Università Di PISA

Department of Computer Science

Master's in data science & business informatics

DM-2 DATA MINING

Advance Topics and Applications

Submitted To:

Prof. Ricardo Guidotti

Dott. Andrea Fedele

Submitted By:

Muzammil Raza Soomro (665575)

Amir Hassan (683185)

Ritoo Talreja (667551)

Academic Year 2023/24

Module -1 Data Understanding and Preparation

1.1 Introduction

The Spotify datasets, both tabular and time series, are analysed in this report. Tracks and artists are the two main parts of the tabular dataset. While the artists dataset contains details on different artists, including popularity, followers, and genres with the shape of (30141,5), the tracks dataset comprises (109547, 34) Spotify songs, each with its properties. Before merging tabular datasets, we dropped missing rows as observed that artists have only two row s with missing values as we have shown in figure 1.1.

and tracks is clear.

	id	name	popularity	followers	genres
11872	NaN	NaN	NaN	NaN	NaN
21223	4oPYazJJ1o4rWBrTw9lm40	NaN	47.0	35655.0	[]

Figure 1.1. Artists Missing Values

1.2. Analysis of Datasets

1.2.1. Analyse of Genres

1. Top 10 Genres by Maximum Danceability

The figure 1.2 highlights the top 10 genres with the highest average danceability scores. Danceability measures how suitable a track is for dancing based on tempo, rhythm stability, and beat strength.

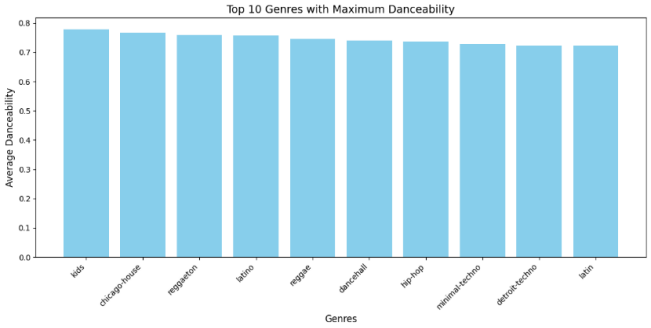


Figure 1.2. Top 10% Genres by Danceability

Key Observations:

Genres such as *Lofi*, *disco house*, and *reggaeton* scored consistently high, indicating their upbeat nature.

Electronic subgenres dominate this category, reflecting their inherent design for dance-oriented music.

2. Top 10 Genres by Average Popularity

The figure 1.3 showcases the most popular genres by their average popularity scores. Popularity is determined by factors such as streaming counts and audience engagement.

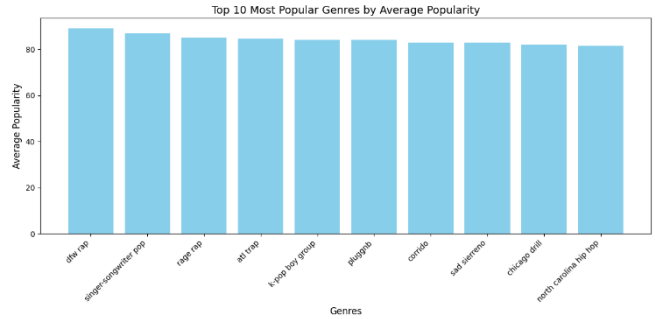


Figure 1.3. Top 10% Genres by Popularity

Key Observations:

Genres like *drill rap*, *independent pop*, and *alt trap* lead in popularity, indicating a preference for contemporary and experimental styles.

Diversity is evident, with genres ranging from pop to niche categories like *hyper pop*.

3. **Top 10 Genres by Total Followers**

The figure 1.4 ranks genres based on the total followers of associated artists.

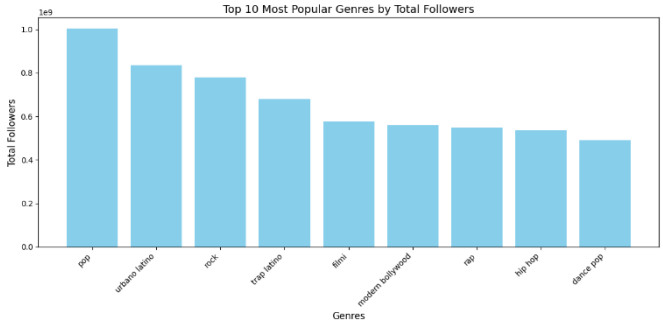


Figure 1.4. Top 10% Genres by total followers

Key Observations:

Pop dominates with the highest total followers, surpassing 1 billion, followed by *urbano latino* and *rock*.

Regional genres such as *trap latino* and *Bollywood* indicate a growing global audience.

4. Trend of Average Popularity Over Decades

The line graph analyzes the popularity of selected genres (*pop-film*, *k-pop*, *grunge*, *sertanejo*, and *chill*) over different decades.

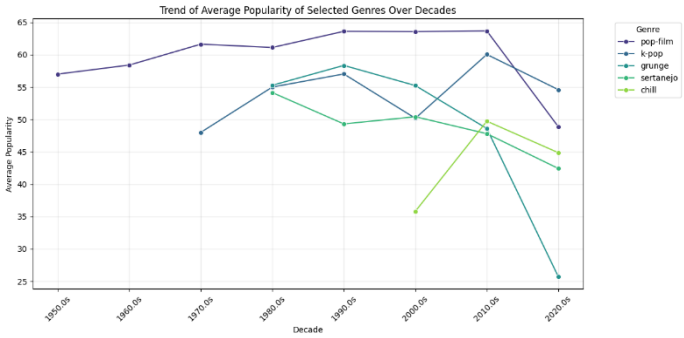


Figure 1.5.Trend of Average Popularity Over Decades *Genres*

Key Observations:

Pop-film maintained steady growth until the 2000s, showing a slight decline afterward.

Emerging genres like *sertanejo* and *chill* gained traction in the 2010s, demonstrating the dynamic nature of audience preferences.

Grunge, which peaked in the 1990s, saw a decline in subsequent decades, reflecting its niche appeal.

1.2.2. Analyse Track Durations

We converted track duration from milliseconds to minutes then visualizes the distribution of track durations with a histogram and density curve as we have shown in figure 1.3.

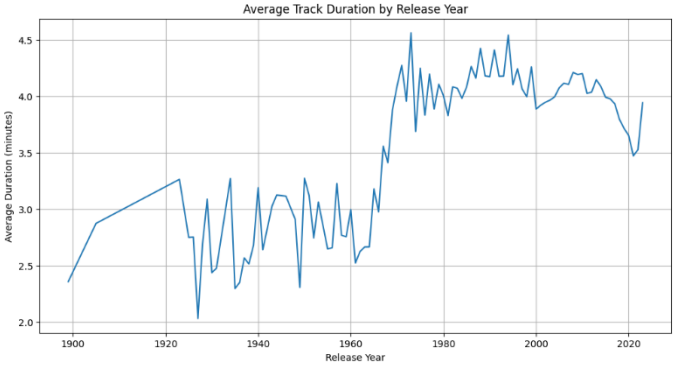


Figure 1.6. Track Duration in Minutes

1.2.3. Analyse Trends by Release Year

Our purpose was to extracts the release year from the album_release_date column and groups data by release year and calculates average track duration for each year. Visualizes the trend as a line chart in figure 1.4.

Key Findings

1. Artists Dataset:

Popularity and follower counts were analyzed,
many artists have low scores with a few outliers.
Missing data was addressed by removing incomplete rows.
Box plots identified extreme outliers in numerical fields.

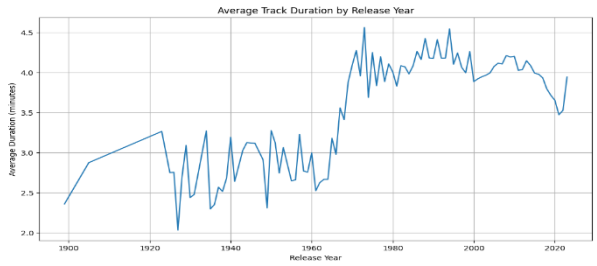


Figure 1.7. Average track duration by Release Year

2. Tracks Dataset:

Track durations were converted to minutes, revealing most tracks range from 2-5 minutes, with a few exceeding 15 minutes.
Popular genres like pop and electronic rank high in danceability and popularity.
Trends show track durations have slightly increased over the years.
Popularity trends highlight a shift in audience preferences over time.

1.3. Data Preparation and Merging Process

The process of merging music track data with artist information to create a comprehensive dataset. The objective is to facilitate analysis of music tracks in relation to their respective artists, focusing on attributes such as popularity and followers.

We split and exploded the artist column because it contained multiple artist names separated by semicolons. Before, a single row for a track with multiple artists. Now, multiple rows for same track, each containing a different artist.

The next step involves merging the exploded dataframe with the artist details by matched its names.

Further, calculated weighted average of popularity and followers for artists associated with music tracks, and the subsequent preparation of a *final dataset* that integrates these metrics with track information. This dataset allows for insights into how artist attributes influence track performance and can be used for further exploration of music industry trends.

1.4. Outliers Detection and Removal

To detect the outliers, we generated box plot and histogram to identify them. At this stage, we didn't remove any outliers.

1.5. Correlation

In this section, we delve into the correlation analysis of the dataset, which helps us understand the relationships between different features. The analysis of the correlation matrix between various track and artist features, providing insights into the relationships among these variables. The correlation coefficients range from -1 to 1, where values closer to 1 indicate a strong positive correlation, values closer to -1 indicate a strong negative correlation, and values around 0 suggest no correlation.

The correlation matrix provides valuable insights into the relationships between track and artist features. Key findings include the strong positive correlation between *energy* and *danceability*, as well as the moderate correlations between *track popularity* and *artist* metrics. These insights can inform further analysis and decision-making in music production and marketing strategies. Understanding these relationships is crucial for identifying trends and preferences within the music industry.

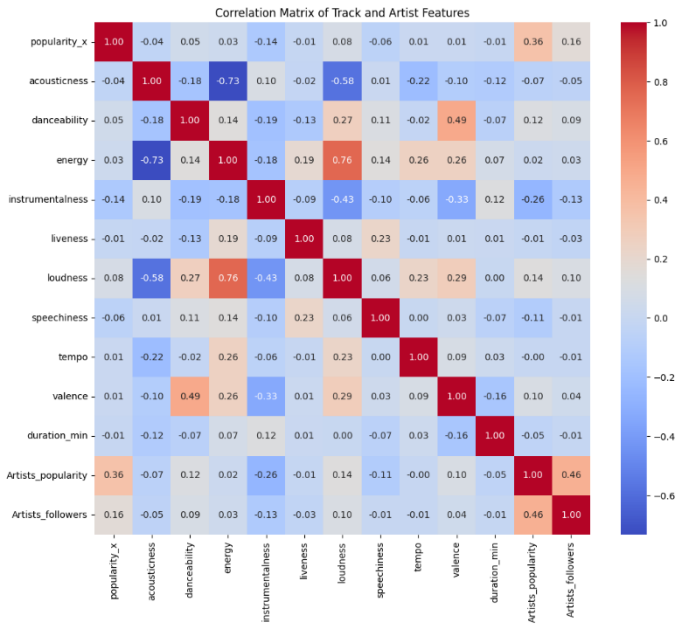


Figure 1.8. Correlation Matrix

1.6. Regression Analysis

The results of regression analyses conducted to explore the relationships between artist popularity, artist followers, and track popularity as we have shown in figure 1.6. The regression plots provide visual insights into these relationships, highlighting trends and potential correlations.

The regression analyses indicate that both artist popularity and the number of artist followers have a positive relationship with track popularity. These findings suggest that more popular artists and those with a larger follower base tend to have more popular tracks. These insights can be valuable for understanding the factors that contribute to track success and for developing strategies to enhance track popularity.

Further we analyze Genre-wise Track Popularity by generating box plot as it has been shown in figure 1.9.

