Comparative Study on Various Algorithms Used to Perform Music Genre Classification

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
in
Computer Science

by

Avni Shah 20BCI0119
Khushi Sharma 20BCI0243

Under the guidance of

Dr. Anusha

SCOPE

VIT, Vellore.

NOV, 2022

DECLARATION

We hereby declare that the project entitled "**Comparative Study On Various Algorithms Used To Perform Music Genre Classification**" submitted by us, for the award of the degree of Bachelor of Technology in Computer Science to VIT is a record of bonafide work carried out by me under the supervision of Dr. Anusha.

We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree ordiploma in this institute or any other institute or university.

Place : Vellore

Date : 10.11.2022

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled "**Comparative Study On Various Algorithms Used To Perform Music Genre Classification**" submitted by Avni Shah(20BCI0119) and Khushi Sharma(20BCI0243), SCOPE, VIT, for the award of the degree of Bachelor of Technology in Computer Science, is a record of bonafide work carried out by us under my supervision during the period, 20. 07 2022 to 7.12.2022, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 10:11:22 **Signature of the Guide**

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

The project "**Comparative Study on Various Algorithms Used To Perform Music Genre Classification"** was made possible because of inestimable inputs from everyone involved, directly or indirectly. We would first like to thank my guide, **Dr. Anusha,** who was highly instrumental in providing not only a required and innovative base for the project but also crucial and constructive inputs that helped make my final product. Our guide has helped us perform research in the specified area and improve our understanding in the area of web development and internet of things and we are very thankful for his support all throughout the project.

Finally, we would like to thank **Vellore Institute of Technology**, for providing us with a flexible choice and execution of the project and for supporting us research and execution related to the project.

Student Name:
Khushi Sharma-20BCI0243
Avni Shah-20BCI0119

Executive Summary

Music genre classification is one of the topics digital music processing is interested in. In this study, we extract the audio features using digital signal processing methods and then classify music using various models such as KNN, SVMs, Naive Bayes, Random Forests, XGBoost, and Logistic Regression and so on. We compare the performances of the models based on the results obtained. In the study, GTZAN dataset has been used and the highest accuracy was obtained using the XGBooster algorithm. We aim to compare these various existing classification models which work using both supervised and unsupervised learning techniques in order to understand which model performs better.

CONTENTS

List of Figures

List of Tables

1.INTRODUCTION

1.1. OBJECTIVE

This project aims to understand patterns in the music in order to classify them accurately. Content and audio signal will be used as major parameters in order to perform classification. The aim is to compare various existing classification models which work using both supervised and unsupervised learning techniques in order to understand which model performs better. We have taken into consideration some widely used models such as SVM, KNN, XGBoost, Logistic Regression, Random Forest, Decision Trees, and Naive Bayes and compared their accuracy to understand which one of them performs better. We have used GTZAN dataset for this purpose which has an extensive collection of music belonging to various categories like pop, blues, metal, and country, classical, disco, jazz and

1.2 MOTIVATION

Music is described as an art form and a cultural activity whose medium is sound. It is not only for entertainment or pleasure but is being used for a wide variety of purposes due its social and physiological effects. Today there are more than 35 million songs available online and thus it becomes very important to categorize them. Several music industries are making efforts to provide filtered and categorized content to their customers and the public, be it spotify, gaana, amazon music etc. Music genres are created in order to describe and categorize music but, there are no strict boundaries available. Thus, the classification of music has become a wide area of research to understand patterns in each of these genres and classify them. This area also plays an important role in music retrieval systems and song suggestion systems. Efficient and accurate systems which can intelligently classify the vast amount of audio data available has become extremely important and there are many researches being done.

1.3 BACKGROUND

The background of our project is the music genre classification using various algorithms like SVM, K nearest neighbors, Convolutional neural networks, Stochastic Gradient Descent, Logistic Regression, Recurrent neural networks, random forest, decision trees, K-means clustering, Cross gradient booster and naïve bayes.

2. PROJECT DESCRIPTION AND GOALS

- Music is described as an art form and a cultural activity whose medium is sound. It is not only for entertainment or pleasure but is being used for a wide variety of purposes due its social and psychological effects.
- Several applications in the music industry are making wholesome efforts to provide filtered and categorized content to their customers, like spotify, gaana, jio-saavn, etc.
- Thus, the classification of music has become a wide area of research to understand patterns in each of these genres and classify them.
- Research are being done to develop efficient and accurate systems which can intelligently classify vast amount of available audio data.
- Content and audio signal will be used as major parameters to perform classification.
- We aim to compare these various existing classification models which work using both supervised and unsupervised learning techniques to understand which model performs better.
- We have taken into consideration some widely used models such as SVM, KNN, XGBoost, Logistic Regression, Random Forest, Decision Trees, and Naive Bayes to understand which one of them performs more accurately.
- We will be using GTZAN dataset for this purpose which has an extensive collection of music belonging to various categories like pop, blues, metal, and country, classical, disco, jazz and hip-hop.

3.TECHNICAL SPECIFICATION

3.1 Dataset used:

(I) Dataset

The **GTZAN dataset** is the most-used public dataset for evaluation in machine listening research in Music Genre Recognition (MGR). The files were collected in 2000-2001 from various sources that included personal CDs, radio, microphone recordings, to represent various recording conditions. It is a collection of 1,000 music with 30-second, 22050 Hz sampling frequency and 16 bits. Following are the specifications of the dataset:

● **Genres Original** - Genres in the GTZAN include blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock and all these genres have 100 music files each.

● **Images original** - A visual representation for each audio file which helps as an input for the various neural network models.

● .**CSV files** - The dataset contains 2 CSV files that contain features of the audio files.

3.2 Tools and Algorithms used:

- We have used **Jupyter Notebook** for our project**.** That is used for writing our code in python language.
- As Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text, and multimedia resources in a single document.
- Computational notebooks have been around for decades, but Jupyter has exploded in popularity over the past couple of years.
- And the OS can be anything above Windows 2000. And hardware requirement is: 2GB and above RAM, 2 GB of hardware storage.

4.DESIGN APPROACH AND DETAILS

4.1 Methodology adapted:

- Understanding Audio-Audio Data, Audio Features

- EDA on the CSV files-EDA is going to be performed on the csv. This file contains the mean and variance for each audio file of the features analysed above.

- Principal Component Analysis-to visualize possible groups of genres

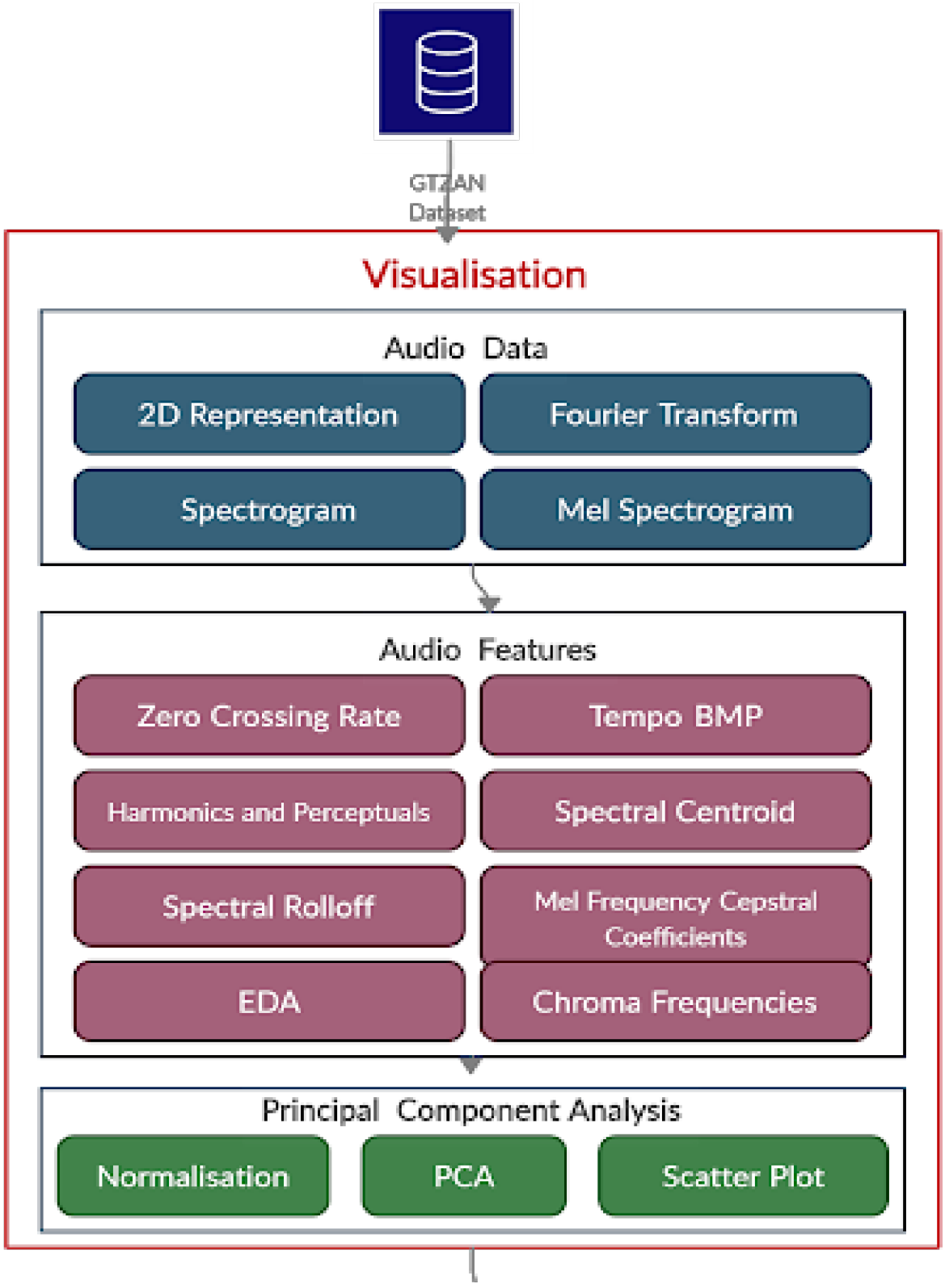
    - Normalization
    - PCA
    - The Scatter Plot

