lower popularity scores, indicating a high number of entities with low popularity and followers. There seems to be a trend where entities with higher popularity scores also tend to have more followers, as expected; but not a correlation trend, in fact the the resulting heatmap show a correlation value of 0.32.

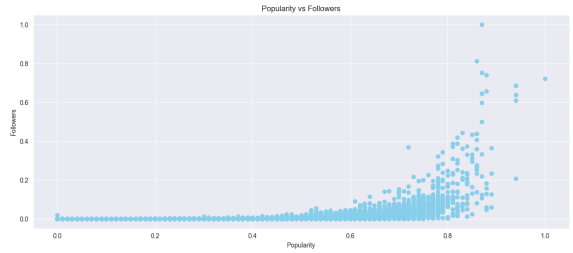


Figure 7: Followers vs Popularity

1.3 Time-Series Dataset

1.3.1 Description

The dataset is made of spectral centroids. These are a measure used in digital signal processing, specifically in the context of audio, to characterize a sound's brightness or dullness. A lower spectral centroid indicates a duller sound (more low-frequency energy) while a higher spectral centroid indicates a brighter sound (more high-frequency energy). The dataset consists of 10,000 time series, with all but two having a length of 1280 entries.

1.3.2 Assessing data quality

The dataset doesn't contains missing values. Each track has a unique identifier, this could be useful for cross referencing with the other datasets. We noticed that there are 271 duplicated track_ids, but the genre is different, so we decided to keep them in order to perform analysis over the genres column.

1.3.3 Distribution Analysis

We have a total of 20 genres, each consisting of 500 time series representing individual songs. We noticed that the dataset have a wide range of values, this indicate the diverse musical characteristics across the tracks and the genres. The mean and standard deviation vary significantly across the various genres as highlighted in image 8.

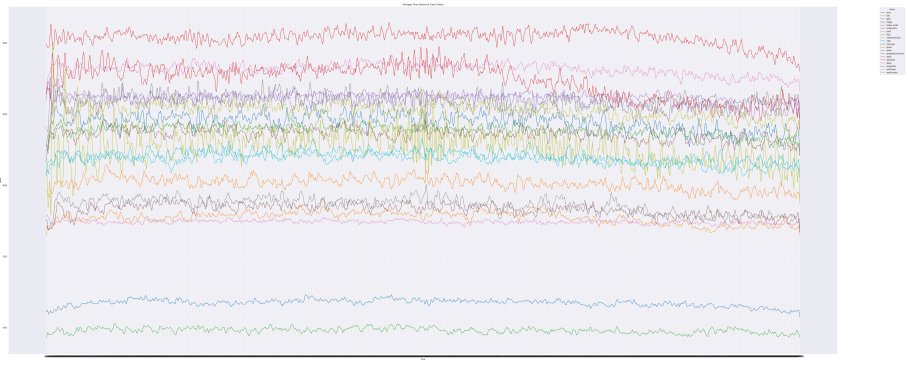


Figure 8: Mean values of spectral centroids for songs genres

We can see that genres like 'happy', 'j-idol' and 'heavy-metal' shows higher mean values of spectral centroids; on the other hand genres like 'new-age', 'piano' or 'sleep' show lower mean values indicating softer sound characteristics. The plot also suggests that there are no discernible trends in the different genres. Each series appears to fluctuate within a particular range that is characteristic of its genre. There might be an exception for the two series in red, which appears to have a trend leading downwards towards the end of the series.

1.3.4 Pre-processing

As depicted in the preceding image (8), various genres occupy distinct ranges. Consequently, we must conduct offset translation and amplitude scaling to align them to a comparable starting point. We performed an algorithm for trend detection, which employs a moving average that stands as a derived series that is then used to assess the presence of a trend. By evaluating the slope and the rate of change of the moving average, the code quantifies the direction and strength of any potential trend. Thanks to this algorithm we got the index of the time series that needed trend removal and performed it.

2 Timeseries Analysis

2.1 Motifs/Discords analysis

2.1.1 Motifs

Before performing the analysis we performed noise removal with a `window = 10` in order to make it easier to identify true motifs and discords. The Motifs analysis was performed with the matrixprofile library in the entire dataset with different `window size` and `n_motifs`. After some comparisons we fixed the parameters at `window size = 50` and `n_motifs` = 3. The motifs discovered in the time series analysis reveal significant patterns which are recurrent across different genres of music. Our analysis found 101 motifs repeated more than 2 times. The distribution of those motifs across different music genres is unbalanced, genres like new-age, minimal-techno or progressive-house have up to 10 times the motifs of genres like sleep, happy or mpb, this is quite expected because those kind of genres tend to have more repeated patterns. Among the different genres we noticed many differences for example the motifs of *minimal-techno* 9 are representative of the genre's emphasis on minimalistic yet distinctive rhythmic patterns.

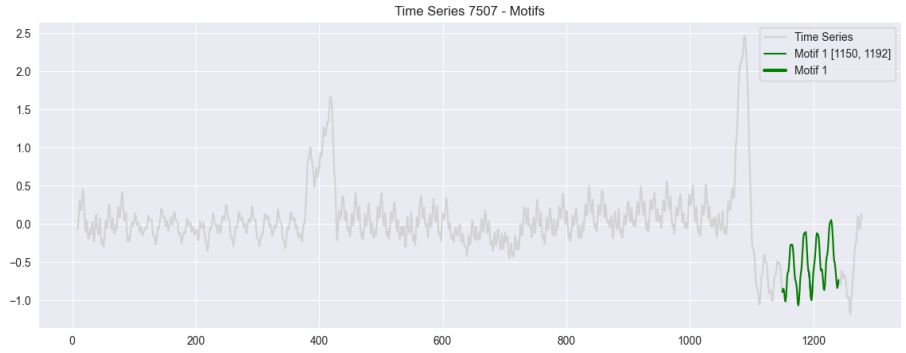


Figure 9: Motif for a minimal-techno song

This motif is shared not only by some *minimal-techno* songs, but also by a Honky-Tonk song, this echoes the up-tempo and piano rhythm. Among the various genres we searched where the motifs tend to appear within the songs as shown in image 10

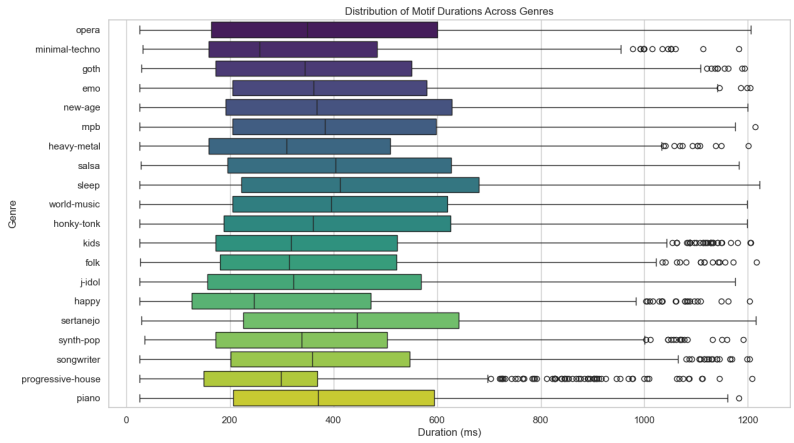


Figure 10: Motifs appearance in the song

The boxplot displays the distribution of motif appearance across different musical genres. We found out that genres like 'minimal-techno' and 'opera', show wider range of motif appearance, indicating variability in the lenght of motifs within these genres; several genres also shows outliers, suggesting presence of exceptionally motifs compared to the typical range within the genre; the

median show point varies across the genres, suggesting that some genres might typically feature repeated pattern at a certain point towards the start of the song. We also calculated the similarities between the various motifs using DTW in all the motifs found with an extensive search, the distribution of similarity scores is the following 11:

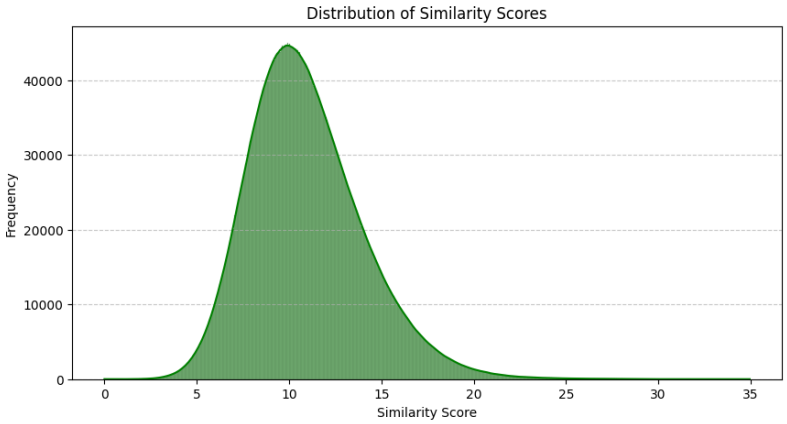


Figure 11: Distribution of similarity scores

We found out that the average similarity scores between different genres show varying levels of relationships: some genres such as "goth", "opera" and "new-age" tend to have higher average similarities with various other genres, possibily indicating more shared characteristics in their spectral centroids across these genres. While genres lile "Heavy-Metal", "Piano" and "Goth" shows the best similarity score within themselves, suggesting that these genres tends to have more consistent and repeated motifs compared to other genres in the dataset. This could reflect a homogeneity in the spectral features or structure of the music within this genre.

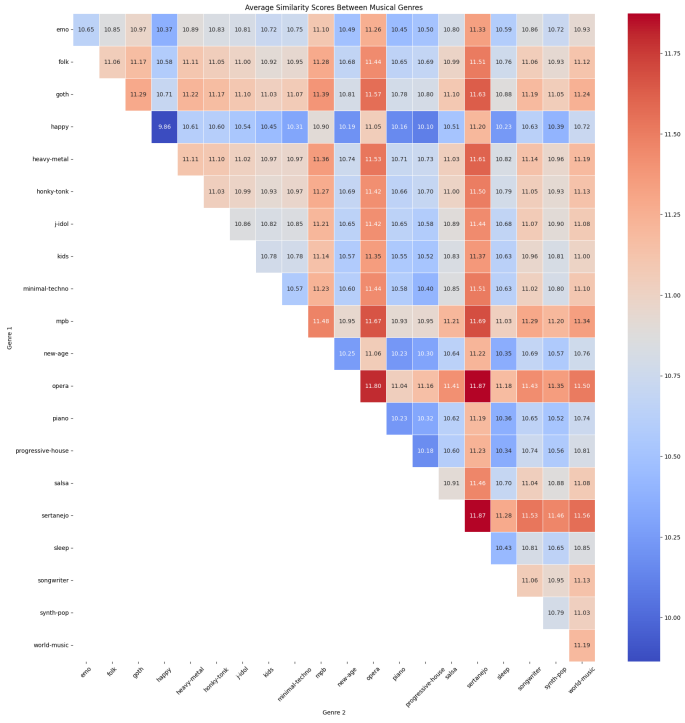


Figure 12: Average similarity scores between musical genres

On the other hand genres like "Honky-Tonk" shows distinctly lower similarity scores, which

could imply unique musical characteristics that set it apart from other genres.