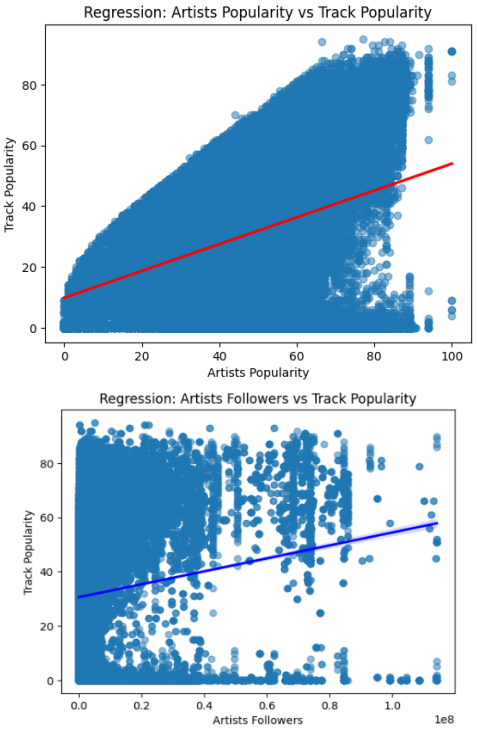


Figure 1.9. Regression Based Analysis

High Popular Genres:

Pop and Hip-Hop/Rap exhibit higher median popularity and include tracks with exceptional popularity.

Moderately Popular Genres:

Electronic/Dance, Jazz/Blues/Soul, and Rock/Alternative show balanced distributions with several high-performing tracks.

Lower Popularity Genres:

Children/Family and Classical/Instrumental tend to have lower medians, with occasional outliers.

Outliers:

Several genres (e.g., Pop and Jazz/Blues/Soul) contain outlier tracks with unusually high popularity.

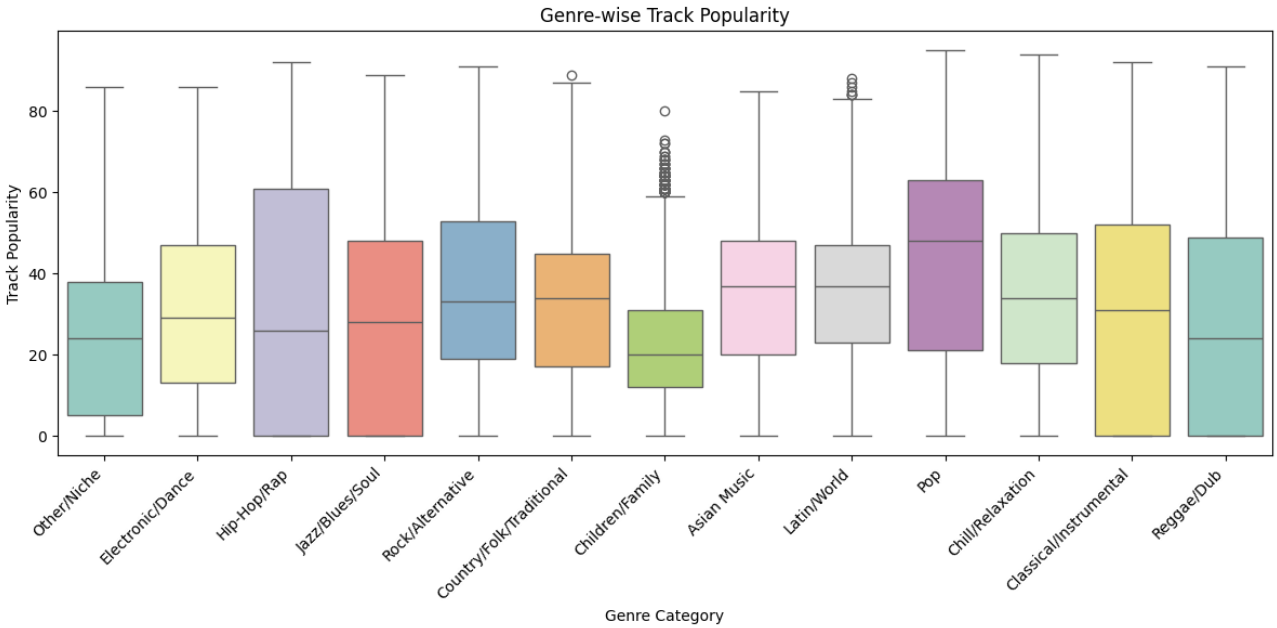


Figure 1.10. Genre-wise Track Popularity

1.7. Time Series Exploration and Analysis

A set of time series plots were generated to analyze the feature values over time for five randomly selected tracks. The tracks were visualized with their corresponding genres for a comprehensive understanding of their temporal patterns in figure 1.11.

1. Track-Specific Variations:

Each track exhibits a unique time series pattern, indicating the diversity in their audio features and dynamics.

For example, the track "Verdi: Aida, Act 2" (Opera genre) shows a clear pattern of peaks and valleys, likely representing dynamic musical expressions.

2. Genre Influence:

Tracks from different genres, such as **Progressive House** and **Synth-Pop**, display distinct trends. Progressive House tracks, for instance, exhibit repetitive and rhythmic peaks, while Synth-Pop tracks have smoother variations.

3. Feature Dynamics:

Some tracks (e.g., "If You Loop It, They Will Come") show consistent oscillations throughout, reflecting steady rhythmic patterns.

Others (e.g., "El Viagra") are more sporadic, with irregular fluctuations.

4. Applications:

These time series plots can help identify temporal trends, repetitive patterns, and peak occurrences in tracks across different genres, aiding in genre classification and feature analysis.

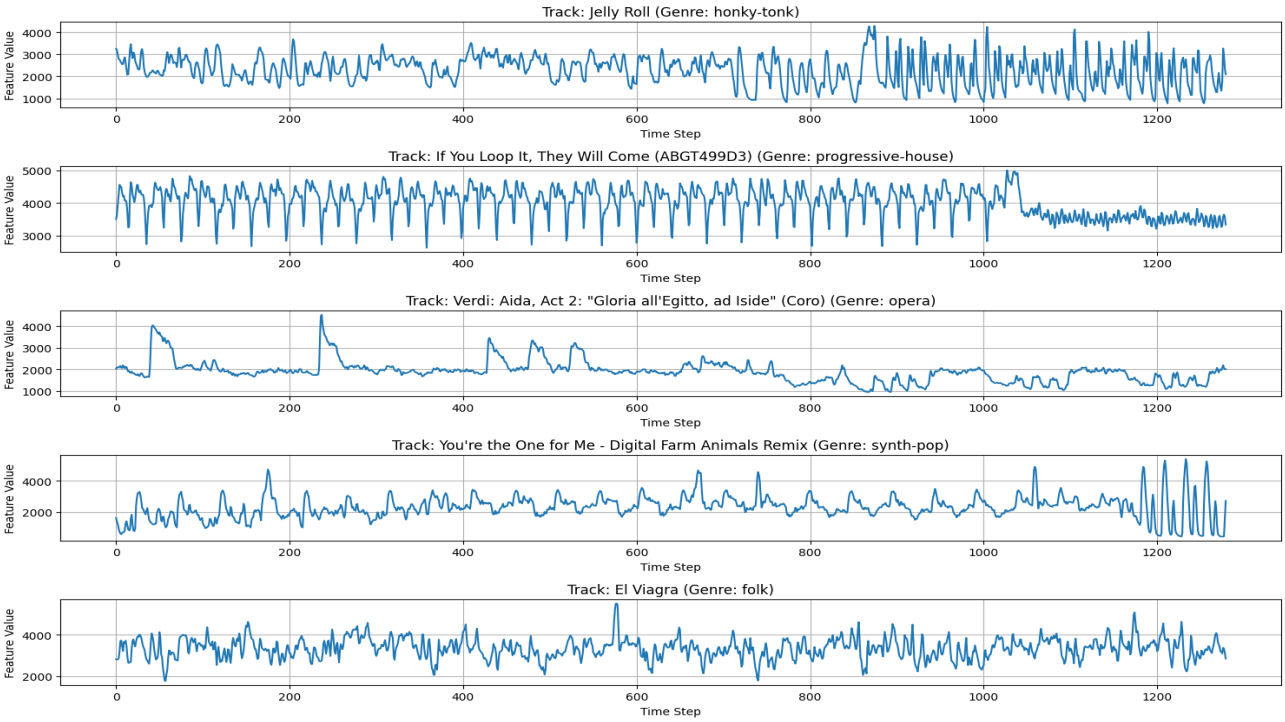


Figure 1.11. Time Series Visualization

1.8. Time Series Approximation and Clustering Using PAA and K-Means

We process and results of clustering time series data using Piecewise Aggregate Approximation (PAA) with K-Means. The aim is to transform high-dimensional time series data into lower-dimensional representations while preserving key features, enabling effective clustering and visualization.

Cluster Visualization

The PCA scatter plot reveals distinct groupings, with five clusters separated along the first and second principal components as we have shown in figure 1.12 to observe the effectiveness.

Clusters show overlapping in some regions, which correlates with the moderate silhouette score.

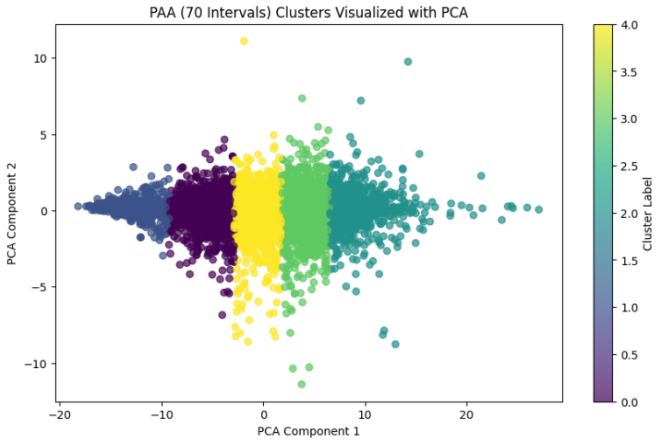


Figure 1.12. PAA Visualization

PAA Transformation

The PAA method effectively reduced the data complexity, enabling clustering without significant information loss.

We used 150, 100, 70 intervals with 5 clusters to captured the core trends of the original time series. The best result that we got at 70.

The primary metric for comparison is the silhouette score, which measures the quality of clustering.

Number of Intervals	Silhouette Score	Observations
70	**0.2235**	Best clustering performance, good balance between dimensionality reduction and feature preservation.
100	**0.2051**	Moderate performance, likely due to increased dimensionality affecting clustering separability.
150	**0.1869**	Lowest silhouette score, indicating reduced clustering quality as dimensionality increases further.

Table 1.1. Summary Result

Final analysis and visualization of a selected time series using scaled features and its corresponding approximation using Piecewise Aggregate Approximation (PAA) as shown in figure 1.13.

It demonstrates the capability of PAA to simplify high-dimensional time series data while retaining its core characteristics. This transformation is particularly useful for downstream tasks like clustering, feature extraction for machine learning models and noise reduction for enhance interpretability.

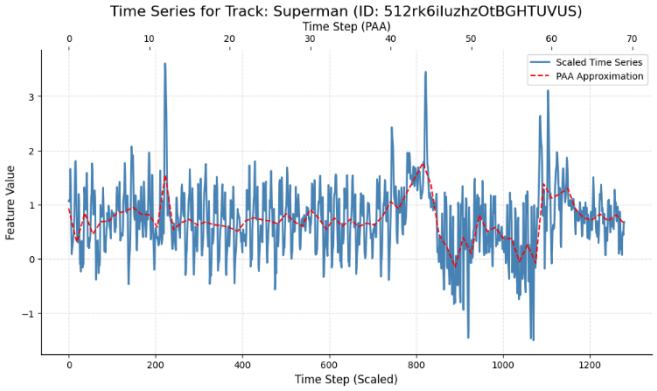


Figure 1.13. Time Series for Track: Superman

Module 2 Time series

2.1 Time Series Data Preparation

2.2 Clustering Analysis

In this module, we applied multiple clustering algorithms to the time series data and analyzed the resulting clusters. The goal was to explore and compare the effectiveness of different clustering methods and visualize the clusters using dimensionality reduction techniques. We employed tree clustering algorithms: K-means, DBSCAN, and hierarchical clustering, similar to the approaches used in DM1.