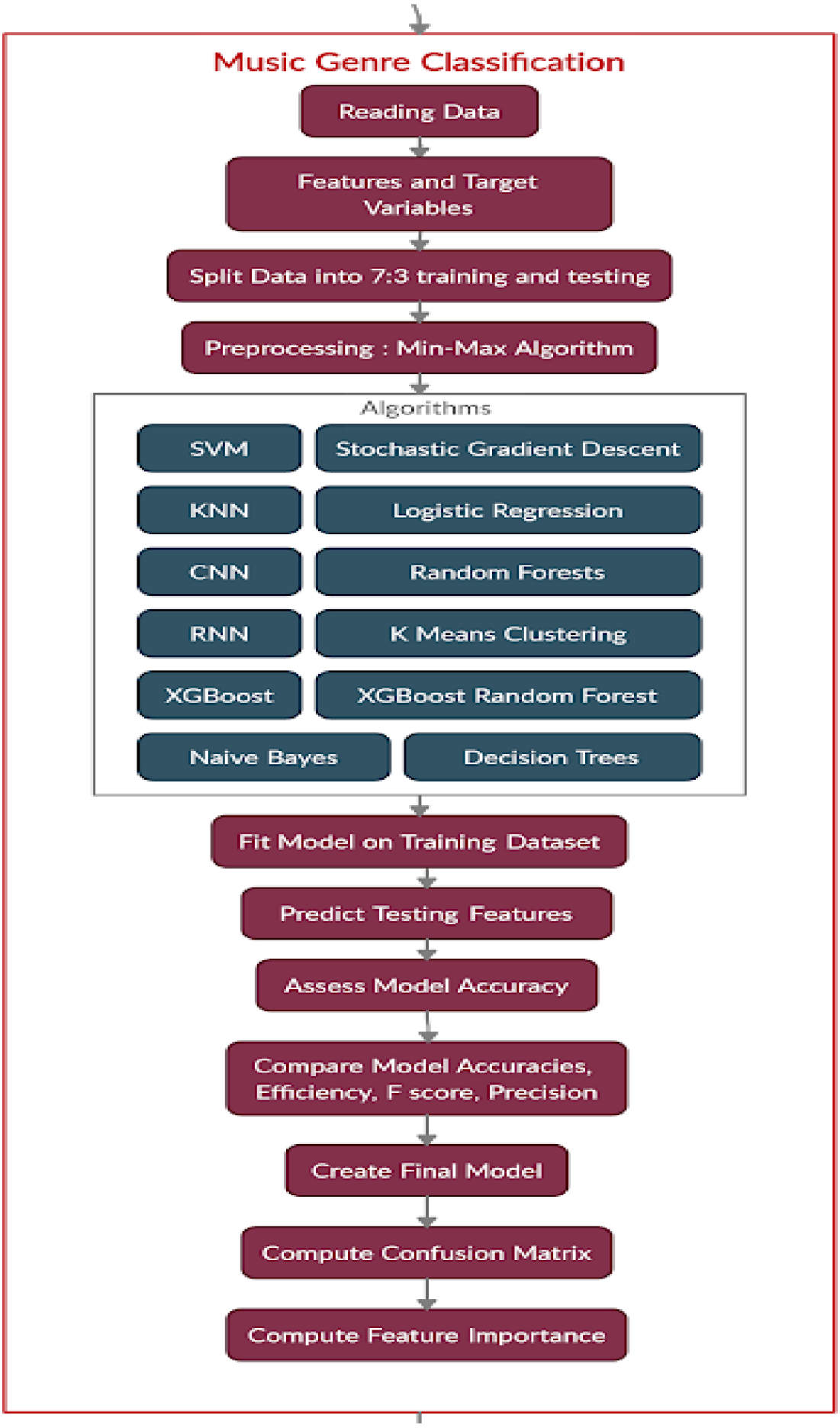
- Machine Learning Classification-Using the csv file, we can try to build a classifier that accurately predicts for any new audio file input it's genre.

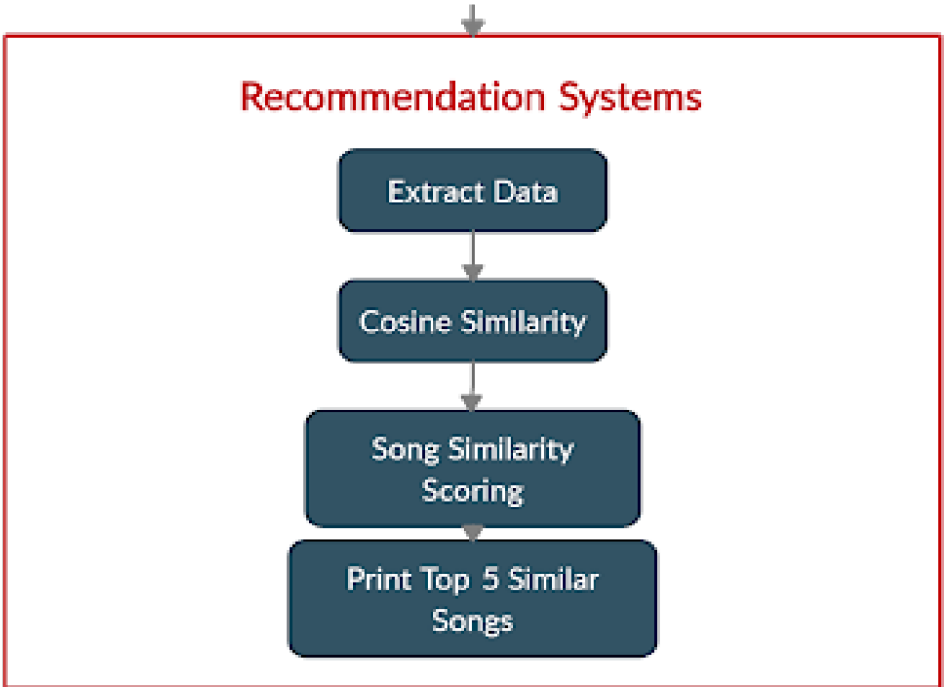
- Recommender Systems-"Recommender" Systems enable us for any given vector to find the best similarity, ranked in descending order, from the best match to the least good match.

- Putting the Similarity Function into Action-To find similar songs by giving a csv file (as in same genre)

4.2 Detailed Architecture Diagram




Music Genre Classification
Reading Data
Features and Target Variables
Split Data into 7:3 training and testing
Preprocessing : Min-Max Algorithm
Algorithms
SVM
Stochastic Gradient Descent
KNN
Logistic Regression
CNN
Random Forests
RNN
K Means Clustering
XGBoost
XGBoost Random Forest
Naive Bayes
Decision Trees
Fit Model on Training Dataset
Predict Testing Features
Assess Model Accuracy
Compare Model Accuracies, Efficiency, F score, Precision
Create Final Model
Compute Confusion Matrix
Compute Feature Importance



4.3 Audio Features

The following is a detailed description about the features that were used from the dataset. Time and frequency domain digital signal processing was done. Also mean, median, standard deviation were used in order to find out useful features.

- **Zero Crossing Rate**: Zero-crossing rate is defined as the number of sign changes of a signal in a certain period of time. The transition of the signal between negative and positive values is called Sign change.

- **Harmonics and Perceptual**: Harmonics are known to be those characteristics that human ears cannot distinguish; it represents the sound colour. Perceptual understanding of the wave represents the rhythm of sound and the emotion it carries.

- **Tempo BPM (beats per minute)**: A dynamic program-based beat tracker.

- **Chroma frequencies**: Chroma features represent audio where the entire broad spectrum is divided into 12 bins where each one of them represent a distinct semitone/chroma of the musical octave.

- **EDA:** It performs analysis on the features_30_sec.csv file. This file contains the mean and variance for each audio file.

- **Spectral Centroid:** Spectral centroid is a feature used on a frequency domain and indicates the point of the centre of gravity of the frequencies in the frequency bin.

- **Spectral Rolloff:** Spectral roll off is the calculated normalized frequency at which the sum of the low frequency power values of the sound reaches a particular rate in the total power spectrum. In short, it could be defined as the frequency value that corresponds to a certain ratio of the distribution in the spectrum. This rate is usually considered to be 85%.

