MUSIC GENRE CLASSIFICATION

BECE309L – Artificial Intelligence and Machine Learning

By

Ashrit Saha(21BEC1257)

Biswajyoti Dutta(21BLC1609)

SUBMITTED TO

Dr. Rabindra

Kumar Singh

Associate Professor Senior, VIT Chennai

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF ELECTRONICS

ENGINEERING VELLORE INSTITUE OF

TECHNOLOGY CHENNAI – 600127

April 2024

ABSTRACT

Music genre classification is a crucial problem in the audio signal processing space with applications varying from music recommendation to content organization in digital libraries. In the past couple of years, deep learning approaches, specifically Convolutional Neural Networks (CNNs), have demonstrated remarkable performance in automatically classifying music based on different genres. This paper considers a CNN-based technique for music genre classification that provides 90% accuracy.

The proposed model exploits the hierarchical and spatial information that is inherent in the spectrogram representation of audio signals. The network structure of multiple layers of convolutions and pooling enables the model to detect both low-level and high-level features for identifying different genres of music. Apart from that, dropout regularization is another technique that prevents overfitting and improves generalization performance.

The effectiveness of the proposed approach was evaluated by performing prolonged experiments on a large-scale music dataset which included different genres. The experiments show that the proposed CNN-based model always exceeds the traditional machine learning algorithms and older deep learning architectures in respect of classification precision. In addition, the model's performance with different audio quality levels and recording conditions is tested to demonstrate its real-world application potential.

Overall, this study extends the boundaries of music genre classification approaches through the demonstration of ever increasing accuracy when employing CNNs. The proposed model presents a solution that is reliable and scalable, and it automates the categorization of music content, consequently making the organization and the search of digital music collections easy.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Dr**.Rabindra Kumar Singh,** Associate Professor, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Susan Elias,** Dean of the School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our Head of the Department **Dr.Mohanaprasad. K for** his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Biswajyoti Dutta
(21BLC1609)

Ashrit Saha
(21BEC1257)

CHAPTER 1

INTRODUCTION

1.1 PURPOSE

- Automated music organization for efficient browsing and discovery.
- Improved music recommendation systems based on accurate genre classification.
- Advancement of research in deep learning for audio signal processing.
- Practical applications in music streaming platforms and digital libraries.
- Scalable and efficient solution for managing large music datasets.
- Enhanced user experience through personalized music content.
- Educational resource for machine learning and music analysis.

1.2 SCOPE

- **Exploration of Different CNN Architectures:** Investigate various CNN architectures and configurations to optimize genre classification performance, such as experimenting with different numbers of convolutional layers, filter sizes, and pooling strategies.

- **Feature Engineering and Representation Learning:** Explore advanced feature engineering techniques or investigate the potential of learning hierarchical representations directly from raw audio data using deep learning models like WaveNet or Transformers.

- **Multi-label Genre Classification:** Extend the project to handle cases where songs belong to multiple genres simultaneously, addressing the challenge of multi-label classification and exploring techniques like binary relevance or classifier chains.

- **Real-time Classification Systems:** Develop real-time music genre classification systems suitable for applications like live music streaming or DJ software, focusing on low-latency processing and efficient model deployment.

- **Cross-domain Generalization:** Assess the model's ability to generalize across different music datasets collected from diverse sources or domains, evaluating its robustness and adaptability to variations in recording quality, instrumentation, and cultural influences.

- **User Interface Integration:** Integrate the genre classification model into user-facing applications with intuitive interfaces, allowing users to interactively explore music collections, receive genre recommendations, and provide feedback to improve classification accuracy.

- **Integration with Music Content Analysis:** Extend the scope to include additional music content analysis tasks such as mood detection, tempo estimation, or instrument recognition, creating comprehensive systems for understanding and organizing music content.

- **Evaluation of Transfer Learning Techniques:** Investigate the effectiveness of transfer learning approaches, such as fine-tuning pre-trained models on music genre classification tasks, to leverage knowledge from large-scale datasets and improve performance on smaller, domain-specific datasets.