K-means Clustering

In K-means clustering, initially applying it to the raw time series data. To determine the optimal number of clusters (k), we utilized the Elbow Method and Silhouette Score, examining a range of k values from 2 to 10 as shown in figure 2.1. The Elbow Method helps identify the point where adding more clusters no longer significantly reduces the SS, while the Silhouette Score evaluates the clustering quality by measuring how similar an object is to its own cluster compared to other clusters. These metrics suggested the optimal at k=3 with average silhouette score is: *0.31489*. For both the elbow method and clustering, we used Euclidean distance as the metric due to its simplicity and effectiveness in measuring the straight-line distance between points in the high-dimensional space of the time series data. We also try to use DTW on the PAA dataset. Thus, we can see how the centroid of the time series changes depending on whether we use PAA and DTW in the figures below:

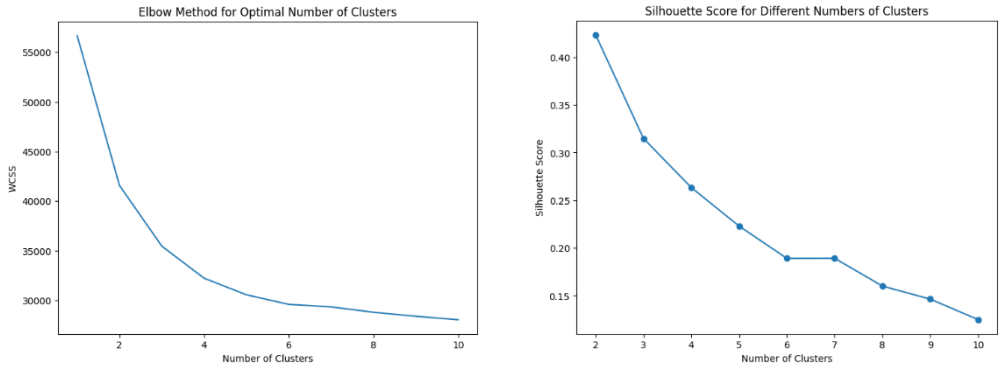


Figure 2.1. Evaluation of Optimal Clusters Using Silhouette Scores and Elbow Method

We compares two different methods for projecting high-dimensional time series data into a 2D space after clustering using the K-Means algorithm. We used two methods Principal *Component Analysis (PCA)* and *t-Distributed Stochastic Neighbour Embedding (t-SNE)* for dimensionality reduction, with clusters coloured to show grouping as shown below figure.

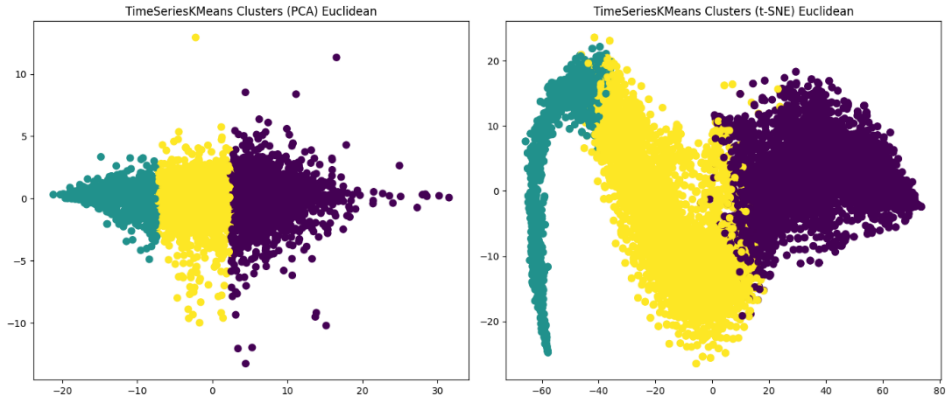


Figure 2.2. Visualization of K-Means Clusters Using PCA and t-SNE Projections

Both projections highlight three clusters, but the t-SNE plot shows a more distinct separation, with reduced overlap between clusters. PCA retains the overall global variance but might underrepresent the fine-grained structure seen in t-SNE. PCA emphasizes large-scale patterns and might "flatten" relationships between data points. t-SNE highlights local data relationships, which makes it ideal for understanding complex, non-linear patterns.

The DTW distance metric on PAA data provides smoother and potentially more meaningful centroids by accommodating time shifts and variations with the average silhouette score is: *0.3087*, better than K-Means. Comparing the centroids from raw data and PAA-reduced data with different distance metrics shows how preprocessing and the choice of distance metric can influence the clustering outcome.

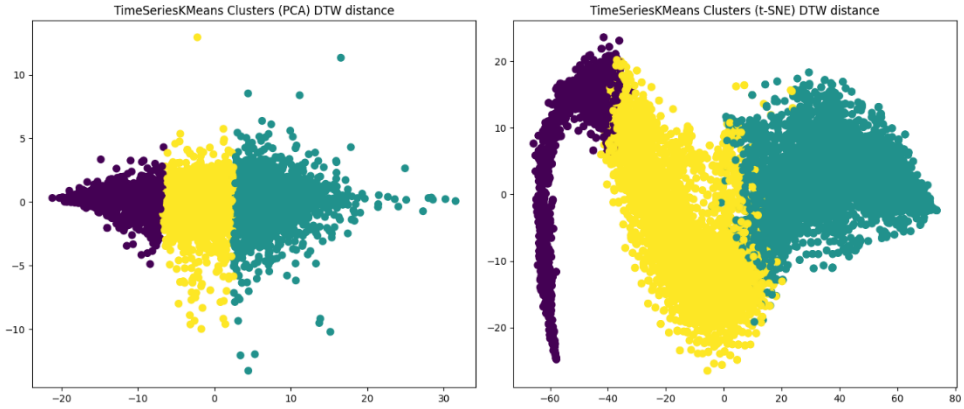


Figure 2.3. Visualization of DTW Clusters Using PCA and t-SNE Projections

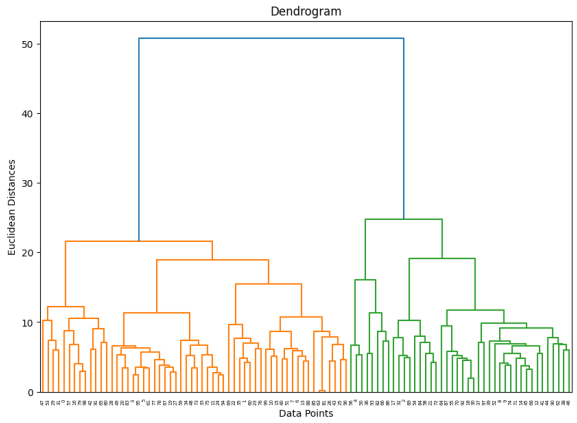
The t-SNE projection (right) provides a clearer and more distinct separation of clusters compared to PCA (left). DTW enhances the clustering process by aligning time series, ensuring meaningful groupings despite temporal misalignments.

DBSCAN Clustering

As in DM 1, we used density-based clustering, where first, we determined the best value of eps using the elbow method. For the min_samples, we set a range from 2 to 11 , which resulted to choose the best min_sample as 2 otherwise we could run into a problem of 1 cluster and a lot of noise points. So, the parameters were eps: 5.71, min_samples: 2, clusters: 4, noise points: 205, which we plotted to visualize the clusters. The clusters were not well-separated, indicating poor clustering performance. Consequently, we explored other algorithms.

Hierarchical Clustering

We applied this technique to the time series dataset to explore how it groups the data without pre-specifying the number of clusters. We used dataset after approximation (PAA) and Euclidean as a distance metric for efficiency and to visualize the hierarchical clustering results, we used dendrograms, which display the hierarchical relationships between clusters, on the figure 2.4 and 2.5. For the distance metric various linkage methods (ward, complete, average, and single) were tested. The approach involves building a dendrogram to determine the optimal number of clusters and visualizing the clustering results using PCA (Principal Component Analysis).



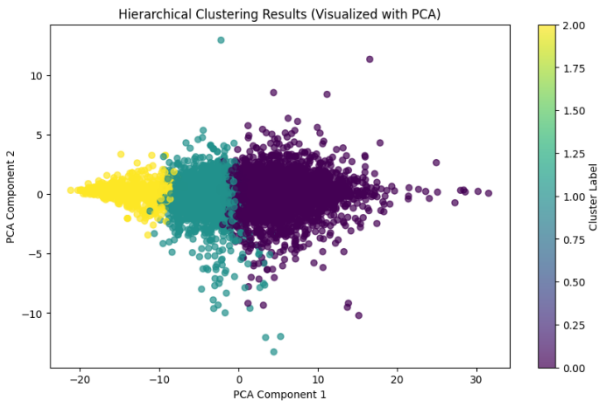


Figure 2.4: Hierarchical Clustering Dendrogram (Ward Linkage)

Figure 2.5. Hierarchical Clustering Visualization with PCA

Hierarchical clustering, visualized with a dendrogram and PCA, effectively grouped the time series data into 3 meaningful clusters. The dendrogram validated the optimal number of clusters, while PCA provided an interpretable view of the cluster structure. This approach is particularly suitable for identifying natural groupings in high-dimensional data like time series.

2.3 Motifs and anomalies

The purpose of this analysis is to detect motifs (repeating patterns) and anomalies (discords) in a time series dataset using Matrix Profiles. Motifs help identify recurring behaviors, while anomalies signal unusual or rare events.

We focus on motif discovery within the time series dataset. Identifying motifs helps in various applications, including pattern recognition, anomaly detection, and predictive modeling. In our case we used it to identify patterns in the specific time series. We began by choosing our first time series for motifs. This initial step was aimed at gaining a preliminary understanding of the patterns and repetitions present within the data.We discovered motifs on original and PAA time series as shown in figure 2.7. Graph of time series with motifs highlighted in different colors for clear distinction. Each motif's repeated occurrences are shown with thick colored lines corresponding to the motif and we set window size 128 on original TS and 12 for PAA as shown in figure 2.8. Further we separated motif as shown below.

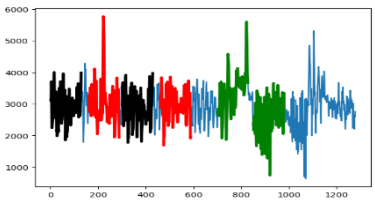
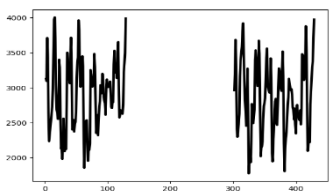


Figure 2.7: Motif on Original TS



Single Motif

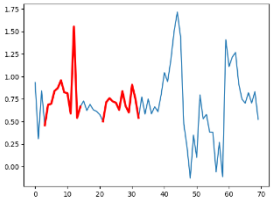


Figure 2.8: Motif on PAA TS

We discovered discords (anomalies), which identifies subsequences with the least similarity to others in the time series. An exclusion zone was applied to avoid detecting anomalies within overlapping windows. Top 5 anomalies were detected, with subsequence highlighted on the plot in figure 2.8. The anomalies are often located in regions of sharp spikes or drops, visually distinguishing them from the motifs. Anomalies suggest significant deviations from normal behavior, potentially indicating critical events.

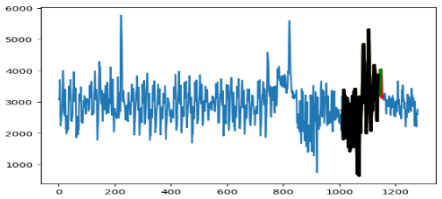


Figure 2.9. Anomalies Detection

2.4 Classification

In this section, we discuss the classification of time series data using various techniques and distance metrics. We used KNN classifier with Euclidean, Manhattan and DTW distances.

First, we attempt to classify the time series data without approximation and without additional features, using Euclidean and Manhattan distances were used. DTW was not applied due to the large dataset size. The result of the accuracy was extremely poor, achieving only 0.12 with Euclidean and 0.13 with Manhattan for 20 genres.

2.5 Shapelets

Method

In this section we applied Shapelet based classification on our genre classification tasks. First, we discovered shapelets considering genreas a target and then we applied the Shapelet Model on the shapelets discovered through the previous steps which produced 100 shapelets of size 128. The Shapelet model gave 0.11 of accuracy. This suggests that the Decision Tree classifier is not performing well on the given dataset. Since there are 20 classes, a random guess would result in an accuracy of about 5%, so 11% is slightly better than random and f-1 score with range from 0.05 to 0.22.

We visualizes and explains three scenarios of shapelet analysis applied to time series data, illustrating how shapelets align with the original data to extract meaningful patterns.

1. Randomly Selected Shapelet

The randomly selected shapelet captures some local patterns but does not necessarily align with a meaningful feature in the time series as shown in figure 2.6.