- **Mel Frequency Coefficient of Cestrum-MFCC:** MFCCs are a set of features that describe the overall shape of the spectral envelope. It could be considered as a feature of timbre. The purpose of an MFCC is to adapt the cepstral coefficients to the human hearing system. Cepstral coefficients are linear scale. Steps included in an MFCC are Frame Blocking, Windowing, Fast Fourier Transform, Mel Frequency Wrapping and Spectrum, respectively.

5. SCHEDULE:

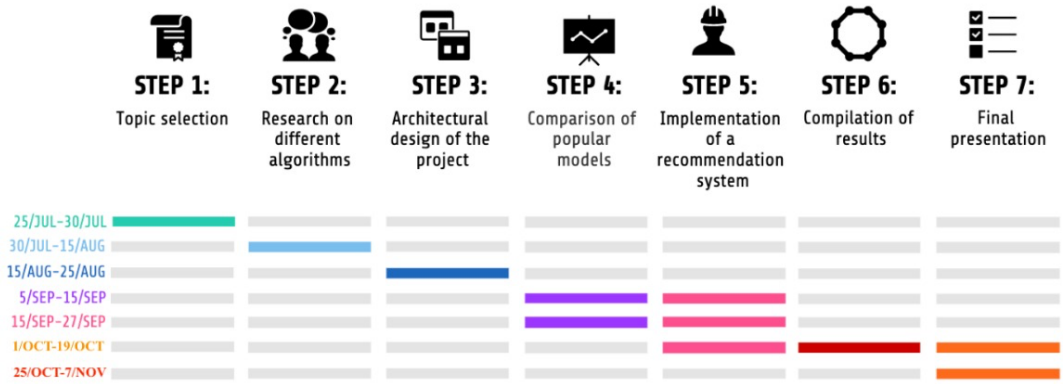


FIG. 5.1

6.PROJECT DEMONSTRATION:

We use librosa, the mother of audio files.

A. Understanding Audio

Initially, we explore the audio data. We use reggae.00036.wav file to understand the features and visualise them. We first calculate the value of sound and sample rate. We then trim the audio file such that the leading and trailing silence is eliminated.

```
In [74]: plt.figure(figsize = (16, 6))
         librosa.display.waveplot(y = audio_file, sr = sr, color = "#A300F9");
         plt.title("Sound Waves in Reggae 36", fontsize = 23);
```

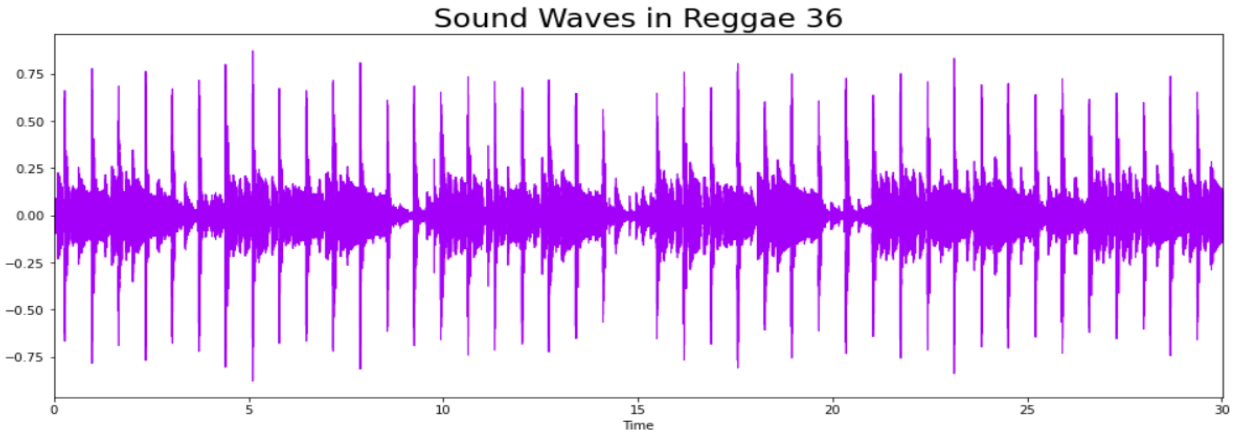


Figure 6.1. Visualizing sound waves in 2D space

Moving on, we visualise the spectrogram of the audio file, which is a representation of the spectrum of frequencies of a signal with respect to time. Given below, is an amplitude spectrogram converted into a decibel scale spectrogram. The audio file used is

reggae.00036.wav.

```
In [77]: # Convert an amplitude spectrogram to Decibels-scaled spectrogram.
         DB = librosa.amplitude_to_db(D, ref = np.max)

         # Creating the Spectogram
         plt.figure(figsize = (16, 6))
         librosa.display.specshow(DB, sr = sr, hop_length = hop_length, x_axis = 'time', y_axis = 'log',
                                  cmap = 'cool')
         plt.colorbar();
```

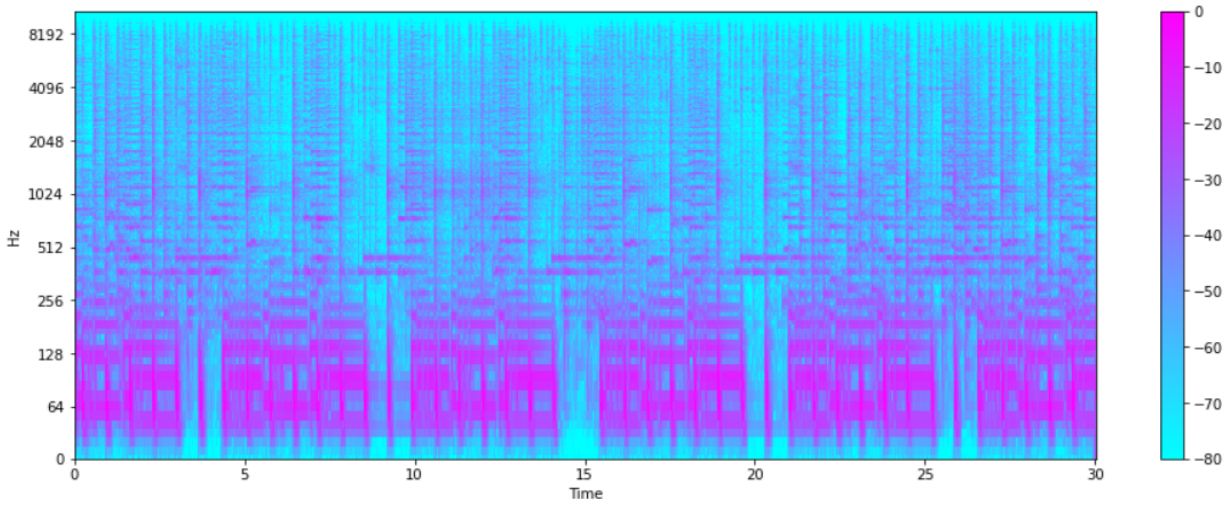


Figure 6.2. Decibel Scale Spectrogram - reggae.0036.wav

Next, we visualise the Mel Spectrogram. The Mel scale is a result of a non linear transformation of the frequency scale. It is a regular spectrogram but with the Mel scale on the y axis. The audio file used for this is **metal.00036.wav.**

```
In [78]: y, sr = librosa.load(f'{general_path}/genres_original/metal/metal.00036.wav')
         y, _ = librosa.effects.trim(y)


         S = librosa.feature.melspectrogram(y, sr=sr)
         S_DB = librosa.amplitude_to_db(S, ref=np.max)
         plt.figure(figsize = (16, 6))
         librosa.display.specshow(S_DB, sr=sr, hop_length=hop_length, x_axis = 'time', y_axis = 'log',
                                  cmap = 'cool');
         plt.colorbar();
         plt.title("Metal Mel Spectrogram", fontsize = 23);
```