- **Collaborative Filtering and Social Recommendations:** Explore collaborative filtering techniques or incorporate social interaction data to enhance music recommendation systems, leveraging user preferences and behavior to provide personalized genre suggestions.

- **Deployment in Edge Computing Environments:** Optimize the model for deployment in edge computing environments or resource-constrained devices, enabling offline genre classification on mobile devices, smart speakers, or IoT devices without relying on cloud-based services.

CHAPTER 2

DESIGN AND IMPLEMENTATION

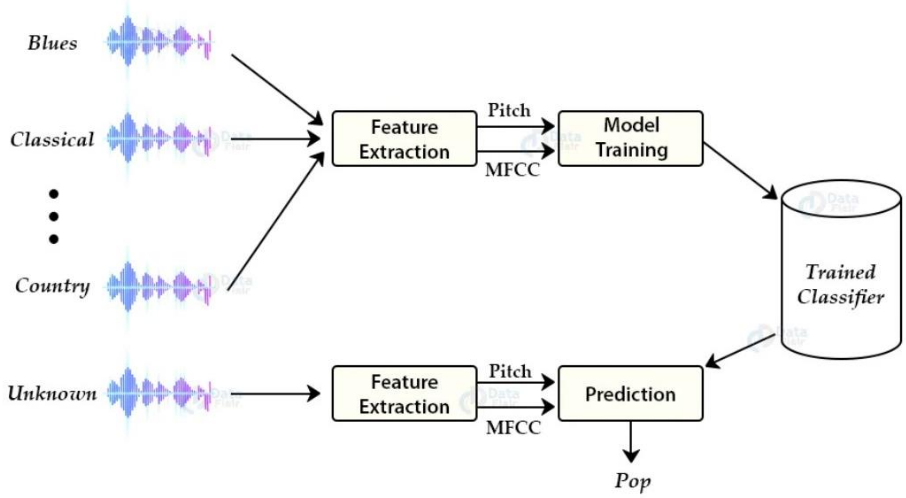
2.1 INTRODUCTION

The identification of music genre is fundamental to the audio signal processing field and can reveal deep down the nature of musical diversity. In this report, we embark on an exploration of music genre classification, leveraging two prominent models: Two main techniques used are Convolutional Neural Networks (CNN) and Mel-Frequency Cepstral Coefficients (MFCC) and both of them are implemented in Python programming language.

The key point of our investigation is the usage of GTZAN dataset, the broadly accepted standard for classification of music genre which is well known in the area of music genre classification. This exhaustively generated dataset represents a spectrum of audio samples grouped under different genres, offering a wealth of analysis for our work in this field.

By using the two-pronged approach of the CNN and MFCC models, we seek to extricate the complex mechanisms embedded within music samples in which patterns and features distinguishable from different genres become apparent. Through the computer power of the models, our principal goal is to explore the boundaries of the automatic music genre classification which would result in more innovation and advancements into the future audio analysis.

In this report, our approach is meticulously described, the main conclusions are precised, and the gained conclusions are interpreted based on the carefully designated genre classification investigation. Through explanation of fulcrum of computational methods in understanding the multilayered fabric of musical genres, we attempt to add value in the general discussion that appears on audio signal processing.



2.2 DESIGN APPROACH

- **Data Preparation and Library Import:** Import dataset GTZAN and instantiate necessary library objects from NumPy, Pandas, Matplotlib, TensorFlow, Seaborn, and Librosa for data manipulation and analysis.

- **Genre Classification and Waveform Extraction:** Increasing awareness of ethical principles has been a significant factor in the development of sustainable supply chains. - Identify the 10 music genres: the mixtures of blues, metal, jazz, classical, hip hop, country, pop, reggae, and rock. - Extract the waveform data for each genre so as to build the basis concept of the content.

- **Mean Variables Heatmap Generation:** In conclusion, the issue of water scarcity is a complex and multifaceted problem that often requires coordinated and sustainable solutions. - Calculate the "heat map" means for each genre to depict the distribution of features among several genres.

- **Feature Extraction and Spectrogram Analysis:** Thus, enforcing stricter emission controls or creating eco-zones in cities is crucial for reducing air pollution in urban areas. - Create spectrograms, spectral roles, chroma representations, and zero-crossing rates independently for each genre collectively to store various audio information.