2. Aligned Extracted Shapelet

The shapelet (orange line) overlays a segment of the time series (blue line), capturing a well-defined pattern. The alignment suggests that this shapelet represents a recurring feature in the data, making it a potential candidate for classification tasks.

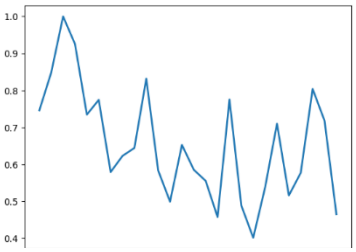


Figure 2.6 - Randomly Selected Shapelet (Shows a shapelet that does not align well with meaningful patterns in the time series)

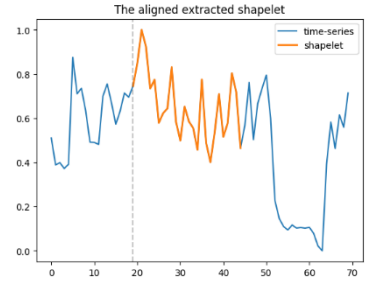


Figure - Aligned Extracted Shapelet (Orange shapelet aligns with blue time-series, showing its effectiveness in pattern recognition)

3. Normalized Shapelet Analysis

Three different representations of the shapelet

Normalized visualization includes, original time series (blue line), extracted shapelet (orange line) and z-normalized version of the shapelet (green line). The z-normalized shapelet adjusts for differences in scale, enhancing its ability to match similar patterns across time series with varying amplitudes. The alignment demonstrates the robustness of the shapelet in capturing the same feature under different conditions.

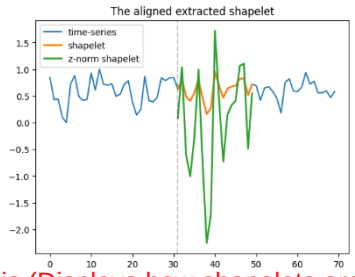


Figure - Normalized Shapelet Analysis (Displays how shapelets are normalized to match patterns in varying amplitudes)

2.6 Classification using RNN and CNN

In this section we compares the performance of different models (RNN, LSTM, and CNN) applied to both original time series data (unscaled) and PAA-approximated data. The aim is to evaluate the effectiveness of feature approximation and different architectures for classification tasks as we shown their results in below table 2.1.

Model	Data Type	Test Loss	Test Accuracy
RNN (LSTM)	Original Time Series	3.0455	0.0510 (5.1%)
RNN (LSTM)	PAA Approximated Data	2.6850	0.1444 (14.4%)
CNN	PAA Approximated Data	2.4503	0.2412 (24.1%)

Table 2.1. Model Performance on Original and PAA Approximated Time Series Data

The **CNN** on PAA Approximated Data achieved the best performance with a test accuracy of **24.1%** and the lowest test loss (**2.4503**) as we can see from the table.

Module 3 Imbalanced Learning

Target Variable Engineering

Exploratory data Analysis (EDA) revealed that there was no such variable present in the dataset which can be used to imbalance learning. Therefore, a new variable ***is_popular*** was engineered based on the existing information contained in the dataset. This new variable is a binary class variable which was created using track popularity variable. A threshold of 70 was set on track popularity. Tracks which had popularity higher or equal than 70 were considered as popular and others as not popular. This exercise resulted in a variable that was imbalanced and perfect for the requirements of this module.

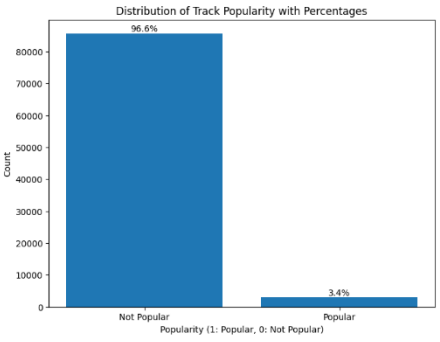


Figure 3.1: Distribution of Track Popularity

Feature Importance and Selection

The objective of this analysis was to determine the most relevant features influencing track popularity using a **Random Forest Classifier**. Feature importance was calculated and visualized to guide the selection of features for predictive modeling and further analysis as shown in figure.

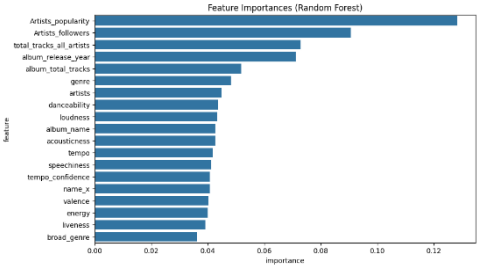


Figure 3.2: Feature Importance

3.1.1. KNN on Unbalanced Data

Before balancing data, we run models on imbalanced data to see differences. While the KNN classifier achieved high accuracy, its low precision, recall, and F1-score for the minority class reveal limitations in handling the class imbalance and model was initialized with n_neighbors=5. The high accuracy (96.6%) is misleading, as it primarily reflects the model's success in predicting the majority class (not popular) an shown in confusion matrix.

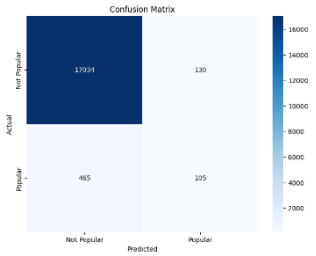


Figure: Confusion Matrix

Metric	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	0.97	0.45	0.71	0.96
Recall	0.99	0.18	0.59	0.97
F1-Score	0.98	0.26	0.62	0.96

Table: KNN Classification Table

3.1.2 Decision Tree on Unbalanced Data

The Decision Tree classifier achieves high accuracy (95.6%) due to the class imbalance but struggles with minority class (Popular) detection as shown in confusion matrix and table.

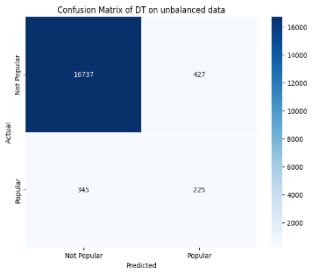


Figure: Confusion Matix

Metric	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	0.98	0.35	0.66	0.96
Recall	0.98	0.39	0.68	0.96
F1-Score	0.98	0.37	0.67	0.96

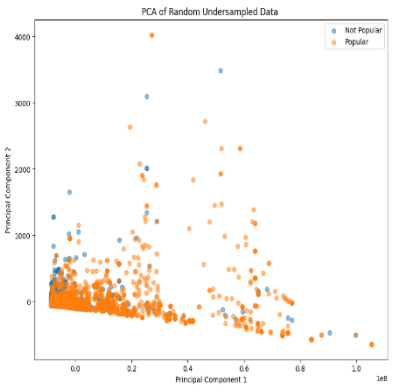
Table: DT Classification Table

3.2. Under Sampling

3.2.1. Random Undersampling

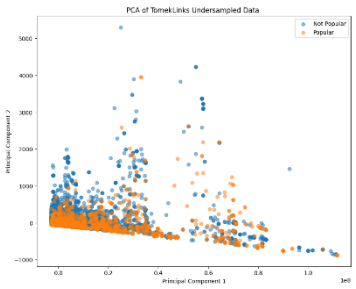
While using Decision Tree (DT) and K-Nearest Neighbors (KNN) classifiers after applying *Random Undersampling* to address the class imbalance. The evaluation metrics highlight the performance of the models on a balanced dataset below. The Decision Tree classifier demonstrated better performance in most metrics after undersampling, making it a suitable choice for the given task. However, both models show limitations in achieving high precision for the minority class.

Metric	Model	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	**DT**	0.99	0.15	0.57	0.97
	KNN	0.99	0.10	0.54	0.96
Recall	**DT**	0.84	0.85	0.85	0.84
	KNN	0.75	0.79	0.77	0.76
F1-Score	**DT**	0.91	0.26	0.58	0.89
	KNN	0.86	0.17	0.51	0.83



3.2.2. TOMEK LINK

Tomek Link sampling technique was applied to the dataset to reduce class imbalance by removing borderline samples that may cause misclassification. The following evaluation metrics were obtained using a DT classifier on the processed dataset and shown in table and PCA graph.



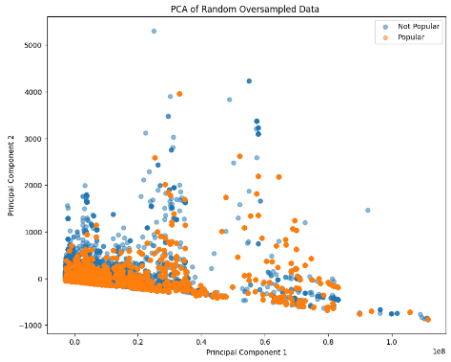
Metric	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	0.98	0.52	0.75	0.96
Recall	0.99	0.32	0.65	0.97
F1-Score	0.98	0.39	0.69	0.96

3.3. Over Sampling

3.3.1. Random Oversampling

Random Oversampling was applied to balance the dataset by duplicating instances of the minority class (Class 1: Popular). The below metrics evaluate the DT classifier's performance after oversampling.

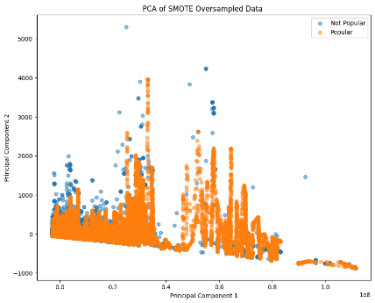
Metric	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	0.98	0.37	0.67	0.96
Recall	0.98	0.34	0.66	0.96
F1-Score	0.98	0.36	0.67	0.96



3.3.2. SMOTE

SMOTE (Synthetic Minority Oversampling Technique) was applied to generate synthetic samples for the minority class (Class 1: Popular), balancing the dataset and potentially improving the classifier's performance. The evaluation metrics for the Decision Tree classifier after applying SMOTE are presented below.

Metric	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	0.98	0.28	0.63	0.96
Recall	0.95	0.54	0.75	0.94
F1-Score	0.97	0.37	0.67	0.95



3.4. Weighted DT

The Class Weighted Decision Tree improves the recall for Class 1 (Popular) significantly compared to standard methods without weights. This improvement comes with a trade-off in precision, resulting in a moderate F1-Score as shown in table. The weighted approach balances the importance of the minority class, leading to better sensitivity for detecting popular tracks while maintaining high accuracy for the majority class.

Metric	Class 0 (Not Popular)	Class 1 (Popular)	Macro Average	Weighted Average
Precision	0.98	0.30	0.64	0.96
Recall	0.96	0.50	0.73	0.95
F1-Score	0.97	0.38	0.68	0.95

Overall Metrics Results

The ***DT (Tomek Link)*** model provides the most balanced performance across all metrics, making it the best choice overall. It achieves high precision, a competitive recall, and the best F1-score, ensuring effective identification of popular tracks while minimizing false positives.

Model	Accuracy	F1 Score	Precision
KNN (Undersampling)	0.7555	0.1725	0.0968
DT (Undersampling)	0.8423	0.2571	0.1515
DT (Tomek Link)	0.9685	0.3917	0.5158
DT (Oversampling)	0.9599	0.3551	0.367
DT (SMOTE)	0.9408	0.3692	0.2809
Class Weighted DT	0.9472	0.3777	0.3041

Table: Metrics Comparison

3.3 Outliers Detection

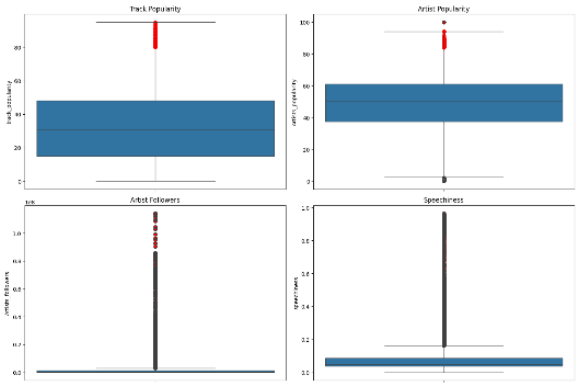


Figure 3.1: Top 1% Outliers Detection

Outliers' detection means finding things that are unusual because they are very different from what usually happens. It is important to do this because it can help us analyze data better. Our goal was to find the top 1% of outliers in the data. We select different features to identify outliers as shown figure. The boxplots provide a clear summary of the key variables, highlighting differences in central tendency, variability, and the presence of outliers. The data suggests a high skew in *artist followers* and *speechiness*, while *popularity* metrics are more normally distributed. We used different methods to do this, which is Histogram-based Outlier Score (HBOS), ABOD algorithm, DBSCAN Isolation Forest algorithm, LOF algorithm.