2.1.2 Discords

For discord extraction we fixed a `window = 50` as for the motifs and extracted 10000 of them. Then for each discord segment, we calculated: Standard Deviation, Mean, Median, Skewness and Kurtosis, and we did the same for the timeseries in which the segment belong. In this way we created a dataset that would help us understand which discord were the most discording from the entire timeseries. Then we restricted the search field by testing with a threshold and trying to maximize values of the discord segments like Mean Z-Score, Standard Deviation of Z-Score, Skewness and Kurtosis. We ended up with a subset of 174 discords, distributed in this way among the classes 13:

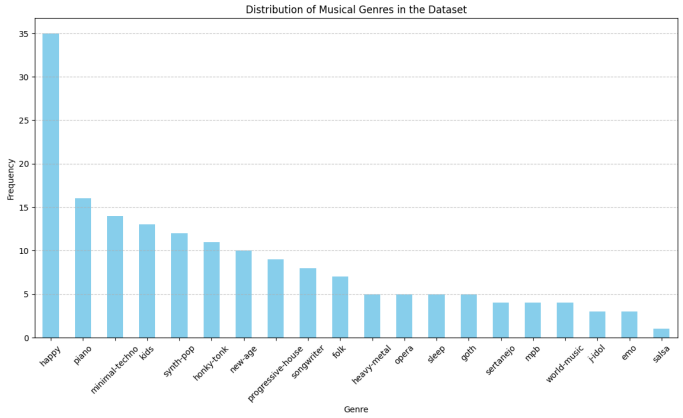


Figure 13: Discord distribution among the genres in maximum variation segments of discords

Among these subset we found some interesting discords/anomalies like 14 and 15, these discords might correspond to dramatic shifts in the music's properties, such as transitions from a verse to a chorus, unexpected breaks, or instrumental solos that differ significantly from the song's main patterns. In particular 15 with the sharp spike could indicate a sudden change in pitch or timbre or even a strong instrument like a drum, while 14 with a deep trough could indicate a drop or absence of sound.

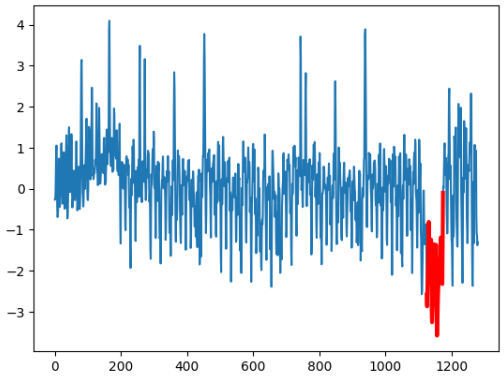


Figure 14: Discord of song Masterplan - Live Version from Dyewitness

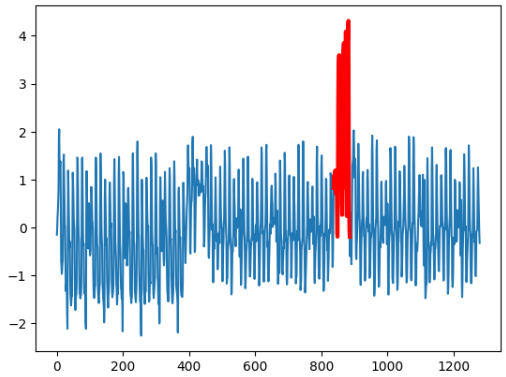


Figure 15: Discord of song Devistated - Edit from Ferocious

2.2 Clustering

2.2.1 TimeSeriesKMeans - Euclidean

We applied offset translation and amplitude scaling to the entire dataset, as outlined in Section 1.3.4. Additionally, Piecewise Aggregation Approximation (PAA) with **frame_size=5** was utilized to reduce computational time and improve clustering performance. Following PAA, we normalized the dataset using MinMax normalization. Subsequently, we conducted the elbow method to determine the optimal number of clusters. Using **euclidean distance** as measure, we explored cluster counts ranging from k=3 to k=19, incrementing by two. We also experimented with a wider range from k=3 to k=200, increasing by 20. While the sum of squared errors (SSE) continued to decrease as anticipated, the silhouette score dropped below 0. Consequently, we constrained the range to 3-19. The outcome of the elbow method is depicted below 16.

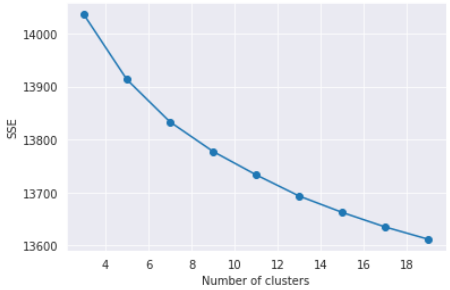


Figure 16: Elbow Method

Ultimately, we opted for k=9, resulting in an SSE of 13777 and a silhouette score of 0.012, indicating relatively poor performance. Interestingly, with higher values of k, the silhouette score progressively declined, falling below 0. Conversely, lower values of k (e.g., 3, 5, 7) yielded inferior SSE but improved silhouette scores. Subsequently, we visualized the clusters using both PCA and UMAP [6], showcased in the accompanying images 17 18.

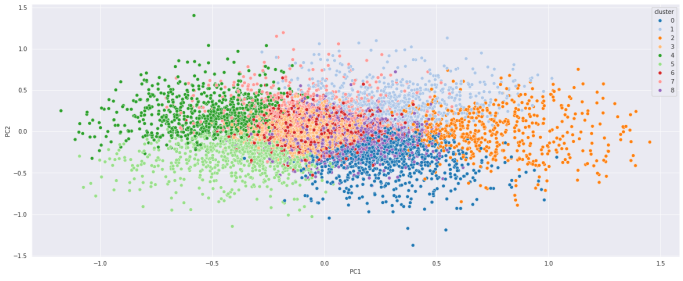


Figure 17: PCA

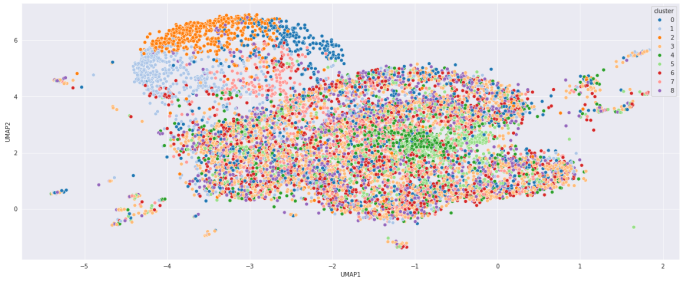


Figure 18: UMAP

PCA reveals a visually appealing clustering structure, whereas UMAP presents a contrasting narrative. The bad quality of the clustering structure becomes evident when examining the distribution depicted in the image 19. Notably, no clearly defined clusters emerge among the genres represented. However, cluster 3 exhibits a noteworthy concentration of genres including emo, kids, minimal-techno, mpb, salsa, and sertanejo. Conversely, cluster 2 displays a deficiency in incorporating salsa and sertanejo, with occurrences totaling fewer than 10. On the whole, the clustering approach employed fails to contribute additional value to our analysis.

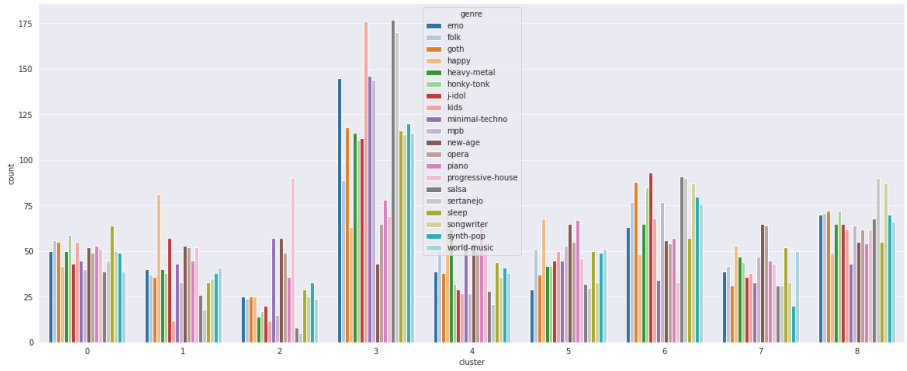


Figure 19: Genres count in clusters

2.2.2 Hierarchical Clustering

We also performed hierarchical clustering with linkage methods: single, complete, average, ward. This last one was the only one that gave reasonable results on the dendrogram. Ward method was the one chosen because other methods gave non-optimal dendrograms.