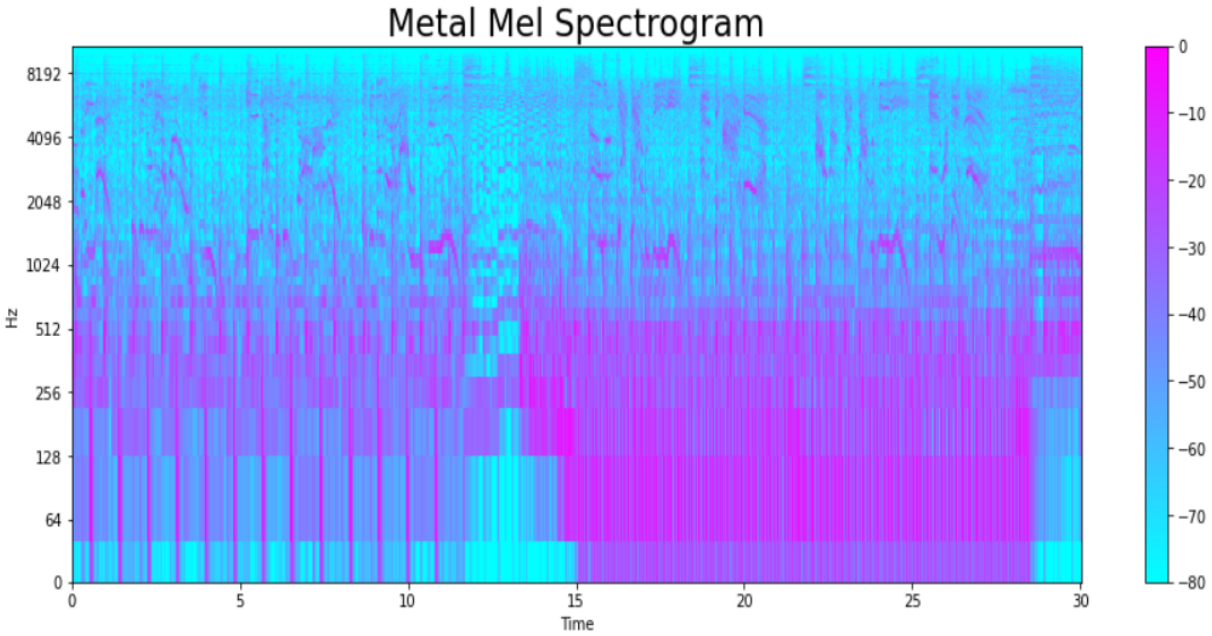


Figure 6.3. Metal Mel Spectrogram - metal.00036.wav

The same visualisation on classical.00036.wav gives the following result.

```
In [79]: y, sr = librosa.load(f'{general_path}/genres_original/classical/classical.00036.wav')
         y, _ = librosa.effects.trim(y)


         S = librosa.feature.melspectrogram(y, sr=sr)
         S_DB = librosa.amplitude_to_db(S, ref=np.max)
         plt.figure(figsize = (16, 6))
         librosa.display.specshow(S_DB, sr=sr, hop_length=hop_length, x_axis = 'time', y_axis = 'log',
                                  cmap = 'cool');
         plt.colorbar();
         plt.title("Classical Mel Spectrogram", fontsize = 23);
```

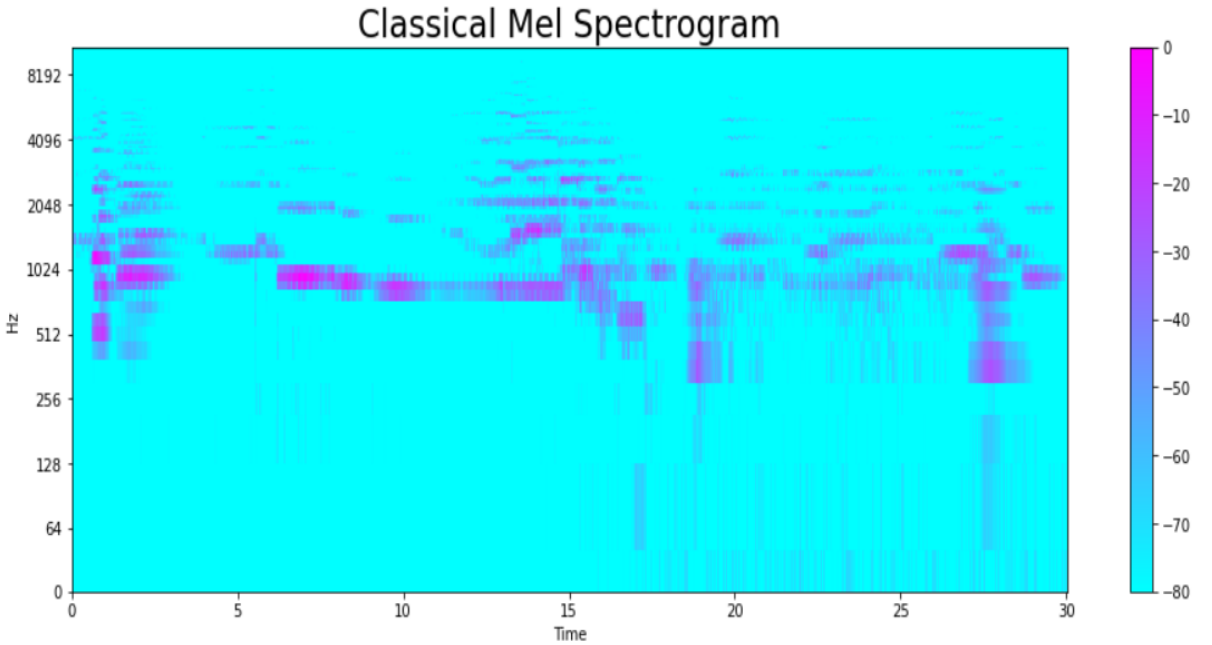


Figure 6.4. Classical Mel Spectrogram - classical.00036.wav

We then visualise the features of the audio file such as **Zero Crossing Rate, Harmonics and**

Perceptual, Tempo BPM, Spectral Centroid, EDA Spectral Rolloff and the Mel-Frequency Cepstral Coefficients.

- **Zero Crossing Rate:**

```
In [80]: # Total zero_crossings in our 1 song
         zero_crossings = librosa.zero_crossings(audio_file, pad=False)
         print(sum(zero_crossings))

         39232
```

  - **We get value as:** 39232

- **Harmonics and Perceptual**

```
In [81]: y_harm, y_perc = librosa.effects.hpss(audio_file)

         plt.figure(figsize = (16, 6))
         plt.plot(y_harm, color = '#A300F9');
         plt.plot(y_perc, color = '#FFB100');
```

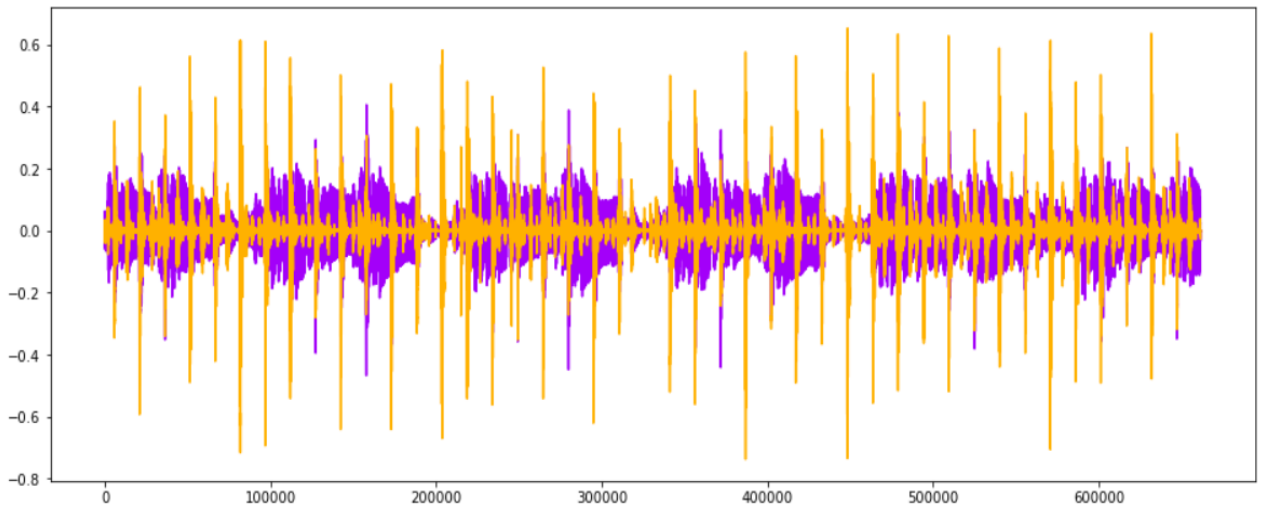


FIG. 6.5

- **Tempo BPM**

```
In [82]: tempo, _ = librosa.beat.beat_track(y, sr = sr)
         tempo

Out[82]: 107.666015625
```

  - **We get Tempo BPM as :** 107.666015625

- **Spectral Centroid**
  - **We get:**

In [83]:
```
# Calculate the Spectral Centroids
spectral_centroids = librosa.feature.spectral_centroid(audio_file, sr=sr)[0]

# Shape is a vector
print('Centroids:', spectral_centroids, '\n')
print('Shape of Spectral Centroids:', spectral_centroids.shape, '\n')

# Computing the time variable for visualization
frames = range(len(spectral_centroids))

# Converts frame counts to time (seconds)
t = librosa.frames_to_time(frames)

print('frames:', frames, '\n')
print('t:', t)

# Function that normalizes the Sound Data
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
```

```
Centroids: [1817.93364736 1953.08392985 2038.8113414  ...  766.50416352 1019.33192639
 1081.69931747]

Shape of Spectral Centroids: (1293,)

frames: range(0, 1293)

t: [0.00000000e+00 2.32199546e-02 4.64399093e-02 ... 2.99537415e+01
 2.99769615e+01 3.00001814e+01]
```


```
Centroids: [1817.93364736 1953.08392985 2038.8113414  ...
766.50416352 1019.33192639
1081.69931747]

Shape of Spectral Centroids: (1293,)

frames: range(0, 1293)

t: [0.00000000e+00 2.32199546e-02 4.64399093e-02 ... 2.9953
7415e+01
2.99769615e+01 3.00001814e+01]
```


- **And the plot as:**

In [84]:
```
#Plotting the Spectral Centroid along the waveform
plt.figure(figsize = (16, 6))
librosa.display.waveplot(audio_file, sr=sr, alpha=0.4, color = '#A300F9');
plt.plot(t, normalize(spectral_centroids), color='#FFB100');
```