3.3.1 Histogram-based Outlier Score (HBOS):

First, we used the Histogram-based Outlier Score (HBOS) outlier detection algorithm with default parameters to identify outliers in our dataset. According to the output of the algorithm, there were 896 outliers detected from 89565 datapoints and 88669 inliers. The x-axis represents the first principal component, which accounts for the largest possible variance in the dataset. The y-axis represents the second principal component, which is orthogonal to PC1 and accounts for the second-largest variance in the dataset. They provide a compressed yet meaningful view of the data's variability, helping us to visualize and interpret clusters, trends, and outliers.

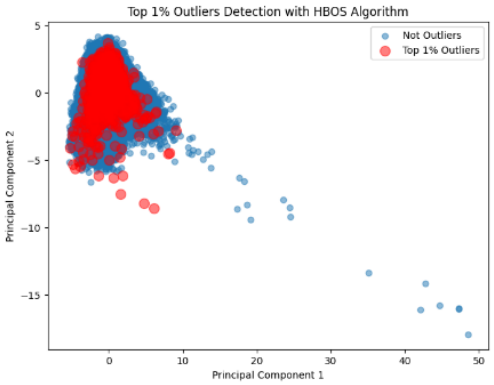


Figure 3.2 HBOS

3.3.2 Local Outlier Factor (LOF):

Next, we applied the Local Outlier Factor (LOF) algorithm with default parameters to detect outliers in our dataset. The algorithm identified 896 outliers out of 89,565 data points, leaving 88,669 as inliers. To visualize the results, we utilized principal component analysis (PCA), where the x-axis represents the first principal component (PC1), capturing the highest variance in the data, while the y-axis represents the second principal component (PC2), which is orthogonal to PC1 and accounts for the second-largest variance. This transformation provides a compact yet meaningful representation of the dataset, allowing for better identification of clusters, trends, and anomalies.

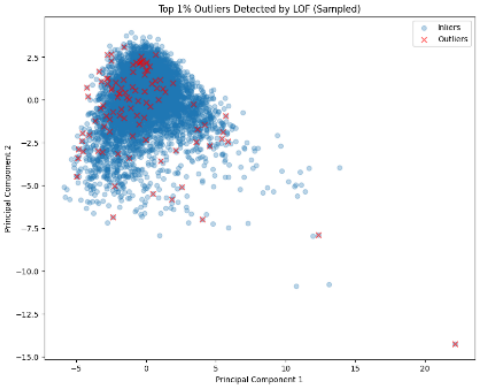


Figure 3.2 LOF

3.3.3 Isolation Forest

We applied the Isolation Forest algorithm with default parameters to detect outliers in our dataset. The algorithm identified 896 outliers out of 89,565 data points, leaving 88,669 as inliers. To visualize the results, we used principal component analysis (PCA), where the x-axis represents the first principal component (PC1), capturing the maximum variance in the data, while the y-axis represents the second principal component (PC2), which is orthogonal to PC1 and accounts for the second-largest variance. This transformation provides a compact yet insightful representation of the dataset, making it easier to identify clusters, trends, and anomalies.

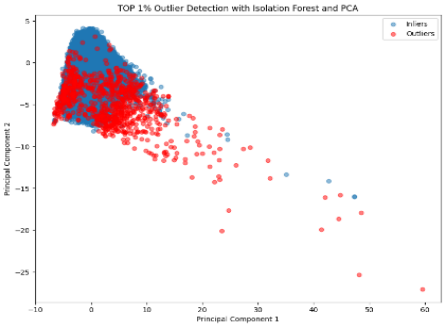


Figure 3.2: Isolation Forest

3.3.4 Angle-based Outlier Detection (ABOD)

Fourth, we used ABOD. According to the output of the algorithm, we didn't find any outliers as shown in figure 3.3. The x-axis represents the first principal component, which accounts for the largest possible variance in the dataset. The y-axis represents the second principal component, which is orthogonal to PC1 and accounts for the second-largest variance in the dataset.

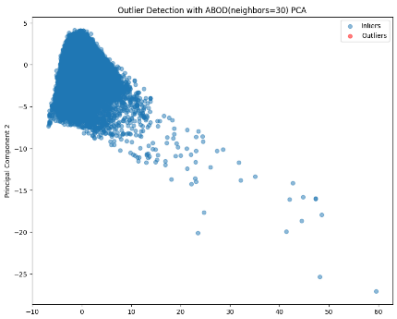


Figure 3.3: ABOD Results

3.3.5 Lightweight On-line Detector of Anomalies (LODA)

Fifth, we used LODA is an efficient and scalable anomaly detection algorithm that combines multiple random projection histograms. It is lightweight and can efficiently handle large datasets, making it suitable for real-time applications. Points with the highest anomaly scores are flagged as outliers. A total of *896 outliers* were detected, making up approximately 1% of the dataset, remaining *88,669 inliers* represent the majority of the data.

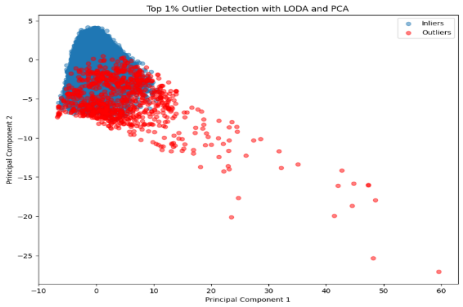


Figure 3.5: LODA Results

3.4 Removing Outliers:

We tried several approaches to manage outliers. The first was to eliminate the top 1% of outliers that were common among at least 3 of the 4 previously proposed methods but we consider HBOS detected outliers, this made their identification more reliable. After this, at this point we created a copy of our original dataset and dropped the outliers as in figure 3.5 before and after removal of outlier's data.

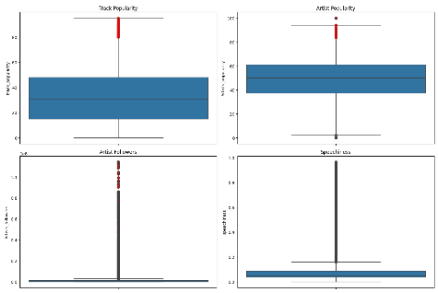


Figure 3.5: Before and After Outlier Removal

Module 4 Advanced Classification

Data Preparation

The goal of data preparation is to preprocess the dataset for building machine learning models for both *multi-class* and *binary classification* tasks. This includes handling categorical variables, scaling features, splitting the dataset, and preparing target variables. To do classification we set two target variables, binary variable *is_popular* that we created in imbalanced learning. Second while doing data preparation we merge similar genres into broad 14 genres in order to remove complexity and have better results. This was done by adding a new column in the dataset named as a *broad_genre*. Before building our model, we prepared the dataset by transforming certain attributes and removing unnecessary features. The feature selection was based on previous knowledge obtained in previous modules. The label encoding was also done on the multi-class target attribute.

4.1. Logistic Regression – Multi Class Classification

The goal of this analysis is to evaluate the performance of a Logistic Regression base model for multi-class classification on a dataset with *14 classes*. Key metrics such as accuracy, precision, recall, and F1-score are used to assess the model's effectiveness. These averages indicate that the model struggles to perform well across all classes, with a notable imbalance.

The base Logistic Regression model exhibits suboptimal performance, with low accuracy of **29.59%** and inconsistent metrics across classes. While it performs moderately well for certain classes (e.g., Class 5), it struggles significantly with others (e.g., Class 6) as shown in table.

	precision	recall	f1-score	support
0	0.19	0.08	0.11	2386
1	0.25	0.04	0.06	1133
2	0.36	0.43	0.39	1484
3	0.27	0.22	0.24	1452
4	0.23	0.39	0.29	2455
5	0.33	0.04	0.07	363
6	0.30	0.49	0.37	3694
7	0.23	0.02	0.04	604
8	0.18	0.02	0.04	1229
9	0.43	0.05	0.08	1608
10	0.31	0.24	0.27	3313
11	0.37	0.20	0.26	1953
12	0.55	0.14	0.22	652
13	0.31	0.59	0.41	4275
accuracy			0.30	26601
macro avg	0.31	0.21	0.20	26601
weighted avg	0.30	0.30	0.26	26601

Figure 4.1: Linear Regression Model Results

4.1.1. Logistic Regression After Hyperparameter Tuning: Multi-Class Classification

	precision	recall	f1-score	support
0	0.19	0.08	0.12	2386
1	0.23	0.06	0.09	1133
2	0.36	0.43	0.39	1484
3	0.27	0.22	0.24	1452
4	0.24	0.39	0.30	2455
5	0.31	0.04	0.07	363
6	0.30	0.49	0.37	3694
7	0.24	0.02	0.04	604
8	0.17	0.02	0.03	1229
9	0.43	0.05	0.08	1608
10	0.31	0.24	0.27	3313
11	0.36	0.20	0.25	1953
12	0.55	0.14	0.22	652
13	0.31	0.59	0.41	4275
accuracy			0.30	26601
macro avg	0.30	0.21	0.21	26601

Figure 4.1.1: Linear Regression Model After Parameter Tunign

Our aim of this analysis was to optimize the Logistic Regression model for multi-class classification by tuning its hyperparameters. The hyperparameter optimization process using Grid Search identified the following optimal parameter settings for the Logistic Regression model: C=10, max_iter=1000, penalty='l1', and solver='saga'. These values suggest a model with moderate regularization strength (C=10), employing L1 regularization (penalty='l1') for feature selection and potentially enhanced interpretability. The 'saga' solver was found to be most effective for this dataset and regularization type, while max_iter=1000 ensured adequate optimization convergence. This configuration yielded the highest cross-validation performance, indicating a balance between model complexity and generalization ability. The tuned model was evaluated for its performance using various metrics such as accuracy, precision, recall, and F1-score as shown in table. The accuracy remained low at **29.63%**, indicating that the model struggles with classifying instances across the 14 categories. While hyperparameter tuning slightly improved the Logistic Regression model, the overall performance remained suboptimal due to the complexity of the multi-class task and class imbalance.

4.1.2. Logistic Regression: Binary Classification Without Imbalance Treatment

	precision	recall	f1-score	support
0	0.97	1.00	0.98	25705
1	0.41	0.07	0.12	896
accuracy			0.97	26601
macro avg	0.69	0.53	0.55	26601
weighted avg	0.95	0.97	0.95	26601

Table: LR Metric

Before making it balanced, we applied Logistic Regression base model was evaluated on a binary classification task to predict whether tracks are **popular (1)** or **not popular (0)**. No imbalance treatment was applied, leading to a significant dominance of the majority class. Results could be seen in table. The best parameter from the previous fine tuning were used after trying the default parameters.

4.1.3. Logistic Regression Model After Imbalance Treatment Using SMOTE

	precision	recall	f1-score	support
0	1.00	0.84	0.91	25705
1	0.16	0.88	0.27	896
accuracy			0.84	26601
macro avg	0.58	0.86	0.59	26601
weighted avg	0.97	0.84	0.89	26601

Table: Evaluation Matrix

The Logistic Regression model was retrained after addressing the class imbalance using SMOTE (Synthetic Minority Oversampling Technique) to generate synthetic samples for the minority class (Class 1: Popular). This approach aimed to improve the model's ability to classify the minority class. With the accuracy of **84.05%** reflects the model's overall correctness across both classes and showed in table as well. After applying SMOTE, the model achieved a substantial improvement in **recall (88%)** for the minority class (Class 1: Popular), highlighting its ability to identify most popular tracks. However, the low **precision (16%)** for Class 1 indicates room for improvement to reduce false positives.