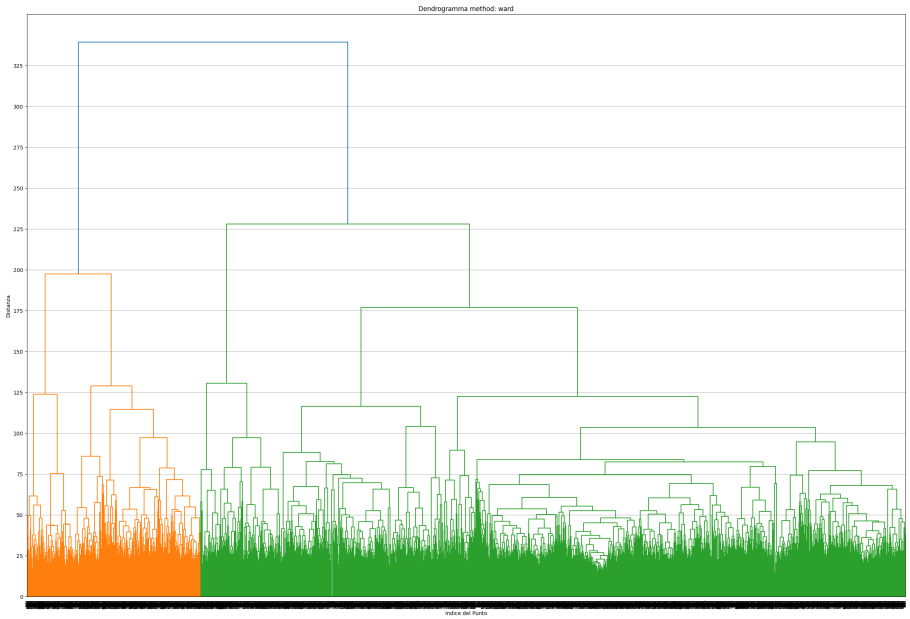


Figure 20: Dendrogram with ward method

After a few attempts we cut the dendrogram at 180 to have 4 clusters. Cutting higher with less clusters of cutting lower with more, gave us not quite interesting results. After that we plotted the results after computing the cluster labels

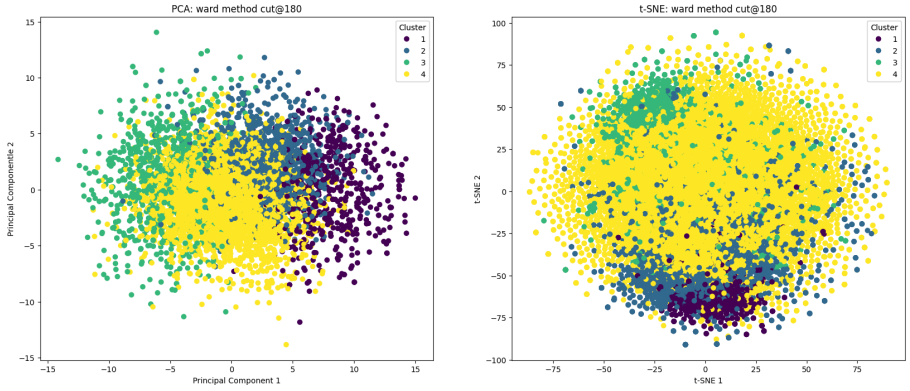


Figure 21: PCA, t-SNE plot

PCA shows relatively clear separation between the clusters. Except for cluster 1 and 2, they overlap. t-SNE shows a very large cluster 4 and small other clusters. So time series in cluster 4 have very high local distances. In the other hand cluster 1,2,3 are composed of time series more near to each other, in terms of distance.

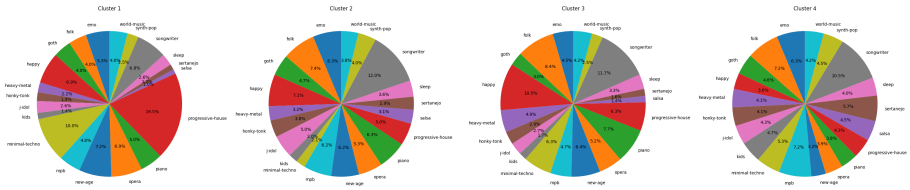


Figure 22: Hierarchical clustering Pie chart

In this pie chart we can see the genres (taken from the file name of the time series) distribution among clusters. Clearly in cluster 1 'progressive-house' genre is the most frequent, followed by 'minimal-techno'. We noticed that cluster 4, the yellow one in t-SNE plot, was the most wide, and in genres distribution is also the most balanced. Cluster 1,2,4 were more dense in t-SNE plot and here we can see that they all have in common one thing: they have a dominant genre. Even tho cluster 4 have 'songwriter' genre which is dominant, the dominance is really small. Cluster 1 for example have two dominant genres which take 30 percent of the cluster, and in fact it was the most dense in t-SNE.

2.2.3 Other considerations about clustering

We also performed feature-based clustering by calculating *Mean, Standard Deviation, Skewness, Kurtosis, and Frequency-Domain Features.* With five clusters, we had a silhouette score of 0.14. However, the PCA and t-SNE visualizations were encouraging, showing well-separated clusters. Despite this, the distributions of attributes were disappointing.

We also joined the *tracks dataset*, using the same IDs as in the time series, to examine the distribution of other song attributes. We initially focused on the categorical attributes. Features like *explicit* and *key* were almost perfectly balanced among clusters. Numerical features like *danceability*, *energy*, and *popularity* were divided into bins, and their distributions among clusters were plotted. However, no significant patterns emerged.

For instance, with *danceability*, we created bins like *0-0.2*, *0.2-0.4*, ..., *0.8-1*, and noticed that three clusters were dominated by the *0.4-0.6* bin, while one cluster was dominated by the *0.6-0.8* bin. For *popularity*, we divided the attribute into bins again, and the distribution was almost perfectly balanced. When analyzing the *energy* attribute divided into bins, we observed a pattern similar to that of *danceability*.

2.3 Classification

2.3.1 KNN using DTW and Euclidean

In this section, we have applied KNN in two kinds of classification tasks, binary classification and multi class classification. In order to choose the K we have performed the KNN many times with a smaller sample of the data. We ended up fixing $K = 9$ for DTW and $K = 6$ for euclidean. Keeping in mind the computational cost of DTW we decided to apply the algorithm with the timeseries approximated through PAA. The KNN training was performed in a dataset of 6000 timeseries with equally represented classes and with each timeseries composed of 320 timepoints. For the binary classification problem we choosed, two among the most dissimilar classes found with motifs analysis "heavy-metal" and "piano".

Looking at table 1 we can see that the KNN with DTW outperforms the one using Euclidean distance, especially in the binary problem, dtw shows superior precision for both classes, meaning that when knn does a prediction while being trained with dtw is almost always right (97% of the times); also the recall, especially for class 4 is much better with DTW, achieving 0.58 of recall vs the 0.18 of euclidean. In general we can say that class 4 "heavy-metal" is better predicted by both methods in terms of recall, but dtw is more precise. Class 12 "piano" is where the dtw demonstrates a clear advantage, with significantly better precision and recall.

	Accuracy	Precision	Recall	F1-score
KNN Binary Class - Euclidean	0.58	0.72	0.58	0.50
KNN Binary Class - DTW	0.78	0.83	0.78	0.77
KNN Multi Class - Euclidean	0.11	0.17	0.11	0.10
KNN Multi Class - DTW	0.19	0.17	0.17	0.17

Table 1: Comparison of KNN Classifications

When comparing the performance of KNN with DTW multiclass and KNN with Euclidean multiclass we can see that the DTW method perfoms better again, we can notice that there are classes (mpb, songwriter, world-music) where both methods struggle, indicated by low scores across all metrics, while for some classes like "minimal-techno" and "happy" the perfomance are much better. The small advantages given by DTW in the multiclass problem come at the cost of a very big computation time of about 36 hours.

2.3.2 Shapelet-Based Classification

We applied the shapelet classification in the dataset after performing MinMax scaler and PAA with 320 time-steps, cause of the computation cost. We used the `tslearn shapelets` classifier using the optimizer with 0.001 optimization steps and calculated shapelets of length 16; we also tried with the sktime RandomShapeletTransform using the DecisionTreeClassifier over the distances but worse results compared to the first one. Before fixing the parameters we did many tries with various parameters and ended up deciding these were the best for our dataset. The results were disappointing with an accuracy of only 26%, the classifier struggles with some£ genres, showing particularly low scores for certain classes like goth and opera, while performing moderately well on other classes like happy, honky-tonk, progressive-house and seranejo. The model has a perfect precision score for class 11 (opera) but a very low recall, indicating that while it's very confident when it predicts opera, it rarely make that prediction. We made a comparison of the extracted shapelets with the motifs, calculating with the euclidean distance the similarity. In order to do it, given that one dataset was approximated with PAA while the other were not, we applied the PAA transformation to the normal fragment (representing the motif) before comparing it with the already approximated one. We found out that 132 motifs and shapelets are very similar, here is an example 25 in this image we can see that the shapelet almost perfectly matches with a motifs present in the timeseries:

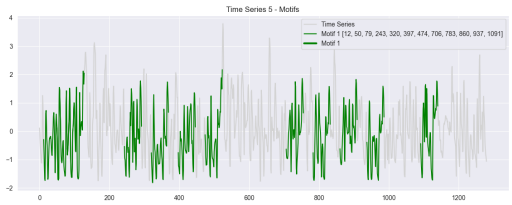


Figure 23: Motif

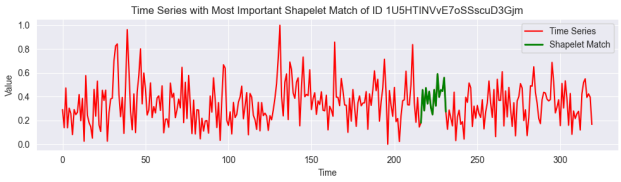


Figure 24: Shapelet

Figure 25: Comparsion between a Motif and a Shapelet

For the shaplet classification problem we also did a binary classification task: we tried to train the model in order to recognize only "heavy_metal" and "piano" songs, with the same hyper-parameters used for the multi-class problem. The results were the following: The performance

Class	Precision	Recall	F1-score	Support
Heavy-metal	0.91	0.86	0.89	50
Piano	0.87	0.92	0.89	50

Table 2: Classification Metrics for binary classification task

are way better then the multi-class problem, as we can see through the table 2 and the confusion matrix 26 the classifier achieved an overall accuracy of 89%, with balanced performance across both classes. For 'heavy-metal' the precision of 91% suggests that the classifier is highly effective at identifying true heavy-metal samples, this were also evident in the multi-class problem where that class was one of the best predicted ones.

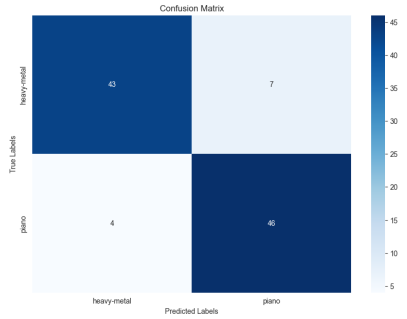


Figure 26: Confusion Matrix

Conversely, for 'piano', the precision is slightly lower at 87% but much better than the precision of 42% of the multi-class problem. Overall the classifier demonstrates robust performance for the binary problem but struggle (also in terms of efficiency) in the multi-class one.