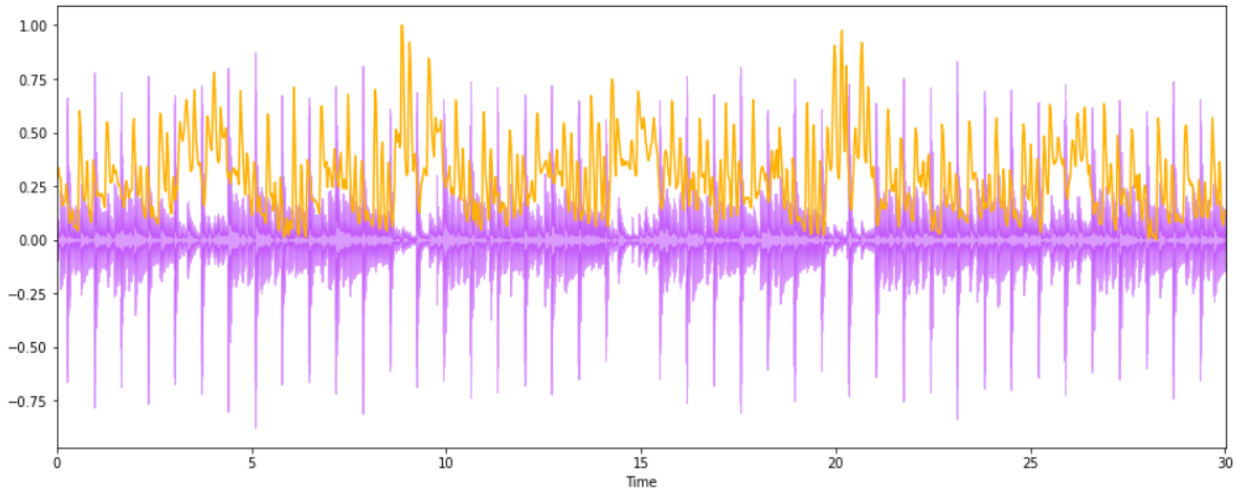


FIG. 6.6

- **EDA Spectral Rolloff:**

```
In [85]: # Spectral RollOff Vector
         spectral_rolloff = librosa.feature.spectral_rolloff(audio_file, sr=sr)[0]

         # The plot
         plt.figure(figsize = (16, 6))
         librosa.display.waveplot(audio_file, sr=sr, alpha=0.4, color = '#A300F9');
         plt.plot(t, normalize(spectral_rolloff), color='#FFB100');
```

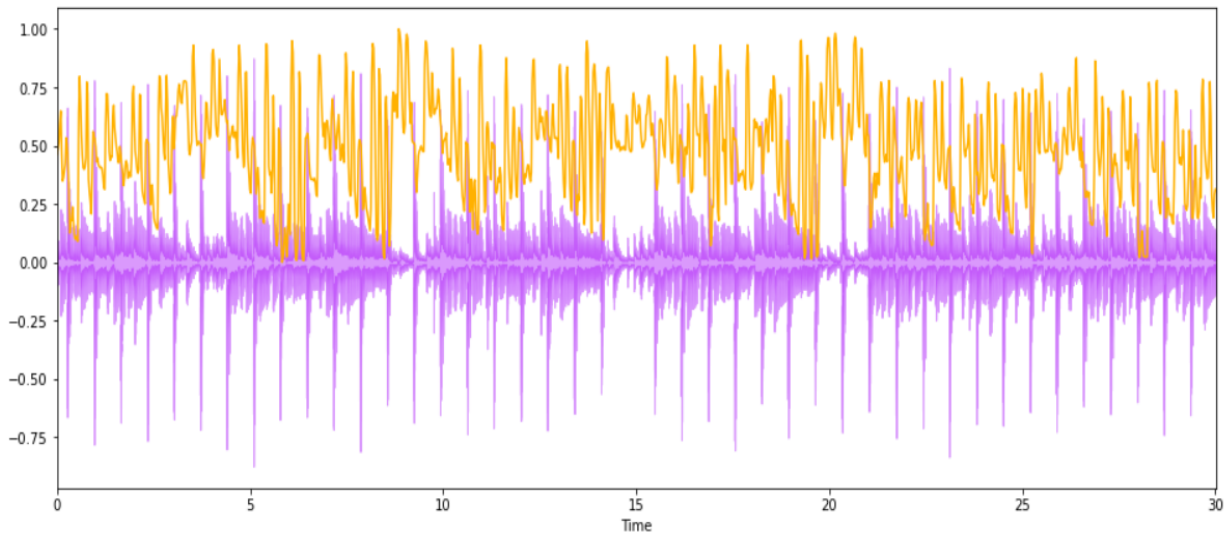


FIG. 6.7

- **Mel-Frequency Cepstral Coefficients.**

```
In [86]: mfccs = librosa.feature.mfcc(audio_file, sr=sr)
         print('mfccs shape:', mfccs.shape)

         #Displaying  the MFCCs:
         plt.figure(figsize = (16, 6))
         librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
```

```
mfccs shape: (20, 1293)
```

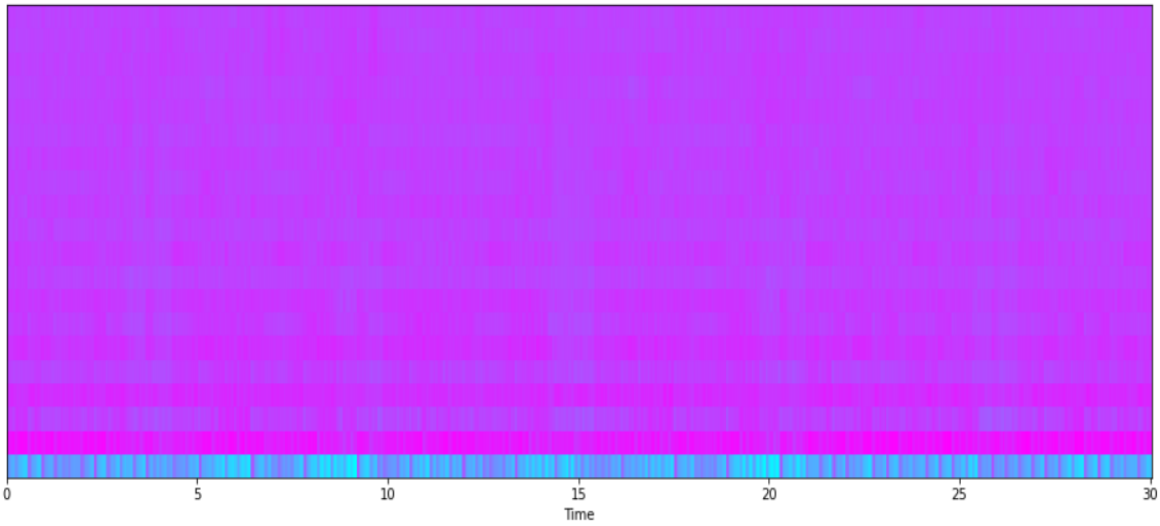


FIG. 6.8

➢ Chroma frequencies:

```
In [88]: # Increase or decrease hop_length to change how granular you want your data to be
         hop_length = 5000

         # Chromogram
         chromagram = librosa.feature.chroma_stft(audio_file, sr=sr, hop_length=hop_length)
         print('Chromogram shape:', chromagram.shape)

         plt.figure(figsize=(16, 6))
         librosa.display.specshow(chromagram, x_axis='time', y_axis='chroma', hop_length=hop_length, cmap='coolwarm');
```

```
Chromogram shape: (12, 133)
```

Chromogram shape: (12, 133)

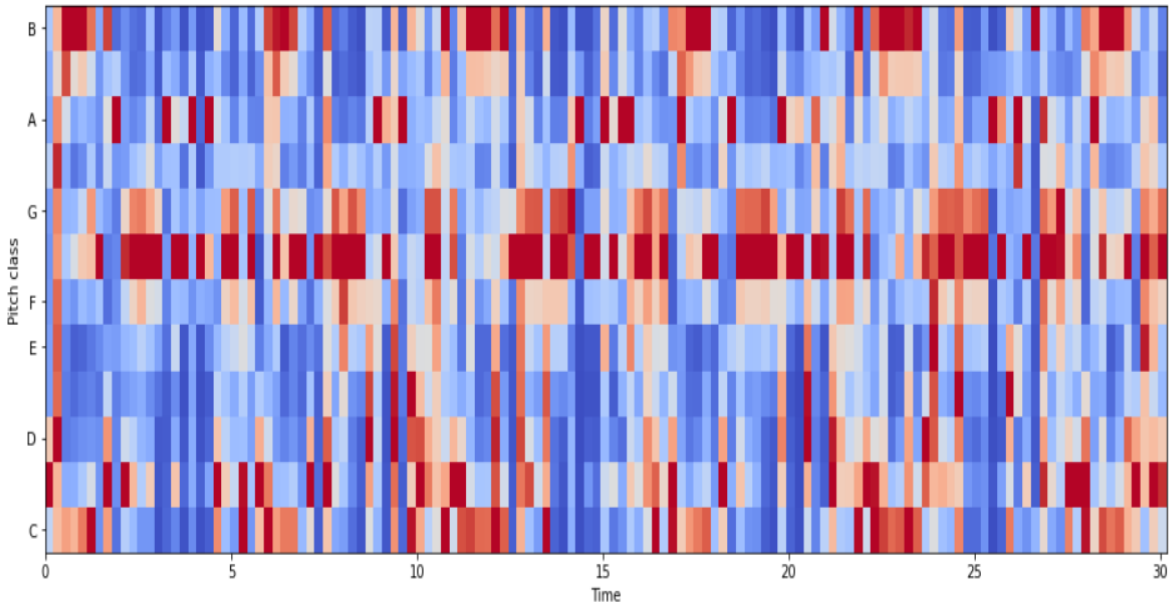


FIG. 6.9

➢ EDA:

```
In [89]: data = pd.read_csv(f'{general_path}/features_30_sec.csv')
         data.head()
```