4.2. SVM - Multi Class Classification

	precision	recall	f1-score	support
0	0.19	0.08	0.12	2386
1	0.23	0.06	0.09	1133
2	0.36	0.43	0.39	1484
3	0.27	0.22	0.24	1452
4	0.24	0.39	0.30	2455
5	0.31	0.04	0.07	363
6	0.30	0.49	0.37	3694
7	0.24	0.02	0.04	604
8	0.17	0.02	0.03	1229
9	0.43	0.05	0.08	1608
10	0.31	0.24	0.27	3313
11	0.36	0.20	0.25	1953
12	0.55	0.14	0.22	652
13	0.31	0.59	0.41	4275
accuracy			0.30	26601
macro avg	0.30	0.21	0.21	26601
weighted avg	0.30	0.30	0.26	26601

Table: Evaluation Matrix

The LinearSVC model was trained using the following parameters: C=1.0 and random_state=42. The regularization parameter C was set to 1.0, indicating a moderate level of regularization to balance model complexity and generalization performance. This choice aims to prevent overfitting while maintaining adequate classification accuracy on the training data. Additionally, the random_state parameter was fixed at 42 to ensure reproducibility of the experimental results. Evaluating the

performance of a Support Vector Machine (SVM) base model for multi-class classification without hyperparameter tuning indicates that the model correctly classified approximately 30% of the instances, performing similarly to Logistic Regression. The model achieved **29.62% accuracy**, which is similar to the Logistic Regression base model, suggesting limited improvements without tuning.

4.2.1 SVM - Binary Classification

We evaluated the performance of a *Support Vector Classifier (SVC)* with the Radial Basis Function (RBF) kernel (gamma=auto) for binary-class classification. Indicates the model correctly classified approximately 87% of the instances. The SVC model with gamma=auto and kernel=rbf achieves high accuracy **86.67%** and recall for both classes, especially for the minority class (Class 1). However, the low precision for Class 1 suggests a high false positive rate as shown in table.

	precision	recall	f1-score	support
0	0.99	0.87	0.93	25705
1	0.19	0.87	0.31	896
accuracy			0.87	26601
macro avg	0.59	0.87	0.62	26601
weighted avg	0.97	0.87	0.91	26601

Table: Evaluation Matrix

4.3 Linear SVC

We evaluated the performance of a *Linear Support Vector Classifier* (Linear SVC) for binary classification. The model correctly classified approximately 84% of the instances. Linear SVC achieves high accuracy (**83.55%**) and excellent recall for the minority class (Class 1). However, its low precision for Class 1 highlights a significant challenge with false positives as shown in matrix.

	precision	recall	f1-score	support
0	1.00	0.83	0.91	25705
1	0.16	0.89	0.27	896
accuracy			0.84	26601
macro avg	0.58	0.86	0.59	26601
weighted avg	0.97	0.84	0.89	26601

Table: Evaluation Matrix

4.2. Random Forest - Multi Class Classification

While evaluating the performance of a *Random Forest Classifier* for multi-class classification. We set 100 trees, the Gini impurity criterion, and default hyperparameters to build the ensemble. Indicates the model correctly classified approximately 30% of the instances. Random Forest Classifier achieves moderate performance in multi-class classification, with an accuracy of **29.62%** and highly variable metrics across classes. While it handles majority classes reasonably well, it struggles with minority classes due to data imbalance as shown in table. Random Forest inherently provides feature importance scores, which can be analysed to identify the most influential features driving the classification and observed feature importance as shown in figure 3.5.

	precision	recall	f1-score	support
0	0.19	0.08	0.12	2386
1	0.23	0.06	0.09	1133
2	0.36	0.43	0.39	1484
3	0.27	0.22	0.24	1452
4	0.24	0.39	0.30	2455
5	0.31	0.04	0.07	363
6	0.30	0.49	0.37	3694
7	0.24	0.02	0.04	604
8	0.17	0.02	0.03	1229
9	0.43	0.05	0.08	1608
10	0.31	0.24	0.27	3313
11	0.36	0.20	0.25	1953
12	0.55	0.14	0.22	652
13	0.31	0.59	0.41	4275
accuracy			0.30	26601
macro avg	0.30	0.21	0.21	26601
weighted avg	0.30	0.30	0.26	26601

Table: Evaluation Matrix

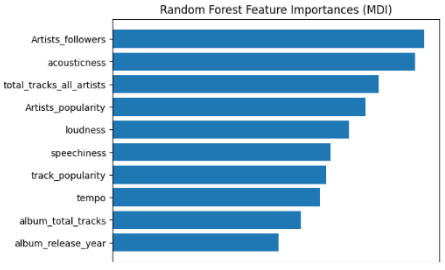


Figure: Feature Importance Chart

4.2.1 Grid Search Best Parameters: Random Forest Classifier

We evaluated the performance of a Random Forest Classifier for multi-class classification using the best hyperparameters determined by grid search. The Random Forest Classifier with the best parameters achieves moderate performance, with an accuracy of **29.62%** and variability in metrics across classes as result shown in metric. The following parameters were identified as optimal for the Random Forest Classifier using grid search. These parameters represent the best configuration to achieve the model's performance on the dataset. Grid Search identified the following optimal hyperparameters for the Random Forest model:

Bootstrap: True, enabling the use of bootstrapping for increased diversity and reduced overfitting;

Criterion: Gini, utilizing Gini impurity to assess split quality;

Max Features: sqrt, considering a subset of features equal to the square root of the

0	0.19	0.08	0.12	2386
1	0.23	0.06	0.09	1133
2	0.36	0.43	0.39	1484
3	0.27	0.22	0.24	1452
4	0.24	0.39	0.30	2455
5	0.31	0.04	0.07	363
6	0.30	0.49	0.37	3694
7	0.24	0.02	0.04	604
8	0.17	0.02	0.03	1229
9	0.43	0.05	0.08	1608
10	0.31	0.24	0.27	3313
11	0.36	0.20	0.25	1953
12	0.55	0.14	0.22	652
13	0.31	0.59	0.41	4275
accuracy			0.30	26601
macro avg	0.30	0.21	0.21	26601
weighted avg	0.30	0.30	0.26	26601

total number at each split to minimize correlation between trees and prevent overfitting to specific features, especially beneficial given the dataset's numerous features;
Max Depth: None, allowing unrestricted tree growth for potentially capturing complex relationships within the data while requiring careful monitoring for overfitting; and
N Estimators: 100, balancing model complexity and computational cost with a suitable ensemble size for robust classification. These settings suggest a preference for flexibility and diversity within the forest, enabling effective pattern learning and generalization ability for the given dataset and task.

4.2.2 Binary Classification: Random Forest Without Oversampling

Evaluate the performance of a Random Forest Classifier for binary classification without any oversampling or imbalance treatment. The Random Forest model achieves high accuracy (97.11%) and exceptional performance for the majority class (Class 0). However, it struggles significantly with the minority class (Class 1), achieving only 26% recall as shown in table.

	precision	recall	f1-score	support
0	0.97	1.00	0.99	25705
1	0.68	0.26	0.38	896
accuracy			0.97	26601
macro avg	0.83	0.63	0.68	26601
weighted avg	0.97	0.97	0.96	26601

4.2.3 Binary Classification: RC With SMOTE Oversampling

Evaluate the performance of a Random Forest Classifier for binary classification after addressing class imbalance using SMOTE (Synthetic Minority Oversampling Technique). Using **SMOTE** significantly improves the model's ability to identify the minority class (Class 1). Recall for Class 1 increased from **26% to 61%**, and the F1-score improved from **38% to 50%**, demonstrating better balance. While precision for Class 1 remains low as result shown in table.

	precision	recall	f1-score	support
0	0.99	0.97	0.98	25705
1	0.42	0.61	0.50	896
accuracy			0.96	26601
macro avg	0.70	0.79	0.74	26601
weighted avg	0.97	0.96	0.96	26601

4.3 AdaBoost with Default Parameters: Multi-Class Classification

The AdaBoost model was trained using default parameters. Evaluating the performance of an AdaBoost Classifier for multi-class classification using default parameters. The AdaBoost model with default parameters achieved limited success, with **29.07% accuracy** and poor performance while it fails to generalize effectively. Hyperparameter tuning are essential to unlock the full potential of AdaBoost.

	precision	recall	f1-score	support
0	0.22	0.21	0.22	2386
1	0.34	0.04	0.07	1133
2	0.33	0.35	0.34	1484
3	0.22	0.34	0.26	1452
4	0.23	0.30	0.26	2455
5	0.12	0.00	0.01	363
6	0.27	0.55	0.36	3694
7	0.12	0.00	0.01	604
8	0.15	0.02	0.03	1229
9	0.14	0.10	0.11	1608
10	0.31	0.17	0.22	3313
11	0.38	0.12	0.19	1953
12	0.50	0.00	0.01	652
13	0.41	0.56	0.47	4275
accuracy			0.29	26601
macro avg	0.27	0.20	0.18	26601
weighted avg	0.29	0.29	0.26	26601

4.3.1 AdaBoost After Parameter Tuning : Multi-Class Classification

Hyperparameter optimization for the AdaBoost Classifier focused on enhancing its multi-class classification performance. Grid Search identified the following optimal settings:

Learning Rate: 1.0: This parameter governs the contribution of each weak learner (typically a decision tree) to the overall ensemble. A learning rate of 1.0 suggests that each weak learner's output is given full weight in the final prediction. This aggressive learning rate may help the model adapt more quickly to complex patterns in the data.

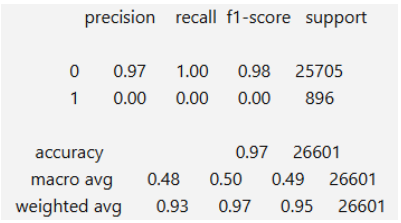
Number of Estimators: 200: This parameter specifies the total number of weak learners combined to form the ensemble. A higher number of estimators (200 in this case) can potentially improve accuracy by capturing more nuanced relationships within the data. However, it also increased computational cost and training time became longer.

	precision	recall	f1-score	support
0	0.25	0.22	0.23	2386
1	0.28	0.11	0.15	1133
2	0.38	0.36	0.37	1484
3	0.24	0.31	0.27	1452
4	0.24	0.35	0.28	2455
5	0.25	0.01	0.02	363
6	0.31	0.53	0.39	3694
7	0.12	0.01	0.02	604
8	0.16	0.02	0.03	1229
9	0.18	0.11	0.13	1608
10	0.27	0.17	0.20	3313
11	0.32	0.24	0.27	1953
12	0.53	0.08	0.14	652
13	0.42	0.58	0.49	4275
accuracy			0.31	26601
macro avg	0.28	0.22	0.21	26601
weighted avg	0.29	0.31	0.28	26601

Despite these optimizations, the AdaBoost Classifier demonstrated only a marginal improvement in overall accuracy (30.90%). While F1-scores for some classes showed positive changes, the model

continued to exhibit challenges in accurately classifying minority classes. This limitation is likely attributed to the data imbalance observed in the dataset, as highlighted in Table.

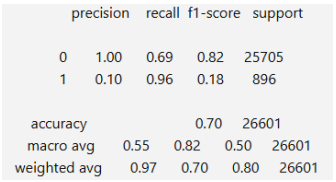
4.3.2. AdaBoost on Non-Treated Data: Binary Classification



	precision	recall	f1-score	support
0	0.97	1.00	0.98	25705
1	0.00	0.00	0.00	896
accuracy			0.97	26601
macro avg	0.48	0.50	0.49	26601
weighted avg	0.93	0.97	0.95	26601

Adaboost was also trained on non-treated binary class data with learning rate =1.0 and n_estimators=200. Evaluating the performance of an AdaBoost Classifier for binary classification on imbalanced data without any treatment for class imbalance. The AdaBoost model on non-treated data achieves high accuracy (**96.63%**) due to its strong performance for the majority class (Class 0). However, it entirely fails to classify the minority class (Class 1), highlighting the critical need for class imbalance treatment as shown in table.

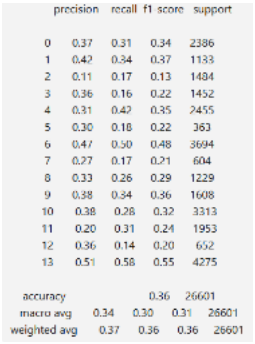
4.3.3. Binary Classification: AdaBoost on SMOTE-Treated Data



	precision	recall	f1-score	support
0	1.00	0.69	0.82	25705
1	0.10	0.96	0.18	896
accuracy			0.70	26601
macro avg	0.55	0.82	0.50	26601
weighted avg	0.97	0.70	0.80	26601

Using the same parameters for untreated data, model was also trained on SMOTE-trained data. Evaluate the performance of an AdaBoost Classifier for binary classification after addressing class imbalance using SMOTE (Synthetic Minority Oversampling Technique). Using SMOTE significantly enhances the recall for the minority class (Class 1) from **0% to 96%**, addressing the critical limitation of the non-treated model. However, precision for Class 1 drops drastically (**10%**), leading to a high false positive rate as values are shown in table.