- **Audio Playback and Visualization:** On the one hand, the psychological effects of long-term isolation in space cannot be minimized. - Integrate the NumPy and Librosa to load each sample of each genre for further aural inspection and heuristic assessment.

- **Label Encoding for Model Input:** The independence of this new nation was therefore not a straightforward process that happened overnight. - The LabelEncoder class from the LabelEncode library can be applied to genre labels in order to convert categorical features into numerical representations to be used for model training.

- **Model Import and Training with Keras:** In the multifaceted world of modern society, individuals often become immersed in the constant stream of external stimuli, which can lead to feelings of alienation and disconnection from themselves. - Since Keras is the preferred library, we will proceed to construct a neural network model for music genre classification. - Apply the model to the relevant dataset with the TensorFlow backend and adjust the parameters to enhance the performance of the model.

- **Accuracy Evaluation with Keras Model:** Attic Insulation ** As a homeowner, air leakage is a primary concern that causes the inefficient use of energy and heat loss. - Apply the Python and test the trained Keras' classification accuracy, which can indicate its ability to identify genre correctly.

- **Data Splitting for Training and Testing:** From its effects on social interaction to its impact on social cohesion, social media undeniably revolutionizes the way we engage in our communities. - Partition the data into a training and validation set where X is a vector of the features and y is the associated genre label.

- **Feature Extraction with Mel Frequency Cepstral Coefficients (MFCCs):** Building policy reforms and regulations that incorporate sustainability practices and incentivize renewable energy sources is vital for modern society. - Apply the Mel-frequency cepstral coefficients (MFCCs) for feature extraction to get the essential coefficients. - Take advantage of Librosa's feature extractor functionality to convert raw sound signals into the features vectors of the machine learning devices.

- **Model Summary and Analysis:** While a small town setting may limit accessibility to certain cultural resources, we strive to leverage local partnerships and engage the community in meaningful cultural experiences. - Analyze the trained model's summary, which contains details on its structure such as the architecture, layer characteristics, and parameter counts, to understand the model's complexity and potential capacity.

2.3 Advantages of this method:

- **Automated Music Organization:** The project does this via automatic identification of music into certain genres, thus, helping one to effectively classify and navigate large music collections. Users are able to find and enjoy their preferred songs genres hence making the overall listening experience more interesting.

- **Enhanced Music Recommendation Systems:** The correct genre classification is not only the vital factor in music recommendation systems but also it gives personalized and useful recommendations for users. Through understanding the key features of music genres, the system could propose songs that tend to the tastes of users as well as keep them happy and involved. Insight into

- **Genre Characteristics:** Through feature extraction and analysis the project is able to offer the unique properties which music genres possess. These idiosyncrasies of the genre are essential for music research, content curation, and music production. They allow the stakeholders to take well-informed decisions in this industry.

- **Machine Learning Skill Development:** The project provides an avenue to improve the competences of practitioners as regards machine learning, data pre-processing, feature engineering, and model evaluation. Through hands-on work on a practical example such as music genre classification, students can not only get an idea of the machine learning process but also strengthen their knowledge about ML algorithms and frameworks.

- **Customizable and Scalable Solutions:** The project's modular design provides the opportunity for customization and flexibility to incorporate different types of data, genres, and user choices. Organizations can adapt the project project to their requirements as they see fit, be it building music recommendations for streaming platforms or content tagging based on genres for digital libraries.