Out[89]:

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	s
0	blues.00000.wav	661794	0.350088	0.088757	0.130228	0.002827	1784.165850	129774.064525	2002.449060	
1	blues.00001.wav	661794	0.340914	0.094980	0.095948	0.002373	1530.176679	375850.073649	2039.036516	
2	blues.00002.wav	661794	0.363637	0.085275	0.175570	0.002746	1552.811865	156467.643368	1747.702312	
3	blues.00003.wav	661794	0.404785	0.093999	0.141093	0.006346	1070.106615	184355.942417	1596.412872	
4	blues.00004.wav	661794	0.308526	0.087841	0.091529	0.002303	1835.004266	343399.939274	1748.172116	

5 rows × 60 columns

Out[89]:

ean	spectral_bandwidth_var	...	mfcc16_var	mfcc17_mean	mfcc17_var	mfcc18_mean	mfcc18_var	mfcc19_mean	mfcc19_var	mfcc20_mean	mfcc20_var	label
060	85882.761315	...	52.420910	-1.690215	36.524071	-0.408979	41.597103	-2.303523	55.062923	1.221291	46.936035	blues
516	213843.755497	...	55.356403	-0.731125	60.314529	0.295073	48.120598	-0.283518	51.106190	0.531217	45.786282	blues
312	76254.192257	...	40.598766	-7.729093	47.639427	-1.816407	52.382141	-3.439720	46.639660	-2.231258	30.573025	blues
872	166441.494769	...	44.427753	-3.319597	50.206673	0.636965	37.319130	-0.619121	37.259739	-3.407448	31.949339	blues
116	88445.209036	...	86.099236	-5.454034	75.269707	-0.916874	53.613918	-4.404827	62.910812	-11.703234	55.195160	blues

Then after that we draw the **box plot:**

```
In [91]: x = data[["label", "tempo"]]

         f, ax = plt.subplots(figsize=(16, 9));
         sns.boxplot(x = "label", y = "tempo", data = x, palette = 'husl');

         plt.title('BPM Boxplot for Genres', fontsize = 25)
         plt.xticks(fontsize = 14)
         plt.yticks(fontsize = 10);
         plt.xlabel("Genre", fontsize = 15)
         plt.ylabel("BPM", fontsize = 15)
         plt.savefig("BPM Boxplot.jpg")
```

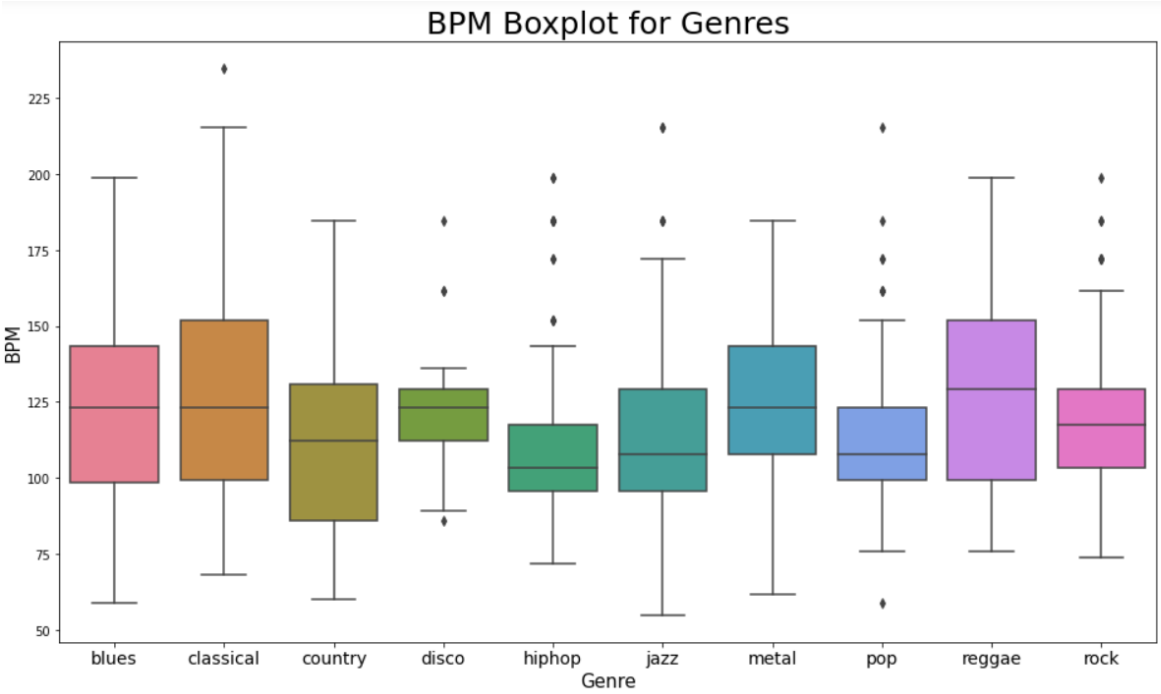


Figure 6.10. Boxplot depicting distribution of Genres

We perform Principal Component Analysis (PCA) in order to visualise the possible groups of genres.

```
In [93]: plt.figure(figsize = (16, 9))
sns.scatterplot(x = "principal component 1", y = "principal component 2", data = finalDf, hue = "label", alpha = 0.7,
                s = 100);

plt.title('PCA on Genres', fontsize = 25)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 10);
plt.xlabel("Principal Component 1", fontsize = 15)
plt.ylabel("Principal Component 2", fontsize = 15)
plt.savefig("PCA Scattert.jpg")
```

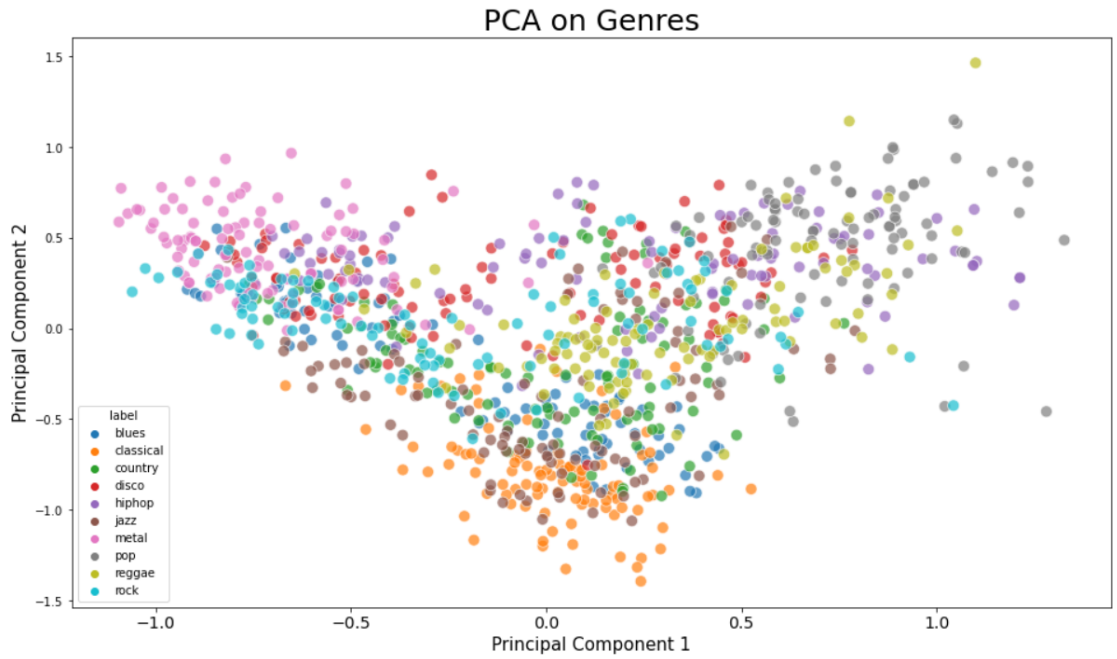


Figure 6.11. Scatter plot depicting PCA on Genres

B. Music Genre Classification:

We read the features_3_sec.csv file to build a classifier that accurately predicts the genre for any audio file input. We create the target and feature variables, normalise the data and then split it into 70% to 30% ratio as training: test data. On executing the algorithms and assessing the accuracy, we found that **XGBooster was the best** performing model.