2.4 Sequential Pattern Mining

For sequential pattern mining we prepared our data using SAX approximation, with $word-size = 128$ and $alphabet-size = 20$, before fixing on these values we had many tries and choose these minding computation time of prefix-span and results. As wrote before we used `prefixspan` in order to identify frequently occurring patterns in the sequences. Our approach was to search, for each genre the frequent patterns in it. We found out that in genres like "emo" with patterns like `[11, 9, 9, 11]` highlight a recurring motif, while recurring similar patterns across different genres like the ones shared between "happy" and "kids" could suggest similarity in style elements. We also plotted those recurring patterns as if they were motifs in order to understand them more:

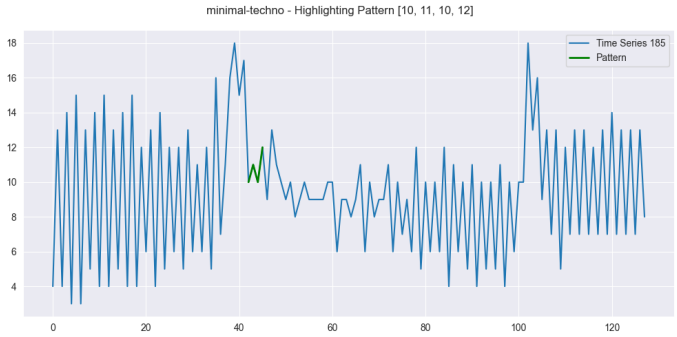


Figure 27: Minimal-techno most frequent longest pattern

It's worth to mention that the explainability of the results was limited by the approximation applied to the time series. Through the analysis of plots and patterns values we had these insights:

- There are some genres that are completely different and with complex rhythmic structures those are **emo**, **folk**, **mpb**, **piano**, **world-music** genres.
- Some genres are more simple and with repetitive patterns kids, honky-tonk, songwriter and newage. Their patterns like [8, 8, 7, 9] or [7, 9, 7, 9] show stable and repetitive rhythms.
- Genres like Heavy-metal, goth or opera show more intense melodic with generally higher values in patterns.

3 Anomaly detection

3.1 Data preparation

Before performing the outlier detection methods, we created a subset with 15 numerical columns (duration_ms, popularity, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature, n_beats, start_of_fade_out and overall_confidence) that we considered the most usefull for the study, then we normalized the dataset with StandardScaler.

3.2 Outlier detection

We identified the top 1% of outliers using 6 different methods from 4 different families: density-based, angle-based, ensemble and model-based. We compared the results and visualized the outliers using two dimensionality reduction techniques PCA and t-SNE. We aggregated the results of all these methods and with a majority vote we created the final subset of outliers.

3.3 Original dataset

Before applying the outlier detection techniques, we performed an initial examination of the dataset to understand the distribution of the variables. This involved creating boxplots for the most significative variables to visually assess their distributions and identify any potential outliers. The boxplots provided a clear representation of the data spread and highlighted the presence of some extreme values that might be considered outliers.

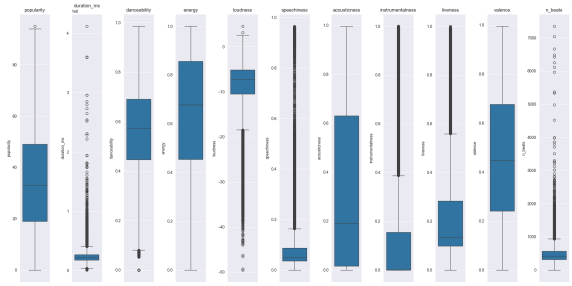


Figure 28: Boxplots of the original dataset

As discussed in subsection 1.1.2, we analyzed the dataset to determine the number of outliers. We found that many features had zero values. For some features, such as popularity, mode, and various confidence measures, these zero values are considered normal. However, for other features like danceability, energy, and speechiness, zero values are a red flag.

After examining the zero values in the original dataset, we proceeded to analyze the zero values in the dataset after removing outliers identified by various detection methods. This step was crucial to understand the impact of outlier removal on the prevalence of zero values.

Furthermore, in the section analyzing outlier detection methods, we will highlight the improvement in reducing zero values compared to the original dataset. This analysis will demonstrate the effectiveness of our chosen outlier detection methods in enhancing data quality by addressing the issue.

3.4 Results and Discussion of the Outlier Detection Methods

In this section, we discuss the results obtained from applying various detection methods.

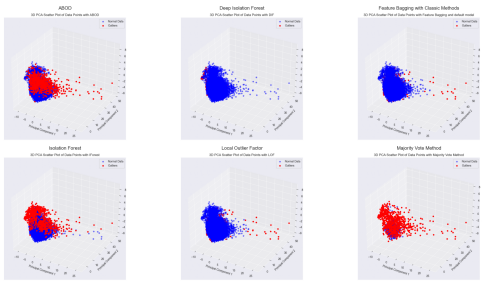


Figure 29: 3D PCA Visualization of the various methods

Each method, passed a parameter tuning process to optimize their performances. We can see that each method detect different elements as outliers, with only 10 indexes that are identified as outliers by every method used. Also 49 rows are found by ABOD, Deep Isolation Forest and Feature Bagging. For each table of outliers found we analyzed the distribution and found that Isolation Forest finds outliers with the highest **'liveness'** values in comparsion to the other methods, LOF and Feature Bagging find outliers with higher values of **'energy'** and **'danceability'** so more intense tracks. Upon analyzing the results from each method, it was evident that no single method was universally superior. So in order to leverage the strengths of each individual outlier detection method and mitigate their weaknesses, we developed a majority voting method. This approach combined the outliers identified by all the individual methods and used a majority voting mechanism to determine the final set of outliers. Here's how the method was implemented:

- **Aggregation of outliers**: We first collected the indices of outliers identified by each method
- **Voting Mechanism**: For each data point, we counted the number of methods that flagged it as an outlier.
- **Threshold Setting**: We set a threshold for the minimum number of votes required for a data point to be considered an outlier.

The outliers found with this method successfully identified outliers that exhibit extreme or unusual values, characterized by unusually long duration's, very low or very high danceability, low energy levels and very low loudness levels.

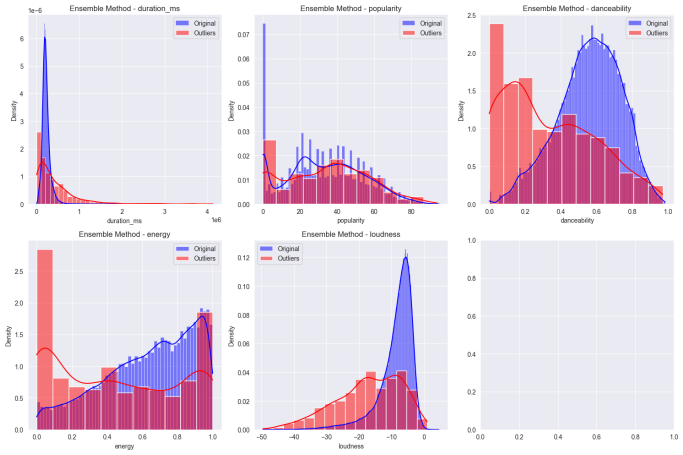


Figure 30: Distribution of outliers

We decided to drop these outliers, and reached a new situation of this kind:

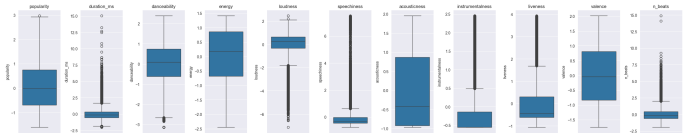
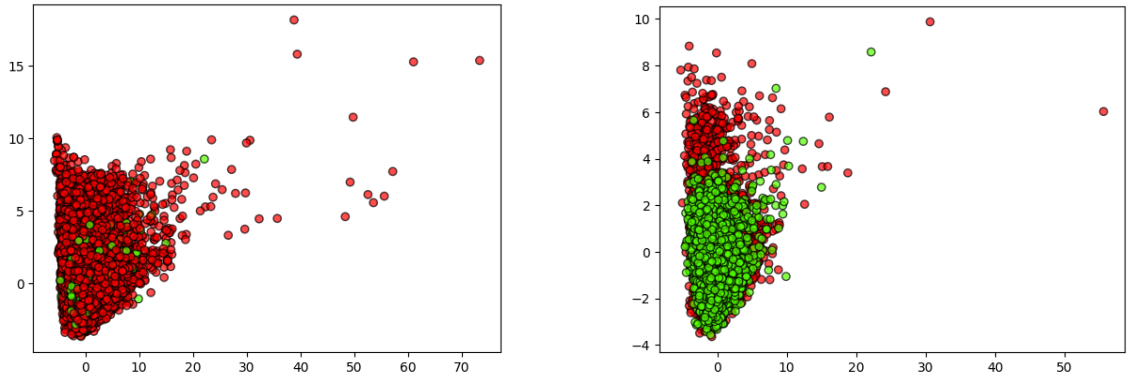


Figure 31: Boxplots after removing the top 1% outliers

4 Imbalanced Learning

We chose the *explicit* feature from our cleaned tracks dataset, which is already highly unbalanced, with a wide majority of *False* instances compared to *True*. We performed various oversampling, undersampling, and mixed techniques to try to balance the attribute, and performed classification with Decision Tree Classifier and KNN. We dropped information about the release date of the track because we had some missing values and believed that the release date of the track does not affect whether or not the song is explicit.



(a) (b)

Figure 32: Distribution of explicit attribute before and after applying Random Undersampler

4.1 Undersampling

The accuracy with no technique is really high, but we cannot rely on this result because of the distribution of the target value. With techniques like Random Undersampler, we can see that the model becomes less conservative, so it catches more positives but also includes more false positives. The same applies to *Undersampling by Cluster Centroids*, but we noticed that Decision Tree had a drop in recall for the False class. We think that this happened because the Decision Tree Classifier overfits on centroids, reducing its generalization capability, while KNN still performs comparably to the other techniques, tending to be more robust with centroids. *Tomek Links* and *Edited Nearest Neighbours* do not guarantee a balanced output distribution; the results are quite similar to those with no technique. Condensed Nearest Neighbour, after more than 160 minutes of execution, did a decent job undersampling the dataset, but still kept the dataset unbalanced. It performed almost like the original dataset with Decision Tree, we can notice an increase in recall for the *True* class while the *False* class slightly decreased. On the other hand, it performed really well with KNN, with high accuracy and decent performance in precision for the *True* class, while recall did not increase much. We think this combination could be a good choice if we need a reliable decision because we have fewer false positives.