- **Integration with Existing Platforms:** The classification model developed in this project can be easily integrated into the already existing music platforms, media players or digital libraries. These platforms will be able to provide value to their users by incorporating genre classification functionality which entails features like genre-based playlists, genre filtering and genre specific content recommendations that enhance the user experience. Contribution to

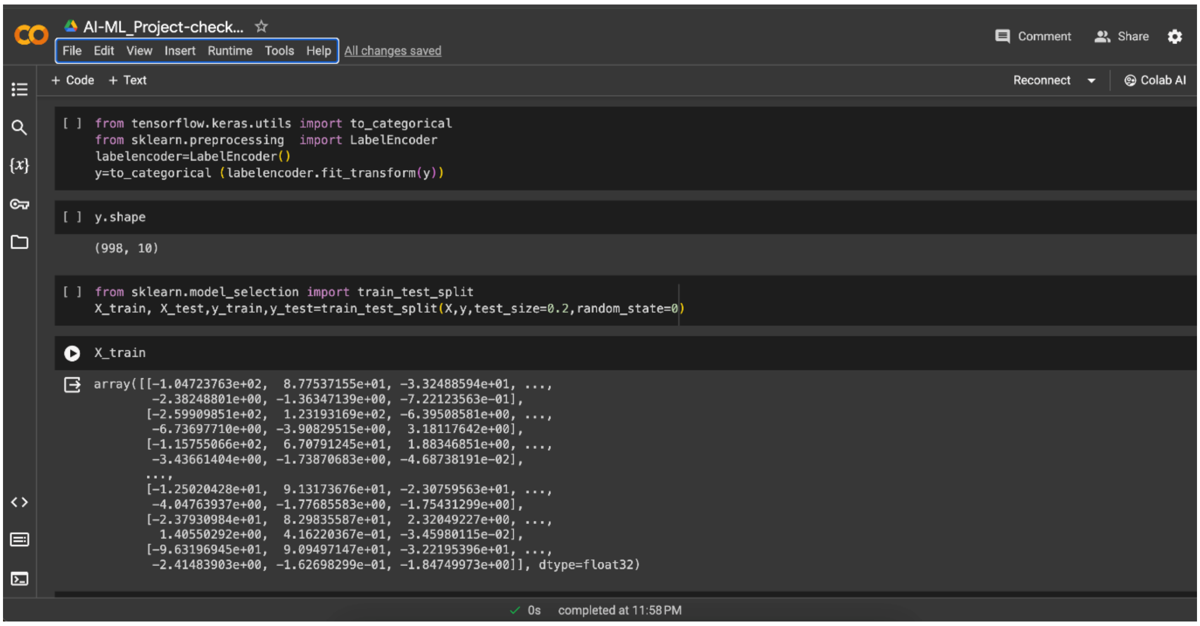
- **Research and Innovation:** The project is part of an ongoing research in the field of audio signal processing and machine learning. Through pursuing new approaches for music genre classification and sharing the outcomes and experiments with research community, the project motivates innovation and advances the level of understanding in the domain.

2.4 OVERVEIW OF SOFTWARE

Google Colab and Jupyter Notebook, both utilizing Python as the primary coding language, form a robust and user-friendly software environment for data analysis, machine learning, and collaborative coding. Here's an overview of each component:

Software Used – Python

jupyter AI-ML_Project Last Checkpoint: 12 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
2000
0
0 5 10 15 20 25 30
Time
−10
−20
−30
[92]:
# Spectral Roll-Off
from sklearn.preprocessing import normalize
spectral_rolloff = librosa.feature.spectral_rolloff(y=data + 0.01, sr=sr)[0]
plt.figure(figsize=(12,4))
librosa.display.waveshow(data,sr=sr,alpha=0.4,color="#004953")
[92]: <librosa.display.AdaptiveWaveplot at 0x14ace3fd940>
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



AI-ML_Project-check...
File Edit View Insert Runtime Tools Help
All changes saved
Comment
Share
+ Code
+ Text
Reconnect
Colab AI
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical (labelencoder.fit_transform(y))
y.shape
(998, 10)
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
X_train
array([[-1.04723763e+02, 8.77537155e+01, -3.32488594e+01, ...,
-2.38248801e+00, -1.36347139e+00, -7.22123563e-01],
[-2.59909851e+02, 1.23193169e+02, -6.39508581e+00, ...,
-6.73697710e+00, -3.90829515e+00, 3.18117642e+00],
[-1.15755066e+02, 6.70791245e+01, 1.88346851e+00, ...,
-3.43661404e+00, -1.73870683e+00, -4.68738191e-02],
...,
[-1.25020428e+01, 9.13173676e+01, -2.30759563e+01, ...,
-4.04763937e+00, -1.77685583e+00, -1.75431299e+00],
[-2.37930984e+01, 8.29835587e+01, 2.32049227e+00, ...,
1.40550292e+00, 4.16220367e-01, -3.45980115e-02],
[-9.63196945e+01, 9.09497147e+01, -3.22195396e+01, ...,
-2.41483903e+00, -1.62698229e-01, -1.84749973e+00]], dtype=float32)
0s completed at 11:58 PM