In [101]:
```
nb = GaussianNB()
model_assess(nb, "Naive Bayes")

sgd = SGDClassifier(max_iter=5000, random_state=0)
model_assess(sgd, "Stochastic Gradient Descent")

knn = KNeighborsClassifier(n_neighbors=19)
model_assess(knn, "KNN")

tree = DecisionTreeClassifier()
model_assess(tree, "Decission trees")

rforest = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=0)
model_assess(rforest, "Random Forest")

svm = SVC(decision_function_shape="ovo")
model_assess(svm, "Support Vector Machine")

lg = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial')
model_assess(lg, "Logistic Regression")

nn = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5000, 10), random_state=1)
model_assess(nn, "Neural Nets")

xgb = XGBClassifier(n_estimators=1000, learning_rate=0.05)
model_assess(xgb, "Cross Gradient Booster")

xgbrf = XGBRFClassifier(objective= 'multi:softmax')
model_assess(xgbrf, "Cross Gradient Booster (Random Forest)")
```

```
Accuracy Naive Bayes : 0.51952

Accuracy Stochastic Gradient Descent : 0.65532

Accuracy KNN : 0.80581

Accuracy Decission trees : 0.64331

Accuracy Random Forest : 0.81415

Accuracy Support Vector Machine : 0.75409

Accuracy Logistic Regression : 0.6977

Accuracy Neural Nets : 0.68135

Accuracy Cross Gradient Booster : 0.90224

Accuracy Cross Gradient Booster (Random Forest) : 0.74575
```

Figure 6.12. Data shows the accuracy of different model

We find XGBooster to be the model with highest accuracy - 90% (approx)

In [102]:
```
# Final model
xgb = XGBClassifier(n_estimators=1000, learning_rate=0.05)
xgb.fit(X_train, y_train)


preds = xgb.predict(X_test)

print('Accuracy', ':', round(accuracy_score(y_test, preds), 5), '\n')

# Confusion Matrix : is a table that is often used to describe the performance of a classification model
confusion_matr = confusion_matrix(y_test, preds) #normalize = 'true'
plt.figure(figsize = (16, 9))
sns.heatmap(confusion_matr, cmap="Blues", annot=True,
            xticklabels = ["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "pop", "reggae", "rock"],
           yticklabels=["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "pop", "reggae", "rock"]);
plt.savefig("conf matrix")
#Confusion Matrix for XGBooster Algorithm
```

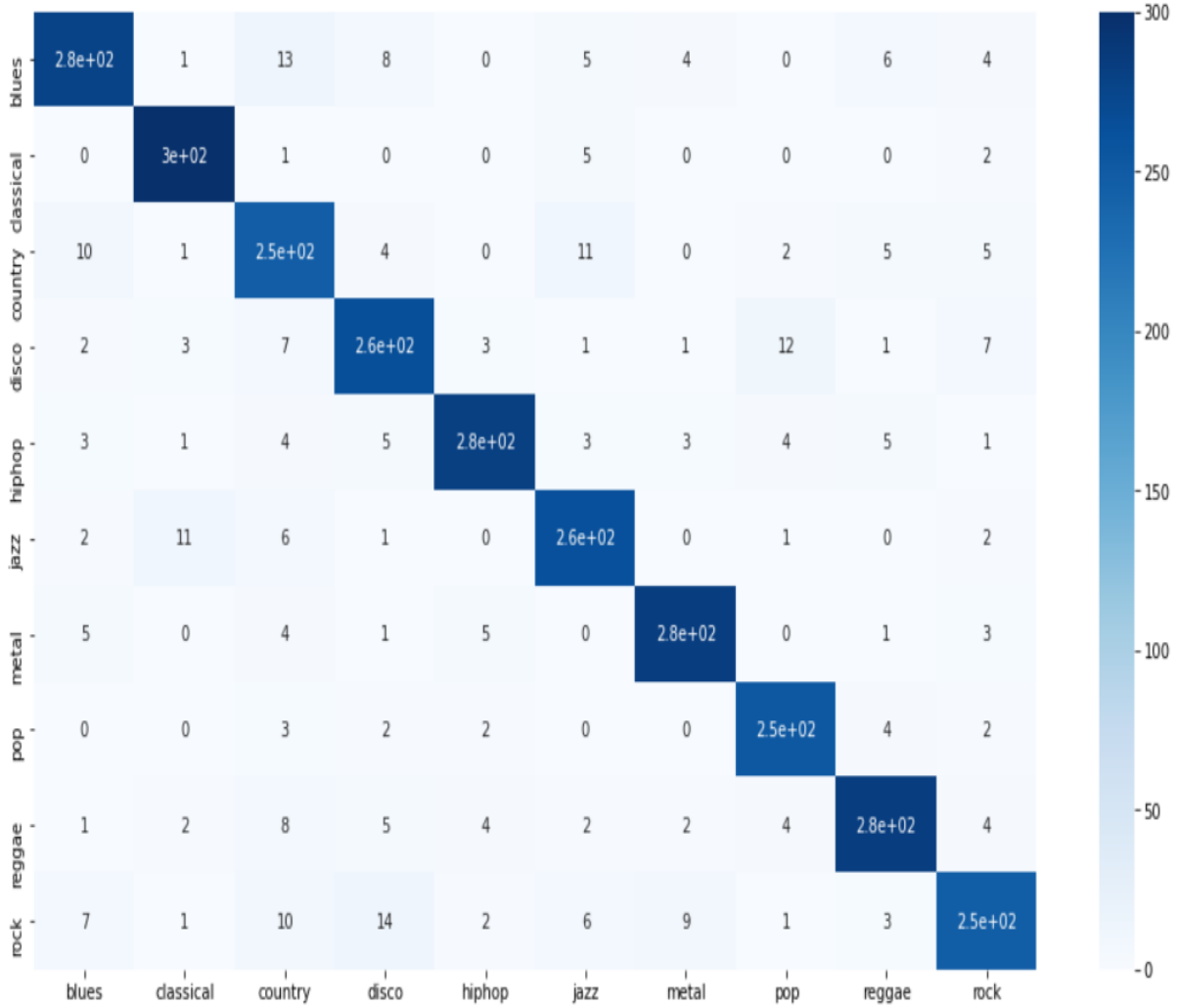


Figure 6.13. Confusion Matrix for XGBooster Algorithm

In [103]:
```
import eli5
from eli5.sklearn import PermutationImportance

# function to probe which features are most predictive
# Feature importance along with their weights

perm = PermutationImportance(estimator=xgb, random_state=1)
perm.fit(X_test, y_test)

eli5.show_weights(estimator=perm, feature_names = X_test.columns.tolist())
```

Out[103]:

Weight	Feature
0.1205 ± 0.0095	perceptr_var
0.0416 ± 0.0031	perceptr_mean
0.0390 ± 0.0049	mfcc4_mean
0.0345 ± 0.0044	chroma_stft_mean
0.0339 ± 0.0062	harmony_mean
0.0280 ± 0.0065	harmony_var
0.0228 ± 0.0049	mfcc9_mean
0.0208 ± 0.0049	mfcc6_mean
0.0181 ± 0.0024	rms_var
0.0174 ± 0.0026	mfcc3_mean
0.0148 ± 0.0031	spectral_bandwidth_mean
0.0147 ± 0.0056	mfcc11_mean
0.0137 ± 0.0046	tempo
0.0116 ± 0.0036	chroma_stft_var
0.0113 ± 0.0026	mfcc7_mean
0.0109 ± 0.0038	mfcc1_var
0.0101 ± 0.0029	mfcc3_var
0.0089 ± 0.0057	mfcc8_mean
0.0089 ± 0.0020	mfcc5_mean
0.0072 ± 0.0038	mfcc18_mean

Figure 6.14. Feature importance along with their weights-Screenshot

Table6.1. Top 5 Most Important Features that determine the Music Genre

Sl no.	Weight	Feature
1.	0.1205 ± 0.0095	Perceptron Variance
2.	0.0416 ± 0.0031	Perceptron Mean
3.	0.0390 ± 0.0049	MFCC Mean
4.	0.0345 ± 0.0044	Chroma Mean
5.	0.0339 ± 0.0062	Harmony Mean

C. Recommender Systems

Firstly, we scale the input data**: features_30_sec.csv file**. Using **cosine_similarity** library, we find the best similar matches ranked in descending order for any given vector input. It calculates the similarity of two audio files, pairwise and generates a 1000 x 1000 matrix where every cell depicts the similarity of the corresponding audio files. We use the **features_30_sec.csv** file to predict the same. We write a function named **find_similar_songs** that takes the name of a song and returns the top five best matches for the input.
cosine similarity: it shows the number that represent the similarity between the songs if songs are exactly identical, we get cosine as 1 else we get its value as less than one.