4.2 Oversampling

We performed *SMOTE* and *ADASYN* for oversampling. We noticed that the resulting distribution with SMOTE was perfectly balanced, while with ADASYN it was slightly unbalanced. Overall

performance is similar to the performance with the original dataset, with a small increase in recall for the minority class. We also noticed that the performance of SMOTE and ADASYN is almost the same for both Decision Tree (DT) and K-Nearest Neighbors (KNN) models.

4.3 Mixed

Given the disappointing performance of Tomek Links and ENN, we applied *SMOTEENN* and *SMOTETomek*, mixed techniques that oversample the minority class with SMOTE and undersample the majority class with ENN or Tomek Links. Comparing performance with no technique, precision dropped for the minority class, while recall increased, just like the other techniques. However, SMOTEENN performed with one of the highest AUC scores, which is due to the increase of recall in minority class.

Technique	**Distribution (F,T)**	**Model**	**Accuracy**	**Precision (F,T)**	**Recall (F,T)**	**f1-score (F,T)**	**AUC**
None	74205, 7020	DT	88	93, 31	94, 28	94, 30	65
None	74205, 7020	KNN (k=12)	92	92, 59	99, 12	96, 19	78
Random Undersampler	4914, 4914	DT	68	96, 17	69, 68	80, 27	72
Random Undersampler	4914, 4914	KNN (k=27)	74	83, 33	74, 75	84, 33	72
Condensed Nearest Neighbors	10958, 4914	DT	78	94, 19	82, 45	87, 27	66
Condensed Nearest Neighbors	10958, 4914	KNN (k=27)	91	93, 47	97, 26	95, 34	82
Undersampling by Cluster Centroids	4914, 4914	DT	34	96, 11	29, 89	44, 19	60
Undersampling by Cluster Centroids	4914, 4914	KNN (k=29)	78	96, 24	79, 70	87, 35	83
TomekLinks	50634, 4914	DT	88	93, 29	93, 30	93, 30	64
TomekLinks	50634, 4914	KNN (k=23)	91	92, 58	99, 15	96, 23	80
Edited Nearest Neighbours	43429, 4914	DT	86	94, 29	90, 41	92, 34	69
Edited Nearest Neighbours	43429, 4914	KNN (k=1)	85	94, 27	90, 39	92, 32	65
SMOTE	51943, 51943	DT	81	94, 21	85, 42	89, 28	67
SMOTE	51943, 51943	KNN (k=2)	84	94, 24	89, 37	91, 29	69
ADASYN	52478, 51943	DT	80	94, 21	84, 44	89, 21	67
ADASYN	52478, 51943	KNN (k=1)	82	94, 22	86, 43	90, 29	64
SMOTEENN	51808, 36608	DT	76	96, 21	77, 63	86, 31	73
SMOTEENN	51808, 36608	KNN (k=2)	78	95, 22	81, 58	87, 32	73
SMOTETomek	51928, 51928	DT	81	94, 21	85, 42	89, 28	67
SMOTETomek	51928, 51928	KNN (k=2)	84	94, 24	89, 37	91, 29	69

Table 3: Outcomes of different Undersampling/Oversampling techniques with Decision Tree classifier and KNN

4.4 Conclusions

We noticed that most of the time when Recall increases Precision decreases and vice versa. Tomek Links and ENN are not good alone with our dataset, but in combination with SMOTE they perform better. The overall best in our opinion is Undersampling by Cluster Centroids with KNN with an AUC of 83% and the best balance in recall for the two classes. The worst is still Undersampling by Cluster Centroids but with Decision Tree because of that drop in Recall for False class and the lowest accuracy.

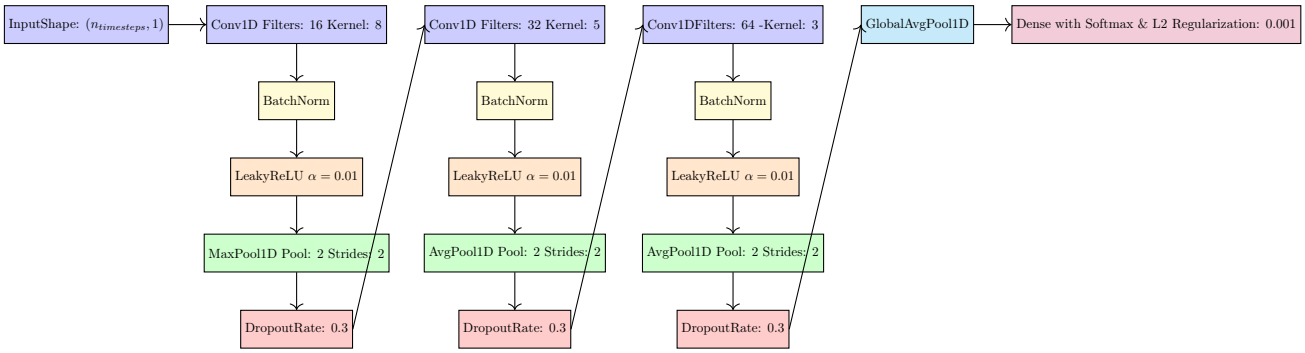
5 Advanced Time Series Classification

In this section, we'll extend our time series classification by incorporating deep neural networks. Specifically, we'll evaluate **Convolutional Neural Networks** (CNNs) and **MiniRocket**. We also tested **ResNet**, but it yielded poorer results compared to CNN and MiniRocket, achieving an accuracy of only 0.21. Given its longer training time, we opted not to delve further into hyperparameter tuning and instead focused our analysis on CNN and MiniRocket. We split our data into 80% for training and 20% for testing, ensuring class distribution remained equal. Both sets were then normalized using StandardScaler.

5.1 Convolutional Neural Network

For our network architecture, we employed three convolutional layers: the initial layer with 16 filters and a kernel size of 8, the second with 32 filters and a kernel size of 5, and the third with 64 filters and a kernel size of 3. Each layer incorporates **BatchNormalization**, **LeakyReLU** as activation function with an alpha value of 0.01, **dropout regularization** set at a rate of 0.3, and

weight initialization ('he_normal').
After the initial layer, a **MaxPooling** layer was incorporated to capture the most salient features, such as the peak values in the spectral centroid of the song. This addition aims to enhance the network's resilience to minor fluctuations or noise while reducing the computational burden for subsequent layers through downsampling, thereby mitigating overfitting.
Following the first convolutional layer, both the second and third layers were augmented with an **AveragePooling** layer. This inclusion was intended to refine the feature maps, facilitating the capture of broader trends or patterns within the data over time.
Following the third convolutional layer, we added **GlobalAveragePooling**, and subsequently, a fully connected layer with **softmax activation** and **L2 regularization** with 0.001. Similarly, weight initialization using 'he_normal' was applied here as well.
As for optimization, we adopted **sparse_categorical_crossentropy** as the loss function and utilized the **Adam optimizer** with a learning rate set to 0.001. We also experimented with setting the dilation rate to 2 and padding to 'same'. Additionally, we tested dilation rates of 4, 8, and 16 for the three convolutional layers, but these resulted in poorer performance. Adding an extra convolutional layer also led to worse results. Below we present the network architecture through a simple diagram.



5.1.1 Parameters

- **LeakyReLu:** Used for hidden layers, it adds a small computational complexity but can help in dealing with the **dying relu problem** where neurons might get stuck in an inactive state.
- **he_normal:** As for the weights, we used he_normal initialization. It draws samples from a truncated normal distribution centered on 0. Networks with He initialization tend to converge faster and reach better performance compared to other initialization methods [1].

5.1.2 Training Parameters and Results

Regarding the training of the network, we implemented **early stopping** with a patience of 50, monitoring the validation loss. The model was trained for 150 epochs with a batch size of 32 and a validation split of 0.3. Training concluded after 150 epochs, achieving an accuracy of 0.41 on both the training and validation sets. When tested on the test set, the model also achieved an accuracy of 0.41, indicating no significant overfitting. The results, displayed in the following table, reveal some discrepancies between class precision and recall:

Genre	Precision	Recall	F1-Score	Support
emo	0.47	0.20	0.28	100
folk	0.50	0.01	0.02	100
goth	0.23	0.03	0.05	100
happy	0.72	0.67	0.69	100
heavy-metal	0.36	0.74	0.49	100
honky-tonk	0.35	0.56	0.43	100
j-idol	0.33	0.64	0.44	100
kids	0.62	0.05	0.09	100
minimal-techno	0.53	0.68	0.60	100
mpb	0.17	0.38	0.23	100
new-age	0.39	0.50	0.44	100
opera	0.34	0.64	0.44	100
piano	0.72	0.18	0.29	100
progressive-house	0.55	0.32	0.41	100
salsa	0.78	0.62	0.69	100
sertanejo	0.37	0.32	0.34	100
sleep	0.55	0.81	0.66	100
songwriter	0.47	0.22	0.30	100
synth-pop	0.33	0.44	0.37	100
world-music	0.43	0.15	0.22	100
Accuracy			0.41	2000
Macro Avg	0.46	0.41	0.37	2000
Weighted Avg	0.46	0.41	0.37	2000

Table 4: CNN Classification Report

5.2 MiniRocket

We experimented with the MiniRocket algorithm using various kernel sizes (100, 1000, 10000) and dilation rates (8, 16, 32) to improve accuracy. However, the highest accuracy achieved was 0.41 using the default parameters.