CHAPTER 3

RESULTS AND ANALYSIS TESTING

3.1 WORK DONE

CODE:-

```
    # IMPORTING THE LIBRARIES
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import librosa.display  # this library is mainly use to deal with sound
datset (audio)

# INSTALLING THE LIBRARIES
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install seaborn
!pip install librosa
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\users\hp\appdata\roaming\python\python312\site-packages (1.26.4)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in c:\users\hp\appdata\roaming\python\python312\site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\hp\appdata\roaming\python\python312\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hp\appdata\roaming\python\python312\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\roaming\python\python312\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\appdata\roaming\python\python312\site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\roaming\python\python312\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: matplotlib in c:\users\hp\appdata\roaming\python\python312\site-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.21 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=8 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\appdata\roaming\python\python312\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\roaming\python\python312\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: seaborn in c:\users\hp\appdata\roaming\python\python312\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\hp\appdata\roaming\python\python312\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\users\hp\appdata\roaming\python\python312\site-packages (from seaborn) (2.2.2)
...
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\hp\appdata\roaming\python\python312\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hp\appdata\roaming\python\python312\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\hp\appdata\roaming\python\python312\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\hp\appdata\roaming\python\python312\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2024.2.2)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
# READING THE DATASET ANN DISPLAYING THE FIRST 10 VALUES
music_data = pd.read_csv('file.csv')
music_data.head(10)
music_data = pd.read_csv('file.csv')
music_data.tail(5)
pip install setuptools
```

```
Defaulting to user installation because normal site-packages is not writeableNote: you may need to restart the kernel to use updated packages.

Requirement already satisfied: setuptools in c:\users\hp\appdata\roaming\python\python312\site-packages (69.3.0)
```

```
# ABOUT THE DATASET
```

```
music_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 60 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   filename                 1000 non-null   object 
 1   length                   1000 non-null   int64  
 2   chroma_stft_mean         1000 non-null   float64
 3   chroma_stft_var          1000 non-null   float64
 4   rms_mean                 1000 non-null   float64
 5   rms_var                  1000 non-null   float64
 6   spectral_centroid_mean   1000 non-null   float64
 7   spectral_centroid_var    1000 non-null   float64
 8   spectral_bandwidth_mean  1000 non-null   float64
 9   spectral_bandwidth_var   1000 non-null   float64
 10  rolloff_mean             1000 non-null   float64
 11  rolloff_var              1000 non-null   float64
 12  zero_crossing_rate_mean  1000 non-null   float64
 13  zero_crossing_rate_var   1000 non-null   float64
 14  harmony_mean             1000 non-null   float64
 15  harmony_var              1000 non-null   float64
 16  perceptr_mean            1000 non-null   float64
 17  perceptr_var             1000 non-null   float64
 18  tempo                    1000 non-null   float64
 19  mfcc1_mean               1000 non-null   float64
...
 58  mfcc20_var               1000 non-null   float64
 59  label                    1000 non-null   object 
dtypes: float64(57), int64(1), object(2)
memory usage: 468.9+ KB
```