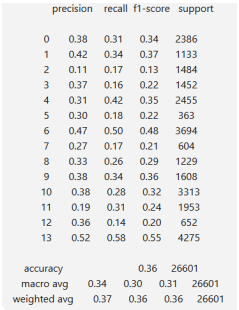
4.4. Multi-Class Classification: Gradient Boosting Machine (GBM)



	precision	recall	f1-score	support
0	0.37	0.31	0.34	2386
1	0.42	0.34	0.37	1133
2	0.11	0.17	0.13	1484
3	0.36	0.16	0.22	1452
4	0.31	0.42	0.35	2455
5	0.30	0.18	0.22	363
6	0.47	0.50	0.48	3694
7	0.27	0.17	0.21	604
8	0.33	0.26	0.29	1229
9	0.38	0.34	0.36	1608
10	0.38	0.28	0.32	3313
11	0.20	0.31	0.24	1953
12	0.36	0.14	0.20	652
13	0.51	0.58	0.55	4275
accuracy			0.36	26601
macro avg	0.34	0.30	0.31	26601
weighted avg	0.37	0.36	0.36	26601

The GBM was trained on default parameters. The GBM model achieves **36.07% accuracy**, with strong performance for dominant classes but significant struggles with minority ones. While it shows promise for majority classes, further tuning, imbalance treatment as shown in table.

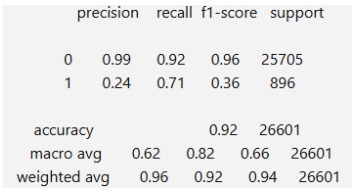
4.4.1. Multi-Class Classification: Gradient Boosting Classifier (GBC) with 150 Estimators



	precision	recall	f1-score	support
0	0.38	0.31	0.34	2386
1	0.42	0.34	0.37	1133
2	0.11	0.17	0.13	1484
3	0.37	0.16	0.22	1452
4	0.31	0.42	0.35	2455
5	0.30	0.18	0.22	363
6	0.47	0.50	0.48	3694
7	0.27	0.17	0.21	604
8	0.33	0.26	0.29	1229
9	0.38	0.34	0.36	1608
10	0.38	0.28	0.32	3313
11	0.19	0.31	0.24	1953
12	0.36	0.14	0.20	652
13	0.52	0.58	0.55	4275
accuracy			0.36	26601
macro avg	0.34	0.30	0.31	26601
weighted avg	0.37	0.36	0.36	26601

The grid search was computationally costly; therefore, the parameter tuning was done manually. We focused on the number of estimators in this case. Using 150 estimators in the Gradient Boosting Classifier provides a marginal improvement in accuracy (**36.09%**) and class-specific F1-scores. While dominant classes see better results, the model continues to struggle with minority classes as shown in table.

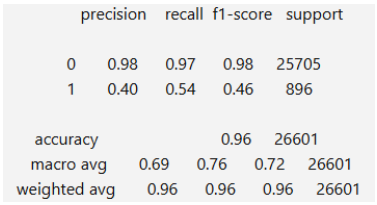
4.4.2. Binary Classification: Gradient Boosting Classifier (GBC)



	precision	recall	f1-score	support
0	0.99	0.92	0.96	25705
1	0.24	0.71	0.36	896
accuracy			0.92	26601
macro avg	0.62	0.82	0.66	26601
weighted avg	0.96	0.92	0.94	26601

The Gradient Boosting Classifier achieves high accuracy (**91.60%**) and performs exceptionally well for the majority class (Class 0). It also demonstrates improved recall for the minority class (Class 1, **71%**). However, low precision for Class 1 (**24%**) indicates a high false-positive rate as shown in table.

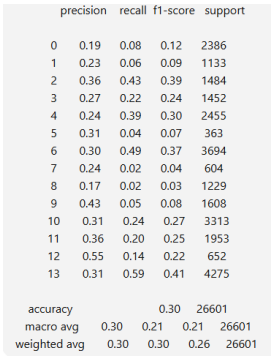
4.4.3. Binary Classification: Gradient Boosting Classifier (GBC) on SMOTE-Treated Data



	precision	recall	f1-score	support
0	0.98	0.97	0.98	25705
1	0.40	0.54	0.46	896
accuracy			0.96	26601
macro avg	0.69	0.76	0.72	26601
weighted avg	0.96	0.96	0.96	26601

Using SMOTE significantly improves the Gradient Boosting Classifier's ability to handle the minority class (Class 1), achieving a recall of **54%** and an F1-score of **46%**. While overall accuracy remains high (**95.76%**).

4.5.1. XGBoost: Multi-Class Classification



	precision	recall	f1-score	support
0	0.19	0.08	0.12	2386
1	0.23	0.06	0.09	1133
2	0.36	0.43	0.39	1484
3	0.27	0.22	0.24	1452
4	0.24	0.39	0.30	2455
5	0.31	0.04	0.07	363
6	0.30	0.49	0.37	3694
7	0.24	0.02	0.04	604
8	0.17	0.02	0.03	1229
9	0.43	0.05	0.08	1608
10	0.31	0.24	0.27	3313
11	0.36	0.20	0.25	1953
12	0.55	0.14	0.22	652
13	0.31	0.59	0.41	4275
accuracy			0.30	26601
macro avg	0.30	0.21	0.21	26601
weighted avg	0.30	0.30	0.26	26601

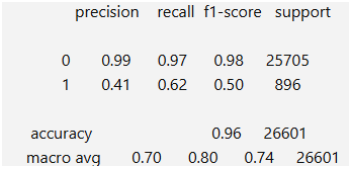
XGBoost algorithm was trained for multi-class classification using default settings. The XGBoost model achieves moderate classification performance with an accuracy of **29.62%**. While it performs well for majority classes like Class 13, its struggles with minority classes highlight the need for additional tuning and class imbalance treatment.

4.5.2. Binary Classification: XGBoost

	precision	recall	f1-score	support
0	0.98	0.99	0.99	25705
1	0.63	0.36	0.46	896
accuracy			0.97	26601
macro avg	0.80	0.67	0.72	26601
weighted avg	0.97	0.97	0.97	26601

Evaluate the performance of an XGBoost model for binary classification using default settings. The XGBoost model achieves high accuracy (**97.13%**) and strong precision for the minority class (**63%**), indicating reasonable reliability in identifying Class 1 instances. However, the recall for Class 1 remains low (**36%**), reflecting challenges in capturing all actual Class 1 instances.

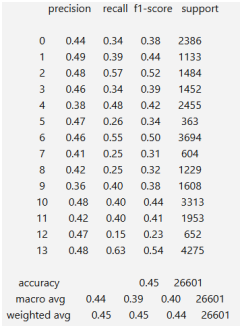
4.5.3. Binary Classification: XGBoost with SMOTE-Treated Data



	precision	recall	f1-score	support
0	0.99	0.97	0.98	25705
1	0.41	0.62	0.50	896
accuracy			0.96	26601
macro avg	0.70	0.80	0.74	26601

Evaluate the performance of an XGBoost model for binary classification after applying SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance. The XGBoost model with SMOTE treatment achieves balanced improvement for the minority class (Class 1), boosting recall to **62%** and F1-score to **50%**. While overall accuracy remains high (**95.77%**).

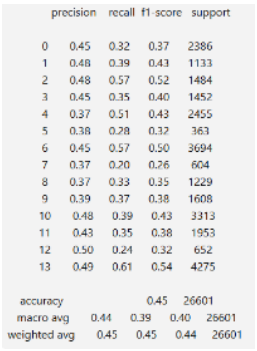
4.6.1. Multi-Class Classification: Multi-Layer Perceptron (MLP) Classifier with Default Parameters



	precision	recall	f1-score	support
0	0.44	0.34	0.38	2386
1	0.49	0.39	0.44	1133
2	0.48	0.57	0.52	1484
3	0.46	0.34	0.39	1452
4	0.38	0.48	0.42	2455
5	0.47	0.26	0.34	363
6	0.46	0.55	0.50	3694
7	0.41	0.25	0.31	604
8	0.42	0.25	0.32	1229
9	0.36	0.40	0.38	1608
10	0.48	0.40	0.44	3313
11	0.42	0.40	0.41	1953
12	0.47	0.15	0.23	652
13	0.48	0.63	0.54	4275
accuracy			0.45	26601
macro avg	0.44	0.39	0.40	26601
weighted avg	0.45	0.45	0.44	26601

A Multi-Layer Perceptron (MLP) classifier was applied to the multi-class classification task using its default configuration. The model demonstrated moderate performance, achieving an overall accuracy of 44.66%. While the MLP exhibits reasonable accuracy for majority classes, such as classes 13 and 6, it encounters difficulties in effectively classifying minority classes.

4.6.2. Multi-Class Classification: Multi-Layer Perceptron (MLP) with Hidden Layer Size = 100



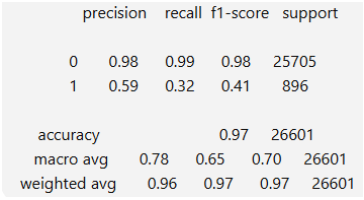
	precision	recall	f1-score	support
0	0.45	0.32	0.37	2386
1	0.48	0.39	0.43	1133
2	0.48	0.57	0.52	1484
3	0.45	0.35	0.40	1452
4	0.37	0.51	0.43	2455
5	0.38	0.28	0.32	363
6	0.45	0.57	0.50	3694
7	0.37	0.20	0.26	604
8	0.37	0.33	0.35	1229
9	0.39	0.37	0.38	1608
10	0.48	0.39	0.43	3313
11	0.43	0.35	0.38	1953
12	0.50	0.24	0.32	652
13	0.49	0.61	0.54	4275
accuracy			0.45	26601
macro avg	0.44	0.39	0.40	26601
weighted avg	0.45	0.45	0.44	26601

A Multi-Layer Perceptron (MLP) model was evaluated for multi-class classification, employing a single hidden layer with 100 neurons (hidden_layer_sizes=(100,)). This architectural choice aims to provide the network with sufficient capacity to learn complex relationships within the data. The hidden layer size significantly impacts the model's ability to extract meaningful features and patterns, influencing its overall performance.

However, despite this adjustment, the MLP with a hidden layer size of 100 exhibited performance similar to the default model. It achieved an accuracy of 44.63% and a macro F1-score of 40%. While the increased hidden layer size led to improved recall for certain classes, it did

not yield a substantial overall performance boost. This suggests that the model's limitations might stem from factors beyond the hidden layer architecture, such as data complexity or potential overfitting..

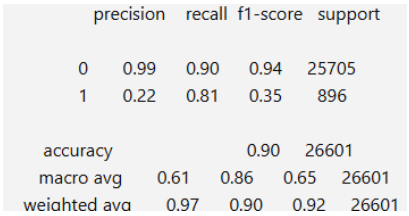
4.6.3.1 Binary Classification: Multi-Layer Perceptron (MLP)



	precision	recall	f1-score	support
0	0.98	0.99	0.98	25705
1	0.59	0.32	0.41	896
accuracy			0.97	26601
macro avg	0.78	0.65	0.70	26601
weighted avg	0.96	0.97	0.97	26601

A Multi-Layer Perceptron (MLP) model, configured with hidden_layer_sizes=(100,), max_iter=200, and random_state=0, was employed for binary classification. The model achieved high overall accuracy, reaching 96.96%. This performance is largely driven by its excellent accuracy in classifying the majority class (Class 0). However, a closer examination reveals a limitation in the model's ability to correctly identify instances of the minority class (Class 1). Specifically, the MLP exhibits a limited recall of 32% for Class 1, indicating that a significant proportion of actual Class 1 instances are misclassified. This performance discrepancy underscores the challenge posed by class imbalance, where the model's training process might be biased towards the majority class.