Class	Precision	Recall	F1-Score	Support
emo	0.34	0.28	0.31	100
folk	0.15	0.09	0.11	100
goth	0.08	0.04	0.05	100
happy	0.71	0.77	0.74	100
heavy-metal	0.39	0.55	0.46	100
honky-tonk	0.42	0.50	0.45	100
j-idol	0.36	0.37	0.36	100
kids	0.28	0.25	0.26	100
minimal-techno	0.61	0.67	0.64	100
mpb	0.19	0.12	0.15	100
new-age	0.40	0.51	0.45	100
opera	0.46	0.55	0.50	100
piano	0.39	0.35	0.37	100
progressive-house	0.48	0.52	0.50	100
salsa	0.51	0.71	0.60	100
sertanejo	0.30	0.38	0.34	100
sleep	0.69	0.66	0.68	100
songwriter	0.27	0.19	0.22	100
synth-pop	0.33	0.35	0.34	100
world-music	0.31	0.29	0.30	100
Accuracy			0.41	2000
Macro avg	0.38	0.41	0.39	2000
Weighted avg	0.38	0.41	0.39	2000

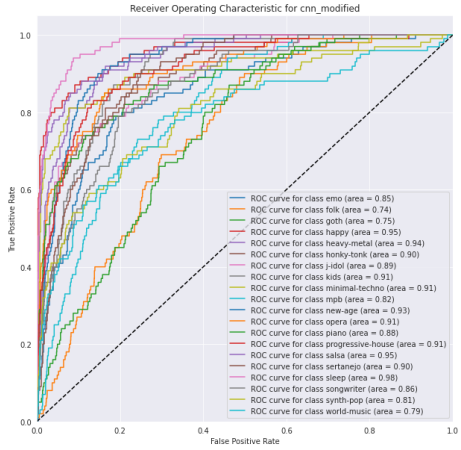
Table 5: MiniRocket Classification Report

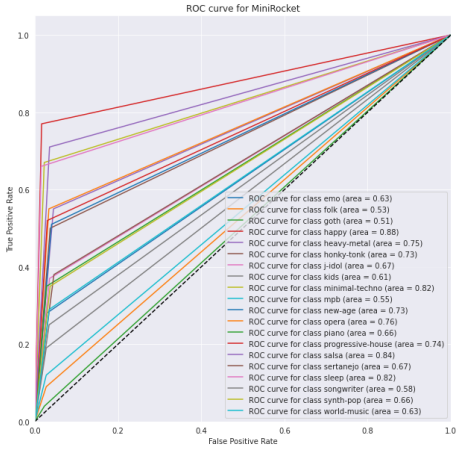
The data in the table indicates that the model's overall performance is poor, with some exception such as "happy", "minmal-techno" and "sleep" genres. The model underperforms across most classes. Notably, there is no evidence of overfitting.

5.2.1 CNN - Minirocket Comparison

Both models performed well on the "happy" and "minimal-techno" classes, suggesting that these classes have a structure that both models can recognize. Overall, Rocket appears to outperform the CNN in terms of training time, as it required significantly less time than the CNN, which needed extensive hyperparameter tuning. Each CNN training run averaged 20 minutes, and multiple runs

were necessary to optimize the parameters for better performance. However, examining the ROC curves displayed below, our CNN ultimately performs better than MiniRocket, as the AUC for all classes is higher for the CNN compared to MiniRocket.





6 Advanced Tabular Classification

In this section, we will classify the genres of songs in the track dataset using various approaches: Logistic Regression, Support Vector Machines, Neural Networks, Ensemble Methods, and Gradient Boosting Machines. We will use the dataset cleaned of outliers and pre-processed as described earlier. Additionally, we removed the following irrelevant features: 'id', 'name', 'artists', 'album_name', 'n_bars', 'month', and 'day'. The explicit feature was one-hot encoded. Since we previously agglomerated songs with the same id but different genres into a single entry with a list of genres, we will retain only the first genre in the list for classification. We also normalized the dataset using the StandardScaler. Random_state for each classifier has been set to 42 for reproducibility purpouses. After selecting the hyperparameter values, we initiated a GridSearchCV with cv=5, which took approximately 2 hours to complete. The analysis identified the optimal parameters as follows: 'C': 10, 'class_weight': None, 'multi_class': 'multinomial', 'penalty': 'l2', 'solver': 'newton-cg'. Subsequent predictions yielded disappointing results, with an accuracy of 0.28 and very low precision and recall for several classes, including some with values as low as 0.. Furthermore, as detailed in the logistic regression subsection, we aggregated genres into broader categories and excluded songs with genres that couldn't be grouped with others. The grouping process involved substituting genres containing certain listed keywords with broader categories, which are listed as follows:

- **Pop:** 'pop', 'r&b', 'soul', 'funk', 'reggae', 'r-n-b', **with 9016 entries**
- **Rock:** 'rock', 'punk', 'alternative','garage', 'grunge', 'ska', with **7332 entries**
- **Metal:** 'metal', 'grindcore', with **4521 entries**.
- **Electronic:** 'electronic', 'house', 'techno', trance','electro','dub', 'dubstep', 'industrial', 'drum-and-bass', with **9811 entries**.
- **National:** 'british', 'swedish', 'spanish', 'brazil', 'indian', 'iranian', 'german', 'french', 'turkish', with **7493 entries**.
- **All the rest has been discarded.**

6.1 Logistic Regression

In order to tune the hyper-parameters for the logistic regressor classifier we used the following parameters:

Each parameter was selected based on the official documentation from scikit-learn, chosen for their

Parameter	Values
`C`	0.1, 1, 10
`penalty`	'l2'
`solver`	'sag', 'newton-cg', 'saga', 'lbfgs'
`class_weight`	'balanced', None
`multi_class`	'multinomial', 'ovr'

Table 6: Parameter grid used for hyperparameter tuning.

potential relevance to our experimentation. After selecting the hyperparameter values, we initiated a GridSearchCV with cv=5, which took approximately 2 hours to complete. The analysis identified the optimal parameters as follows: 'C': 10, 'class_weight': None, 'multi_class': 'multinomial', 'penalty': 'l2', 'solver': 'newton-cg'. Subsequent predictions yielded disappointing results, with an accuracy of 0.28 and very low precision and recall for several classes, including some with values as low as 0. Therefore, as outlined in the introduction to this section, we conducted genre grouping. Following the grouping, we reran the grid search, completing computations in just 2 minutes. This time, we obtained different parameters: 'C': 0.1, 'class_weight': 'balanced', 'multi_class': 'multinomial', 'penalty': 'l2', 'solver': 'sag'. Subsequently, we applied the classifier to the test set and achieved the following results: These results are significantly improved compared to the

Genre	Precision	Recall	F1-score	Support	AUC
Pop	0.52	0.46	0.49	1803	0.79
Rock	0.46	0.54	0.49	1323	0.81
Electronic	0.70	0.67	0.68	1962	0.88
Metal	0.57	0.81	0.67	904	0.94
National	0.50	0.40	0.44	1499	0.77
Accuracy			0.56	7491	
Macro avg	0.55	0.57	0.56	7491	
Weighted avg	0.56	0.56	0.55	7491	

Table 7: Logistic Regressor - Classification report

previous ones. The analysis is now more clear and understandable since we are dealing with only 5 classes, and the computation time has been greatly reduced. Additionally, the classifier's performance appears to be quite strong, as evidenced by the ROC curve below, where each class achieves an AUC above 0.77.

6.2 SVM

Here are the parameters used for the grid search, listed in the following table: After approximately

Parameter	Values
`Kernel`	linear, poly, rbf, sigmoid
`C`	0.1, 1, 10
`degree (poly)`	2, 3, 4, 5
`coeff0 (poly/sigmoid)`	0, 0.1, 0.5, 1
`gamma (rbf/sigmoid)`	0.001, 0.01, 0.1, 1

Table 8: Parameter grid used for hyperparameter tuning.

1 hour of computation with a 5-fold cross-validation grid search, the best parameters identified were: kernel: 'rbf', C: 1, gamma: 0.1. Applying the classifier to the test set yielded the following results:

In addition to achieving higher accuracy compared to the Logistic Regressor, we can see that the AUC is also significantly better, with all classes showing an AUC above 0.83.

	Precision	Recall	F1-Score	Support	AUC
Pop	0.57	0.62	0.60	1803	0.84
Rock	0.54	0.59	0.57	1467	0.84
Electronic	0.73	0.75	0.74	1962	0.91
Metal	0.71	0.73	0.72	904	0.95
National	0.60	0.44	0.51	1499	0.83
Accuracy			0.63	7635	
Macro Avg	0.63	0.63	0.63	7635	
Weighted Avg	0.63	0.63	0.63	7635	

Table 9: SVM - Classification report

6.3 Random Forest

Here are the parameters used for the grid search with cv=3, listed in the following table which yielded the following best combination of paramaters 'bootstrap': False, 'criterion': 'gini', 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 5, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 500:

Parameter	Values
`n_estimators`	100, 300, 500
`criterion`	gini, entropy, log_loss
`max_depth`	None
`min_samples_split`	2, 5, 10
`min_samples_leaf`	1, 2, 4
`min_weight_fraction_leaf`	0.0, 0.1, 0.3, 0.5
`max_features`	auto, sqrt, log2
`bootstrap`	True, False
`random_state`	42
`n_jobs`	-1

Table 10: Parameter grid used for hyperparameter tuning.

Furthermore, there is notable enhancement in accuracy and AUC. Interestingly, the model places greater reliance on the confidence associated with features such as key, mode, and time_signature rather than the features themselves. We observe that acousticness is the most significant feature, likely because it effectively distinguishes between our five classes. For instance, if we examine the mean acousticness for each genre, pop and national have values above 0.32, while the rest are below 0.20. In terms of danceability, electronic music scores above 0.64, whereas the rest are all below 0.59, with metal at 0.36. Regarding popularity, rock and pop have means over 0.41, while the others are below 0.35. Lastly, for instrumentalness, electronic and metal genres have means above 0.27, while the other three genres are below 0.15. Regarding the number of estimators used, we also tested 1000 and 5000 estimators but received the same classification report.

Class	Precision	Recall	F1-Score	Support
Pop	0.61	0.67	0.64	1803
Rock	0.59	0.63	0.61	1467
Electronic	0.75	0.79	0.77	1962
Metal	0.77	0.76	0.77	904
National	0.66	0.49	0.56	1499
Accuracy			0.67	7635
Macro Avg	0.67	0.67	0.67	7635
Weighted Avg	0.67	0.67	0.67	7635

Table 11: Random Forest - Classification report

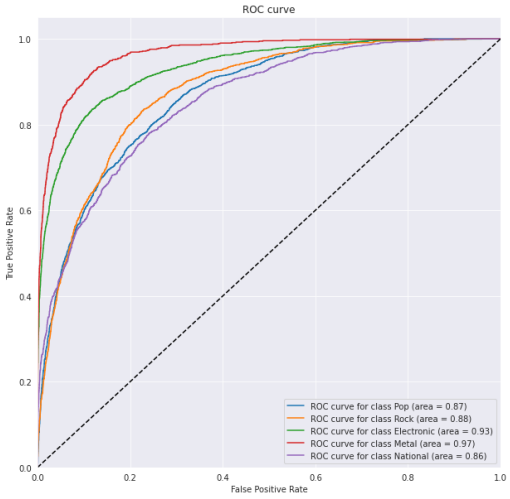


Figure 33: RandomForest ROC

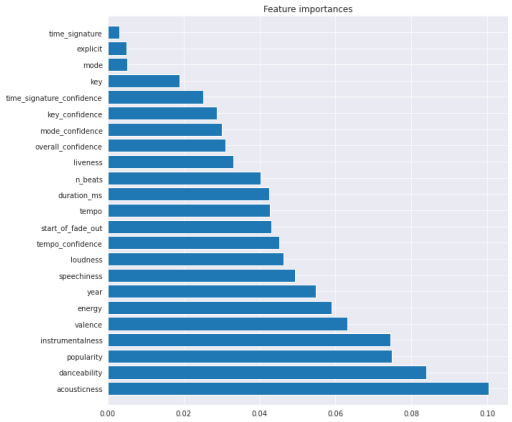


Figure 34: RandomForest Feature Importances

6.4 Bagging and Boosting

We also tried both Bagging and AdaBoost, but they performed worse than the others model used. We also attempted to run the RandomForest using both bagging and boosting techniques with n_estimators=100. However, the performance was slightly worse, achieving an accuracy of 0.65. Additionally, it's noteworthy that a laptop with 16GB of RAM was insufficient for executing bagging and boosting with RandomForest, necessitating the use of a laptop with 32GB.