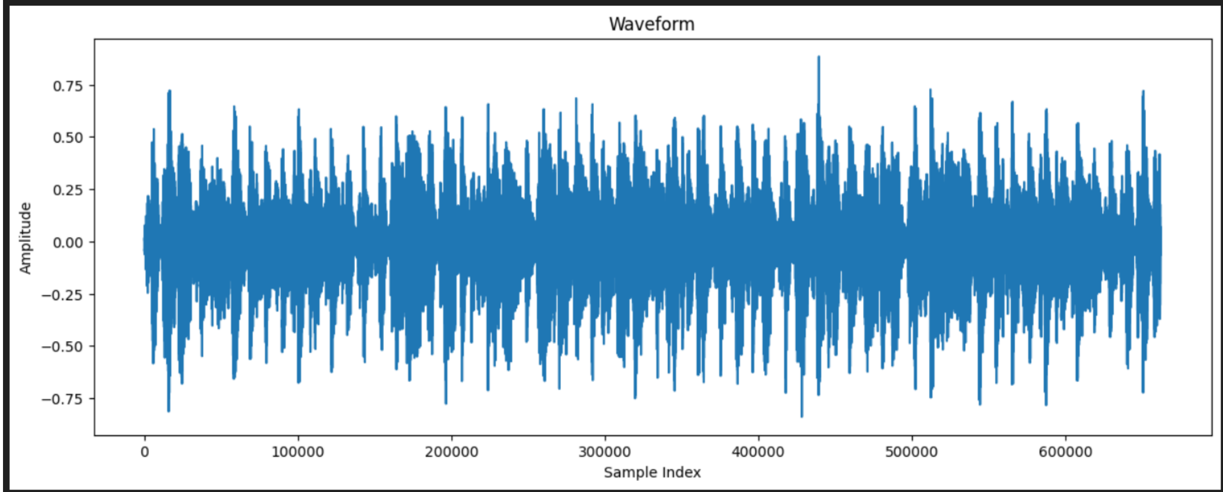
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
import zipfile
with zipfile.ZipFile('genres_original-20240412T172231Z-001.zip', 'r') as
zip_ref:
    zip_ref.extractall()
```

```
# BLUES MUSIC WAVEFORM
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/blues/blues.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x)
plt.title('Waveform')
```

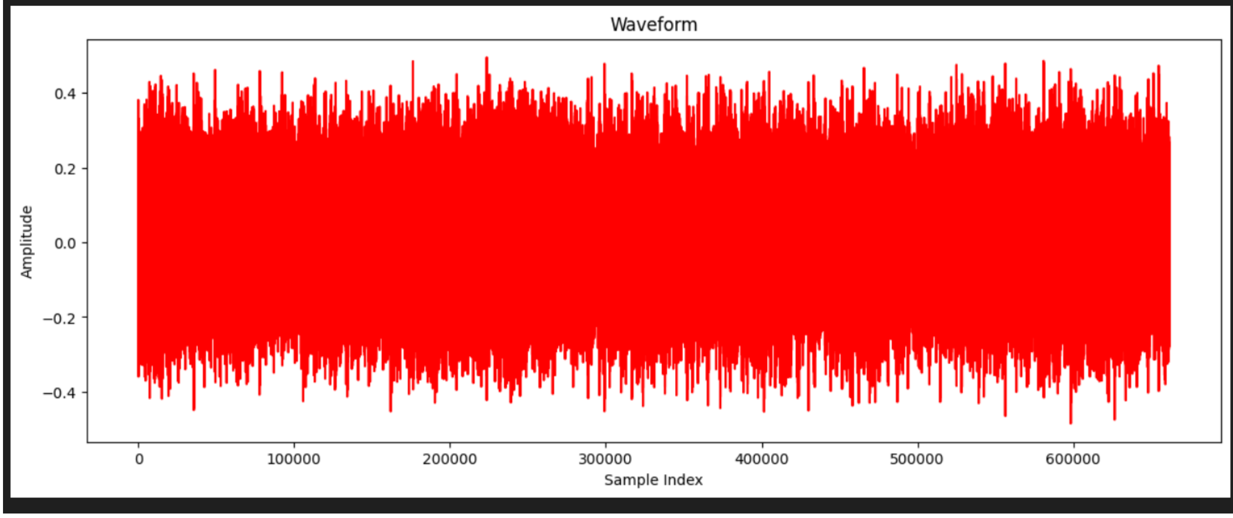
```
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("Blues")
```