4.6.3.2 Binary Classification: Multi-Layer Perceptron (MLP) with SMOTE-Resampled Data



	precision	recall	f1-score	support
0	0.99	0.90	0.94	25705
1	0.22	0.81	0.35	896
accuracy			0.90	26601
macro avg	0.61	0.86	0.65	26601
weighted avg	0.97	0.90	0.92	26601

Evaluate the performance of a Multi-Layer Perceptron (MLP) model for binary classification after applying SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance. The MLP model trained on SMOTE-resampled data significantly improves recall for the minority class (Class 1) to **81%**, addressing class imbalance effectively. However, precision for Class 1 remains low (**22%**), resulting in a modest F1-score (**35%**).

4.6.4 Keras Sequential Model

4.6.4.1. Model Accuracy and Loss with SMOTE:

The Keras Sequential model employs a three-layer architecture with 128, 64, and 1 neuron in the input, hidden, and output layers, respectively. ReLU activation is used in the input and hidden layers to introduce non-linearity for learning complex patterns, while sigmoid activation in the output layer produces probability scores for binary classification. This architecture, compiled with binary_crossentropy loss and the adam optimizer, effectively learns from the SMOTE-balanced dataset, demonstrating high accuracy (reaching ~0.98 by the 50th epoch) and decreasing loss (reducing to ~0.10). These results indicate strong learning, generalization, and accurate predictions, likely attributed to the careful architecture design and the balanced dataset provided by SMOTE.

4.6.4.2. Model Accuracy and Loss with Original Data:

When same architecture was employed and model trained on the original dataset, the model also achieves high accuracy, starting at around 0.964 and increasing to approximately 0.976 by the 50th epoch. This indicates that the model performs well even without the synthetic data augmentation provided by SMOTE.

The model loss with the original data is lower initially, starting at around 0.10 and decreasing to approximately 0.06 by the 50th epoch. This shows that the model's predictions are consistently accurate, with minimal error.

Both the SMOTE-augmented and original datasets yield high accuracy and low loss for the Keras Sequential Model. The model demonstrates strong learning capabilities and generalization, making it suitable for the given task as shown in figure.

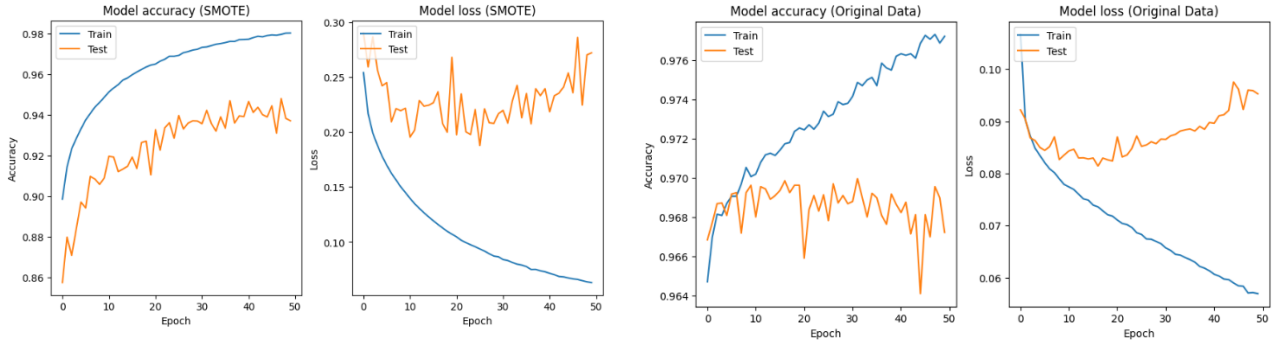
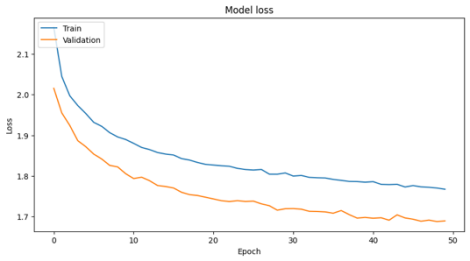
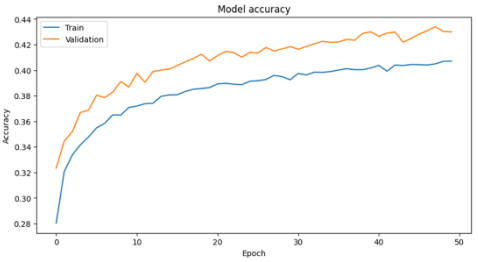


Figure: 4.6. Keras Sequential Model

4.6.4.3. Model Train and VAL Accuracy CNN

The implemented CNN model utilizes a *1D convolutional layer with 32 filters* and a *kernel size of 3* to extract local patterns from the input features, followed by max pooling for downsampling. Subsequently, the data is flattened and passed through two dense layers with 128 units and *ReLU* activation for learning complex relationships. *Dropout* is employed for regularization to prevent overfitting. Finally, a *softmax* activation layer outputs probability estimates for each class in the multi-class classification problem. This architecture aims to leverage the benefits of CNNs in capturing local patterns while adapting them to tabular data by treating features as a sequence. The choice of *'adam'* optimizer and *'sparse_categorical_crossentropy'* loss function is commonly used in multi-class classification tasks.



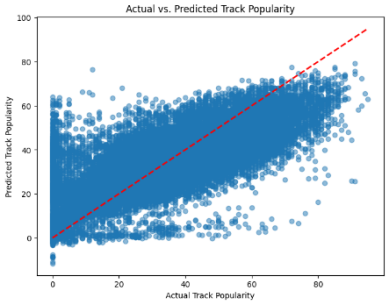


However, the model achieved a test accuracy of only 0.439, suggesting suboptimal performance. This indicates that the model struggles to generalize to unseen data and might be overfitting or not effectively capturing the underlying patterns in the dataset. The relatively high test loss (1.686) further supports this conclusion.

4.7. Advance Regression

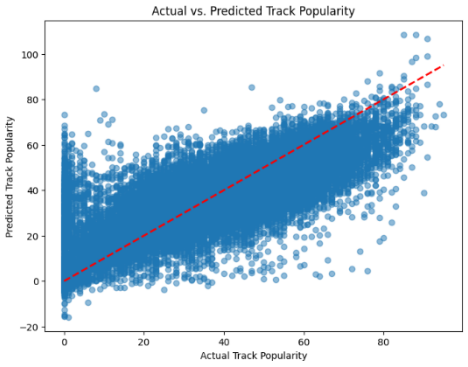
4.7.1. Random Forest Regressor

We implemented a Random Forest Regressor to predict the popularity of tracks based on various features. The MSE of **135.30** indicates the average squared difference between the predicted and actual track popularity scores. While lower is better, this value suggests there is still room for improvement in reducing prediction errors. The R^2 value of **0.707** shows that our model explains **70.7%** of the variance in track popularity. This indicates a strong predictive power, but not perfect accuracy as shown in figure.



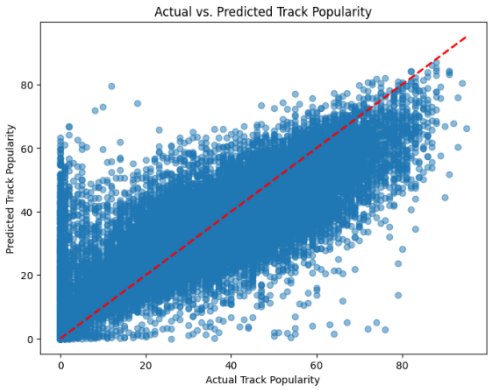
4.7.2. XGBoost Regression

We used XGBoost Regression to predict track popularity, and here's how it performed:



Mean Squared Error (MSE): 156.40 this represents the average squared difference between predicted and actual values. A lower MSE is ideal, so there's room for improvement. **R-squared (R^2): 0.662** This means the model explains about **66.2% of the variance** in track popularity, indicating a decent but not perfect fit. XGBoost provides a solid predictive baseline, but the error rate suggests that fine-tuning could improve accuracy as shown in figure.

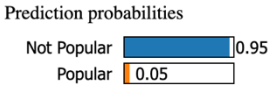
4.7.3. Gradient Boosting Regressor

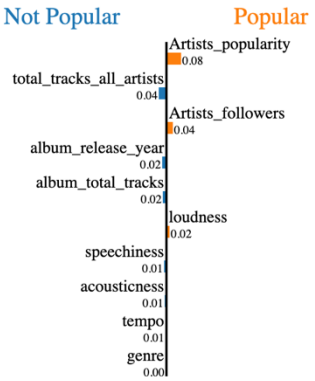


We evaluated Gradient Boosting Regressor for predicting track popularity, and here are the results: Mean Squared Error (MSE): 180.05 → Higher error means predictions have more deviation from actual values. R-squared (R^2): 0.611 → The model explains **61.1% of the variance**, indicating moderate predictive power but lower than expected. While Gradient Boosting captures some trends in track popularity, its performance is weaker compared to other models.

4.8. Explainable AI

4.8.1. LIME





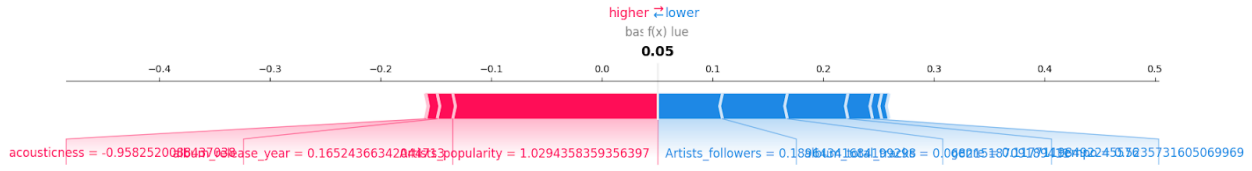
Feature	Value
Artists_popularity	1.03
total_tracks_all_artists	-0.17
Artists_followers	0.19
album_release_year	0.17
album_total_tracks	0.07
loudness	0.65
speechiness	-0.40
acousticness	-0.96
tempo	0.52
genre	0.12

The LIME (Local Interpretable Model-agnostic Explanations) visualization explains the prediction of the 10th record. Here's a breakdown of what the LIME plot indicates:

For a specific prediction (instance 25), the LIME (Local Interpretable Model-agnostic Explanations) analysis revealed the individual feature contributions to the Random Forest model's output. it was observed that 'Artists_popularity' had the most significant positive influence (0.083) on the prediction, while 'Artists_followers' also contributed positively (0.035). Conversely, features like 'total_tracks_all_artists' and 'album_release_year' exhibited negative influences (-0.046 and -0.024 respectively). The remaining features had relatively minor impacts on the prediction. This localized interpretation provided by LIME enhances the transparency and understanding of the Random Forest model's decision-making process for individual instances.

4.8.2. SHAP

The SHAP (SHapley Additive exPlanations) plots provided aim to explain the predictions of a machine learning model, in this case, the random forest model. We tried to understand feature influence as shown in figure. These SHAP plots provide a detailed understanding of how individual features influence the predictions of the random forest model. The global plot offers an overview of feature importance across all predictions, while the single-instance plots provide insights into how features contribute to specific predictions, illustrating the nuanced behavior of the model. For a single instance (record number 25), we got the SHAP values: These numbers `[ 0.18537728, -0.05943682, 0.00056254, 0.0138502 , -0.05934383, -0.05465295, -0.00532268, 0.01057148, -0.02219024, -0.00941497]`, basically show us how much each feature in our data affected a single prediction made by the model. A positive value, like 0.185 for 'Artists_popularity', means that feature pushed the prediction higher or towards a certain category. A negative value, like -0.059 for 'Artists_followers', means it pulled the prediction lower or away from that category. The bigger the number, the stronger the push or pull. So, we can see that 'Artists_popularity' was a big factor in this particular prediction, while others like 'Artists_followers' had a smaller, opposite effect. Looking at these values gives us a peek into how the model is thinking when it makes a prediction, which is really helpful for understanding why it's making the choices it does.



4.8.2. LIME vs SHAP explaination Comparison

Top features: In results, both LIME and SHAP seem to identify "Artists_popularity" as the most influential feature, positively impacting the prediction. This agreement strengthens the confidence in this feature's importance.

Direction of influence: It is likely to find that most features have a similar direction of influence (positive or negative) in both LIME and SHAP explanations. This consistency further validates the interpretation of feature effects.

Magnitude of influence: There might be some differences in the magnitude of feature importance scores between LIME and SHAP. This can be due to the different methodologies they use. However, the relative importance rankings of features is broadly similar.