6.5 Gradient Boosting Machines: XGBoost & LightGBM

Due to the significant increase in the number of parameters to be chosen, we employed a RandomizedSearch for tuning with cv=3 and 100 iterations. This approach was necessary because GridSearch would require excessive time and exceed our RAM limits. The parameters and their tested values are as follows:

Parameter	Values
`objective`	multi:softmax, multi:softprob
`num_class`	5
`booster`	gbtree, dart, gblinear
`learning_rate`	0.01, 0.1
`max_depth`	-1, 8, 16 ,32
`subsample`	0.5, 0.75, 1.0
`sampling_method`	uniform, gradient_based
`lambda`	0.0, 0.5, 1
`tree_method`	auto, exact, approx, hist
`refresh_leaf`	0, 1
`process_type`	default, update
`multi_strategy`	one_output_per_tree, multi_output_tree
`random_state`	42

Table 12: Parameters for XGBoost

Parameter	Values
`boosting_type`	gbdt, goss, dart
`max_depth`	-1, 8, 16, 32
`num_leaves`	31, 62, 128
`n_estimators`	100, 200, 300
`subsample_for_bin`	200000, 400000, 600000
`objective`	multiclass, multiclassova
`num_class`	5
`reg_alpha`	0.0, 0.5, 1.0
`reg_lambda`	0.0, 0.5, 1.0
`learning_rate`	0.01, 0.1
`random_state`	42

Table 13: Parameters for LightGBM

The best parameters identified are:

- **XGBoost:** 'tree_method': 'exact', 'subsample': 0.75, 'sampling_method': 'uniform', 'refresh_leaf': 0, 'random_state': 42, 'process_type': 'default', 'objective': 'multi:softmax', 'num_class': 5, 'multi_strategy': 'one_output_per_tree', 'max_depth': 16, 'learning_rate': 0.1, 'lambda': 0.5, 'booster': 'gbtree'
- **LightGBM:** 'subsample_for_bin': 200000, 'reg_lambda': 0.0, 'reg_alpha': 0.0, 'random_state': 42, 'objective': 'multiclassova', 'num_leaves': 128, 'num_class': 5, 'n_estimators': 300, 'max_depth': -1, 'learning_rate': 0.1, 'data_sample_strategy': 'bagging', 'boosting_type': 'gbdt'

Yielding the following classification reports:

Class	Precision	Recall	F1-Score	Support	AUC
Pop	0.63	0.66	0.64	1803	0.88
Rock	0.59	0.64	0.62	1467	0.89
Electronic	0.77	0.79	0.78	1962	0.94
Metal	0.77	0.78	0.77	904	0.97
National	0.62	0.52	0.57	1499	0.86
Accuracy			0.67	7635	
Macro Avg	0.68	0.68	0.68	7635	
Weighted Avg	0.67	0.67	0.67	7635	

Table 14: Classification report for XGBoost

Class	Precision	Recall	F1-Score	Support	AUC
Pop	0.62	0.67	0.64	1803	0.87
Rock	0.60	0.64	0.62	1467	0.88
Electronic	0.77	0.80	0.78	1962	0.94
Metal	0.81	0.77	0.79	904	0.97
National	0.63	0.52	0.57	1499	0.86
Accuracy			0.68	7635	
Macro Avg	0.69	0.68	0.68	7635	
Weighted Avg	0.68	0.68	0.68	7635	

Table 15: Classification report for LightGBM

6.6 CatBoostClassifier

We tuned CatBoostClassifer using a GridSearchCV with these values `iterations [100, 500, 1000, 2000]`, `depth [2, 4, 6, 8, 10]`, `learning_rate [0.01, 0.1, 0.3]`, `loss_function ['MultiClass']`. We ended choosing as best parameters based on accuracy result, iterations 1000, depth 8 and learning_rate 0.1. The performance of the classifier was almost identical to those of RandomForest.

Class	Precision	Recall	F1-Score	Support	AUC
Pop	0.63	0.65	0.64	1803	0.87
Rock	0.60	0.64	0.62	1467	0.88
Electronic	0.77	0.80	0.79	1962	0.94
Metal	0.78	0.79	0.78	904	0.97
National	0.62	0.51	0.56	1499	0.86
Accuracy			0.68	7635	
Macro Avg	0.68	0.68	0.68	7635	
Weighted Avg	0.68	0.68	0.67	7635	

Table 16: Classification report for CatBoostClassifier

We also noticed that the Metal class is the best predicted class, just like in the time-series classification task. The top feature just like the RandomForest are popularity, instrumentalness and danceability, these features have the highest importance scores, but we will deeply talk about this in the Explainability part.

6.7 Neural Network - Kolmogorov Arnold Network

Having employed a convolutional neural network for advanced time series classification, we decided to explore a more recent model called KAN [4], for which we provide a link to its related article through the bibliography. The primary motivation for experimenting with this new model lies in its three key advantages:

- **Accuracy:** It is expected to achieve higher accuracy with fewer parameters compared to traditional Multi-Layer Perceptrons (MLPs). The main reason is the learnable activation functions on the edges, which enable KANs to capture complex patterns and relationships in the data more effectively.
- **Interpretability:** It offers substantial improvements in interpretability over traditional MLPs.

- **Scalability:** The model benefits from more favorable scaling laws, owing to its ability to decompose complex functions into simpler, univariate functions. This allows it to achieve lower error rates with increasing model complexity more efficiently than MLPs.

We built it using a first layer made up of 23 input nodes (one for each feature), a second layer with 11 nodes, and lastly the output layer made up of the 5 output classes. Using grid=3, k=3 and seed=42 for reproducibility. The training was performed for 50 steps, with "LBFGS" optimizer, learning rate =0.001, lamb=0.01, lamb_entropy=10 and CrossEntropyLoss as the loss function. We didn't have time to perform hyperparameter tuning due to the project deadline. Several factors contributed to this: we spent a significant amount of time on initial trials, the training process was slow (approximately 30 minutes per run), and a considerable amount of time was lost in function fixing. The function fixing was done using the model.auto_symbolic() method, which attempts to find the best function for each connected node. This process takes approximately 1 hour. Additionally, a bug [2] in the library forced us to repeat the process multiple times before it was successfully completed. The bug was related to the functions fixed by the method, as some of them were not recognized. To solve the issue, we tested on a smaller dataset and used only the functions that worked on that smaller dataset (we used the following: $[x, x^2, x^3, x^4, exp, 1/x^2, cosh, sin, abs]$). At the end of this process, we aimed to extract the mathematical formula used to predict the various classes. However, the resulting formula was very large (3681 characters long), so we will not report it here. This might be due to the fact that we used a restricted set of functions. Overall the accuracy was not that stunning, performing worse than any model previously employed. We report the classification report below:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Pop	0.44	0.55	0.49	1803
Rock	0.43	0.48	0.46	1467
Electronic	0.65	0.68	0.67	1962
Metal	0.64	0.71	0.67	904
National	0.57	0.27	0.37	1499
Accuracy			0.53	7635
Macro Avg	0.55	0.54	0.53	7635
Weighted Avg	0.54	0.53	0.53	7635

Table 17: KAN - Classification Report

Lastly, one of the other noteworthy features is the ability to visualize the different functions used across the various links between nodes, as demonstrated in the example [3]. However, when we used the model.plot() method, we got the following result:

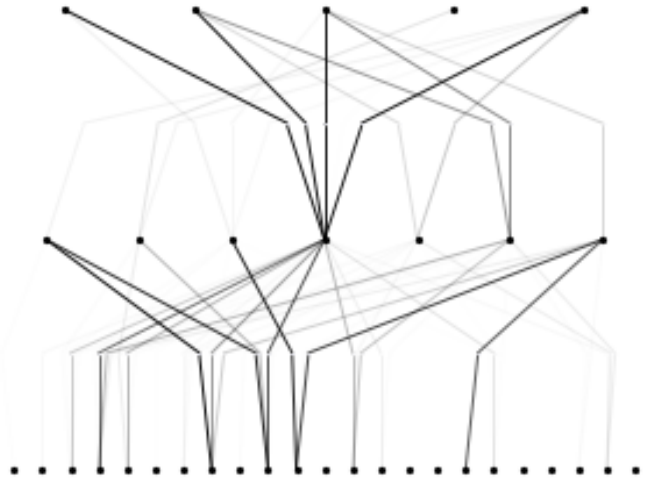


Figure 35: Plotting KAN

6.8 Conclusions

In conclusion, both Random Forest and Gradient Boosting Machines perform exceptionally well compared to other models. However, their primary drawback is the extensive time and resources required for hyperparameter tuning. A significant advantage of Random Forest is its ability to enhance interpretability through feature importance. Gradient Boosting Machines offer very fast training times, slightly better accuracy, and a wide range of parameters to choose from. This extensive parameter pool can be particularly beneficial for fine-tuning to further improve model accuracy.

7 Advanced Tabular Regression

We choose to predict popularity feature from tracks dataset, because we think that can be interesting to predict if a song will become popular knowing its features. First we tried with features only from tracks dataset, and after that we merged with artist dataset to try to improve the performance with more data.

7.1 Random Forest

We performed Random Forest with GridSearchCV with cv=5 and after more than 180 minutes of execution ended up with best parameters as follows: 'bootstrap': False, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200. We tried n_estimators = 50, 100, 200 and since it took so long to discover that 200 was the best number of estimators we decided to fix the other parameters to the best found and explored results varying just n_estimator = 200, 300, 400. The best n_estimator resulted 400.

7.2 XGBoost

We performed XGBoost with search grid and cv=3 and surprinsingly it took less than 30 minutes. We ended up with best parameters as follows: 'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200, 'subsample': 1.0.