Here we are using **cosine similarity** and the data that we get are:

In [105]:
```
# Cosine similarity
similarity = cosine_similarity(data_scaled)
print("Similarity shape:", similarity.shape)

# Convert into a dataframe and then set the row index and column names as labels
sim_df_labels = pd.DataFrame(similarity)
sim_df_names = sim_df_labels.set_index(labels.index)
sim_df_names.columns = labels.index

sim_df_names.head()
```

```
Similarity shape: (1000, 1000)
```

Out[105]:

filename	blues.00000.wav	blues.00001.wav	blues.00002.wav	blues.00003.wav	blues.00004.wav	blues.00005.wav	blues.00006.wav	blues.00007.wav	blues
filename									
blues.00000.wav	1.000000	0.049231	0.589618	0.284862	0.025561	-0.346688	-0.219483	-0.167626	
blues.00001.wav	0.049231	1.000000	-0.096834	0.520903	0.080749	0.307856	0.318286	0.415258	
blues.00002.wav	0.589618	-0.096834	1.000000	0.210411	0.400266	-0.082019	-0.028061	0.104446	
blues.00003.wav	0.284862	0.520903	0.210411	1.000000	0.126437	0.134796	0.300746	0.324566	
blues.00004.wav	0.025561	0.080749	0.400266	0.126437	1.000000	0.556066	0.482195	0.623455	

5 rows × 1000 columns

Out[105]:

blues.00008.wav	blues.00009.wav	...	rock.00090.wav	rock.00091.wav	rock.00092.wav	rock.00093.wav	rock.00094.wav	rock.00095.wav	rock.00096.wav	rock.00097
0.641877	-0.097889	...	-0.082829	0.546169	0.578558	0.662590	0.571629	0.610942	0.640835	0.49
0.120649	0.404168	...	-0.098111	-0.325126	-0.370792	-0.191698	-0.330834	-0.077301	-0.222119	-0.30
0.468113	-0.132532	...	-0.032408	0.561074	0.590779	0.583293	0.514537	0.495707	0.566837	0.58
0.352758	0.295184	...	-0.320107	-0.206516	-0.151132	0.041986	-0.172515	-0.000287	0.020515	-0.10
0.029703	0.471657	...	0.087605	0.017366	0.138035	0.104684	-0.034594	0.063454	0.063546	0.17

Out[105]:

rock.00090.wav	rock.00091.wav	rock.00092.wav	rock.00093.wav	rock.00094.wav	rock.00095.wav	rock.00096.wav	rock.00097.wav	rock.00098.wav	rock.00099.wav
-0.082829	0.546169	0.578558	0.662590	0.571629	0.610942	0.640835	0.496294	0.284958	0.304098
-0.098111	-0.325126	-0.370792	-0.191698	-0.330834	-0.077301	-0.222119	-0.302573	0.499562	0.311723
-0.032408	0.561074	0.590779	0.583293	0.514537	0.495707	0.566837	0.589983	0.216378	0.321069
-0.320107	-0.206516	-0.151132	0.041986	-0.172515	-0.000287	0.020515	-0.107821	0.502279	0.183210
0.087605	0.017366	0.138035	0.104684	-0.034594	0.063454	0.063546	0.172944	0.153192	0.061785

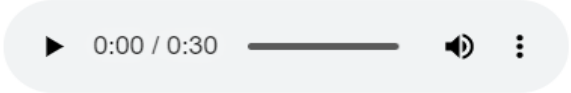
On searching for songs similar to **pop.00019.wav**, we get the following list and also we are displaying the sample song **pop.00019.wav**:

In [106]:
```
def find_similar_songs(name):
    # Find songs most similar to another song
    series = sim_df_names[name].sort_values(ascending = False)

    # Remove cosine similarity == 1 (songs will always have the best match with themselves)
    series = series.drop(name)

    # Display the 5 top matches
    print("\n*******\nSimilar songs to ", name)
    print(series.head(5))
```

In [107]:
```
# pop.00019 - Britney Spears
find_similar_songs('pop.00019.wav')

ipd.Audio(f'{general_path}/genres_original/pop/pop.00019.wav')
```

```
*******
Similar songs to  pop.00019.wav
filename
pop.00023.wav    0.862836
pop.00034.wav    0.860499
pop.00078.wav    0.829135
pop.00088.wav    0.824456
pop.00091.wav    0.802269
Name: pop.00019.wav, dtype: float64
```

Out[107]:

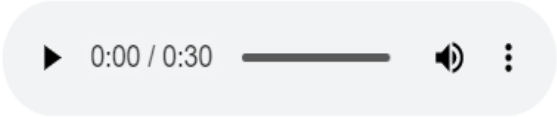


On searching for songs similar to **metal.00002.wav**, we get the following list and also, we are displaying the sample song **metal.00002.wav**:
Now since the accuracy is 90% hence, we are getting some of the rock songs with the metal songs. But it can be improved by increasing the training time.

In [108]:
```
# metal.00002 - Iron Maiden "Flight of Icarus"
find_similar_songs('metal.00002.wav')

ipd.Audio(f'{general_path}/genres_original/metal/metal.00002.wav')
```

```
*******
Similar songs to  metal.00002.wav
filename
metal.00028.wav    0.904367
metal.00059.wav    0.896096
rock.00018.wav     0.891910
rock.00017.wav     0.886526
rock.00016.wav     0.867508
Name: metal.00002.wav, dtype: float64
```

Out[108]:



7.RESULTS AND DISCUSSION

Table 7.1. RESULTS

Algorithm	Accuracy	Efficiency	F-score	Precision
SVM	0.75409	0.75410	0.75210	0.75209
K-Nearest Neighbours	0.80581	0.80567	0.79432	0.79338
Convolution Neural Networks	0.67734	0.67723	0.66478	0.66474
Stochastic Gradient Descent	0.65532	0.65527	0.64213	0.64205
Logistic Regression	0.69970	0.69966	0.68778	0.68764
Recurrent Neural Networks	0.69012	0.69034	0.68982	0.68991
Random Forest	0.81415	0.81427	0.80126	0.80113
Decision Trees	0.64631	0.64660	0.63990	0.63984
K-Means Clustering	0.72146	0.72123	0.71291	0.71278
Cross Gradient Booster	0.90224	0.90222	0.90121	0.90125
Cross Gradient Booster (Random Forest)	0.74875	0.74863	0.73245	0.73249
Naive Bayes	0.51952	0.51940	0.50349	0.50412

Accuracy, efficiency, f-score and precision are few of the most important parameters that are used to evaluate the performance of the algorithms used and decide which one to finally incorporate in the project. We can deduct from the above table that the algorithm with the highest accuracy, efficiency, f-score and precision is the Cross Gradient Booster Algorithm and the one with the lowest is Naive Bayes Algorithm. Hence it is ideal to choose the **Cross Gradient Booster** for building our model

7.2. DISCUSSIONS:

Music Genre classification has wide areas of applications. This is the basis for many applications that can be made using this classifier. Genre of the music is a very useful knowledge that can be used for a wide variety of manipulations as it provides some great insight.

1. Music Genre Classification can be used to build a music Recommender System. A music Recommender system understands your choice of music based on your selections and then recommends suitable music that you can listen to. Being able to classify the type of music that the user listens will enable the model to suggest even more music in the same category.

2. Music Genre Classification can be used in Sentimental analysis systems. Based on the genre of music the user is listening to or would like to listen to, the system can understand the genre of music and then predict the mood of the person. For example - Person listening to pop music can be happy.

3. Music Genre classification can be used to automatically make playlists for songs. This is a system that many famous applications use in order to make playlists and suggest them to users.

4. Music Genre classification enables systems to manage, classify and store the large amount of data we have in audio format. Genre classification will help systems to automatically classify them making everything very organised.

5. Good classification and categorisation of data will also help making retrieval systems much more efficient. Retrieval system is

something that enables you to retrieve based on some provided data like the lyrics, category, music itself etc. Thus, music genre classification will also play a major role in this field.

Thus, we can see that music genre classification is the basis for a wide variety of problems and it can provide very useful knowledge about any music.

8. CONCLUSION

- As per the accuracy results, we find that **XGBooster has the highest accuracy of 90%**, and then Random Forests and KNN have an accuracy of 81% and 80% respectively.

- We build our final model using the XGBooster algorithm and perform feature importance.

- We find *perceptr_var*, *perceptr_mean* and *mfcc4_mean* to be features that have the highest weights of importance.

- Finally, using the cosine_similarity library we find the best similar matches to the given input audio.

- Our project is having is able to distinguish the songs on the basis of their genre like: blue, classical, pop, metal etc. and also it is having a decent amount of accuracy. This type of AI implementations are used in many apps and also many companies like: sportify, gaana etc, that helps them to give suggestions to the peoples requirement and interest.

- In general, it is very difficult to say what a good enough prediction accuracy for an automatic system would be since even humans are not perfect at classifying songs. However, it could be useful for a system where human confirmation is used, as it would decrease the workload. Also, this project helped us to know more about AI and also about various concepts of ML and AI such as: XGBoose, KNN, Naïve Byes etc.

9. REFERENCES

[1]. Bahuleyan, Hareesh. "Music genre classification using machine learning techniques." *arXiv preprint arXiv:1804.01149* (2018).
[2]. Dong, Mingwen. "Convolutional neural network achieves human-level accuracy in music genre classification." *arXiv preprint arXiv:1802.09697* (2018).
[3]. Tang, Chun Pui, et al. "Music genre classification using a hierarchical long short term memory (LSTM) model." *Third International Workshop on Pattern Recognition*. Vol. 10828. International Society for Optics and Photonics, 2018.
[4]. Bahuleyan, Hareesh. "Music genre classification using machine learning techniques." *arXiv preprint arXiv:1804.01149* (2018).
[5]. Tang, Chun Pui, et al. "Music genre classification using a hierarchical long short-term memory (LSTM) model." *Third International Workshop on Pattern Recognition*. Vol. 10828. International Society for Optics and Photonics, 2018