Given that it took not so long, we decided to perform another run with more choice of parameters and cv=5. It ended up with best parameters as follows: 'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'n_estimators': 400, 'subsample': 1.0.

7.3 Scores

Scores between first and second run are not that different. In random forest we can see almost identical results both in errors and r2. We expected that because the only difference is in number of estimators. In XGBoost we expected more difference, mostly because of the learning rate and max depth. But we can see that the differences are barely noticeable.

Model	Parameters	Test Set MSE	Test Set MAE	Test Set R^2	CV MSE	CV R^2
Random Forest	n_estimators: 200 max_depth: None max_features: sqrt min_samples_leaf: 2 min_samples_split: 2 bootstrap: False	0.6500	0.639	0.345	0.6677 ± 0.0059	0.333 ± 0.0059
Random Forest	n_estimators: 400 max_depth: None max_features: sqrt min_samples_leaf: 2 min_samples_split: 2 bootstrap: False	0.6485	0.637	0.347	0.6660 ± 0.0059	0.335 ± 0.0059
XGBoost	colsample_bytree: 0.7 learning_rate: 0.1 max_depth: 7 n_estimators: 200 subsample: 1.0	0.6461	0.632	0.349	0.6651 ± 0.0082	0.336 ± 0.0082
XGBoost	colsample_bytree: 0.7 learning_rate: 0.05 max_depth: 9 n_estimators: 400 subsample: 1.0	0.6364	0.627	0.359	0.6555 ± 0.0077	0.346 ± 0.0077

Table 18: Performance of Different Models (solo tracks)

7.4 Adding artists dataset

So we decided to merge the tracks dataset with artists dataset. We extracted the main artist from *artists* column of tracks dataset, which is a semicolon separated list, taking the first element of the

list in that column. After that we merged the two datasets with this new column of *tracks* dataset and '*name*' column from *artsits* dataset. The only useful columns from *artists* are popularity and followers. We expected that knowing the popularity of the track's artist would be a great hint to guess the popularity of the song, and so it was.

7.5 New scores

Adding artists' information to our dataset increased the performance a lot. We can see that r2 almost doubled both in Random Forest and XGBoost. Also the errors decreased noticeably. We still computed the two runs we performed before with XGBoost, and we noticed that now the upgrade between the two executions is more noticeable, but still very small.

Model	Parameters	Test Set MSE	Test Set MAE	Test Set R^2	CV MSE	CV R^2
Random Forest	n_estimators: 400 max_depth: None max_features: sqrt min_samples_leaf: 2 min_samples_split: 2 bootstrap: False	0.4054	0.482	0.595	0.4018 ± 0.0062	0.598 ± 0.0062
XGBoost	colsample_bytree: 0.8 learning_rate: 0.1 max_depth: 7 n_estimators: 200 subsample: 1.0	0.4122	0.484	0.588	0.4099 ± 0.0016	0.590 ± 0.0016
XGBoost	colsample_bytree: 0.6 learning_rate: 0.05 max_depth: 9 n_estimators: 400 subsample: 1.0	0.3895	0.469	0.611	0.3855 ± 0.0073	0.614 ± 0.0073
MLP	activation: relu alpha: 0.0001 hidden_layer_sizes: (50,) learning_rate: constant max_iter: 200 solver: adam	0.4974	0.544	0.503	0.4841 ± 0.0018	-

Table 19: Performance of Different Models (tracks + artists)

7.6 MultiLayer Perceptron

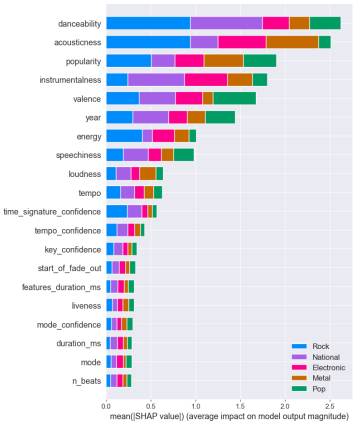
We decided to perform a grid search with MLP regressor with this parameters grid: 'hidden_layer_sizes': [(50,), (100,), (50,50), (100,50)], 'activation': ['relu', 'tanh'], 'solver': ['adam', 'sgd'], 'alpha': [0.0001, 0.001, 0.01], 'learning_rate': ['constant', 'adaptive'], 'max_iter': [200, 500, 1000] with a cross validation = 3.

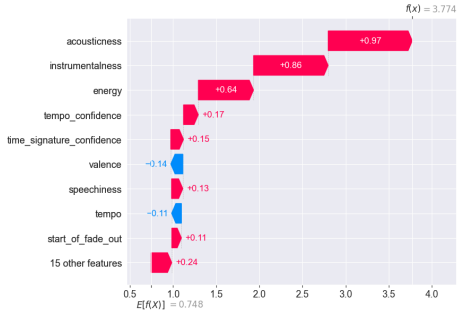
After a long execution of about 315 minutes we ended up with these best parameters: 'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'. We performed just one shot with merged dataset, and the scores are not better than XGBoost nor Random Forest. So we was very disappointed by this performance.

8 Explainable AI

8.1 SHAP

We used SHAP to understand why CatBoostClassifier classified our multiclass problem the way it did. Through SHAP we had the possibility to see the average contribution of a feature value among all its possibile combinations. Through this summary plot 8.1 we can analyze the importance of different features in predicting five music genres: Rock, National, Electronic, Metal and Pop. We noticed that the model relies heavily on features like danceability, acousticness, and popularity to differentiate between different genres probably because these features capture key characteristics that distinguish one genre from another. We also studied the summary plot of each genre and found out that, as expected for **rock** an high danceability increases the predictions, while acousticness and instrumentalness decrease them. For **Electronic** songs the most influential features for predicting the class are acousticness; positively impacting when high and instrumentalness; negatively impacting when high.





We also studies the interaction between pair of features and their importance. The top interactions found are popularity and year suggesting that a combination of a song popularity and its release year significantly impact the model's prediction, probably this is cause of the changing trends over the years. Popularity has a central role in the predictions, this can reflect the ability of the model to understand and predict listener preferences, knowing that different genres might have varying levels of inherent popularity. For example Pop music tends to be more popular and mainstream than national music. Then we analyzed a random instance correctly predicted as rock, we can see that in this case 8.1, as saw before in the the figure 8.1, acousticness, instrumentalness and energy have the most important contributions to the final prediction of the model; while valence and tempo slightly lowered it.

8.2 LORE

We also tested the LOcal Rule-based Explainer in order to better understand the model's decision-making process for a given instance. This time we focused on the prediction of an instance which genre is Pop. This enriches our understanding of the model's decision making process and the importance of features. As expected from a pop song an high Valence suggests that the song is perceived as positive or cheerful, this is also according to **SHAP**, in fact in the feature importance worked out with shap we had been able to see that Valence was the most important value in predicting a song like Pop. Significant changes in valence (to be very low), higher danceability, and increased energy and instrumentalness are required to shift the prediction to genre 4 (Rock). This suggests that the rock genre is characterized by more danceable, energetic, and instrumental music.

8.3 LIME

Finally we tried to understand a prediction of an Electronic song using LIME which provides a detailed breakdown of the prediction probabilities of each class along with visual explanations.

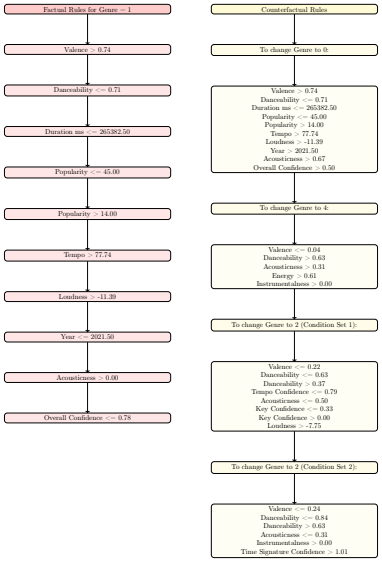


Figure 36: Lore Factual and Counterfactual Rules for Genre Prediction

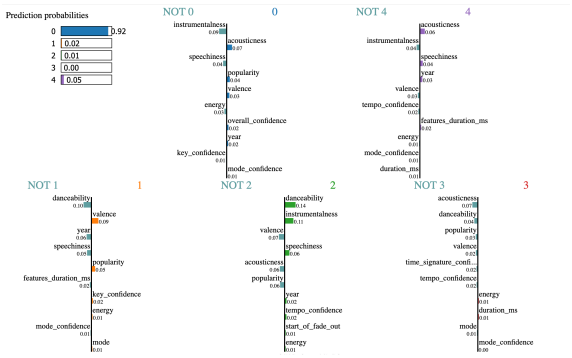


Figure 37: Probabilities of predicting and not predicting each class

We can see that the features with the highest absolute values have the greatest impact on the prediction. For this instance, danceability, valence, and year are highly influential. Just as expected from the SHAP summary plot. The instance is most likely predicted as genre 0 with a probability of 0.92. Genre 4 is the next probable with a much lower probability of 0.05. So we can say that the classifier had no doubts in classifying this instance. Based on LIME and SHAP explanations we can say that electronic songs have high valence. They are typically highly danceable, although this can vary. Electronic music generally features synthetic sounds rather than acoustic elements, evidenced by low acousticness. Many electronic tracks are instrumental or have minimal vocals, showing high instrumentalness. They are usually characterized by high energy and fast tempo. Electronic songs tend to have a broad appeal, reflected in their moderate popularity, but aren't necessarily mainstream. They also feature low speechiness, meaning there are fewer lyrics. These features collectively help the model accurately identify and classify electronic music.

References

[1] he_normal. https://medium.com/@sakeshpusuluri/activation-functions-and-weight-initialization-in~:text=In%20He%2D%20initialization%20also%2Cwe,in%20faster%20convergence%20to%20minima.

[2] Kan bug. https://github.com/KindXiaoming/pykan/issues/174.

[3] Kan example. https://github.com/KindXiaoming/pykan/blob/master/tutorials/Example_4_symbolic_regression.ipynb.

[4] Kan paper. https://arxiv.org/pdf/2404.19756.

[5] List of music award shows. https://totalmusicawards.com/uncategorized/list-of-music-award-shows/, 2023. Accessed: 2023-04-02.

[6] Understanding umap. https://pair-code.github.io/understanding-umap/, 2023.

[7] Spotify. Normalizzazione dell'intensità del suono. https://support.spotify.com/it/artists/article/loudness-normalization/, 2023. Accessed: 2024-04-01.