```
Blues
```



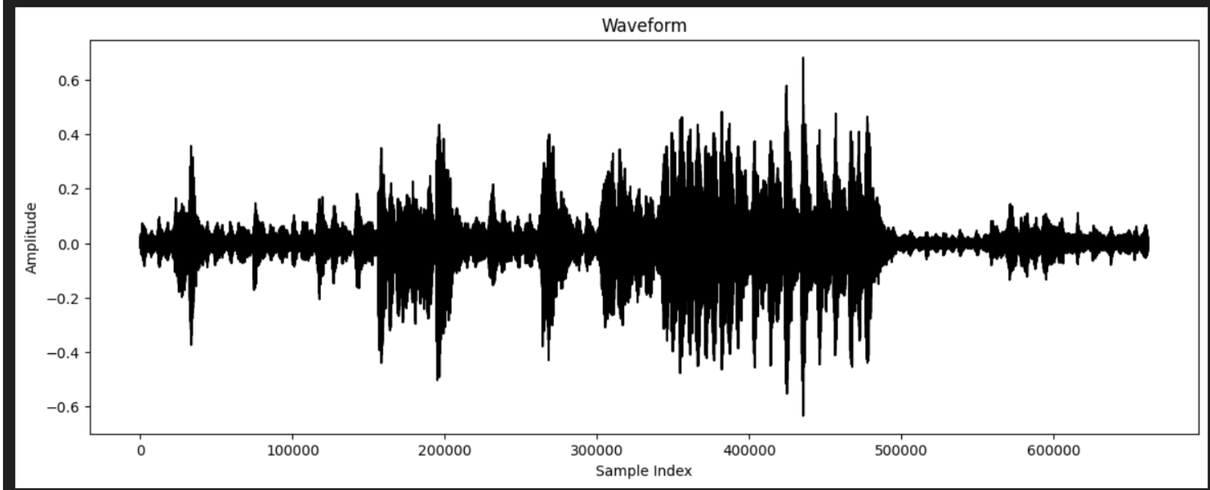

```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/metal/metal.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='red')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("Metal")
```

Metal




```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/jazz/jazz.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='black')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("JAZZ")
```

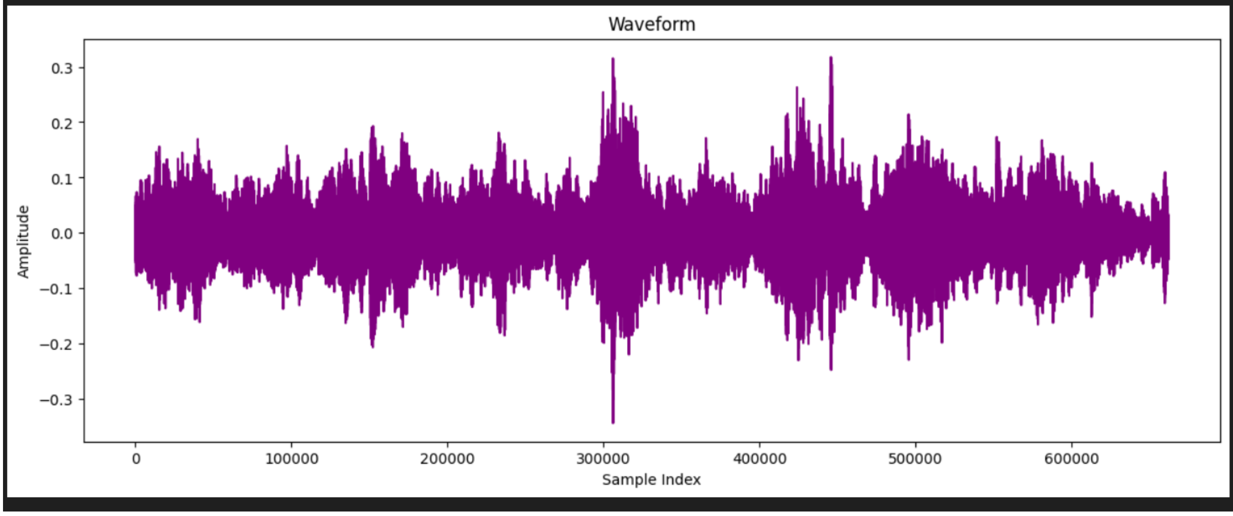
JAZZ




```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd
```

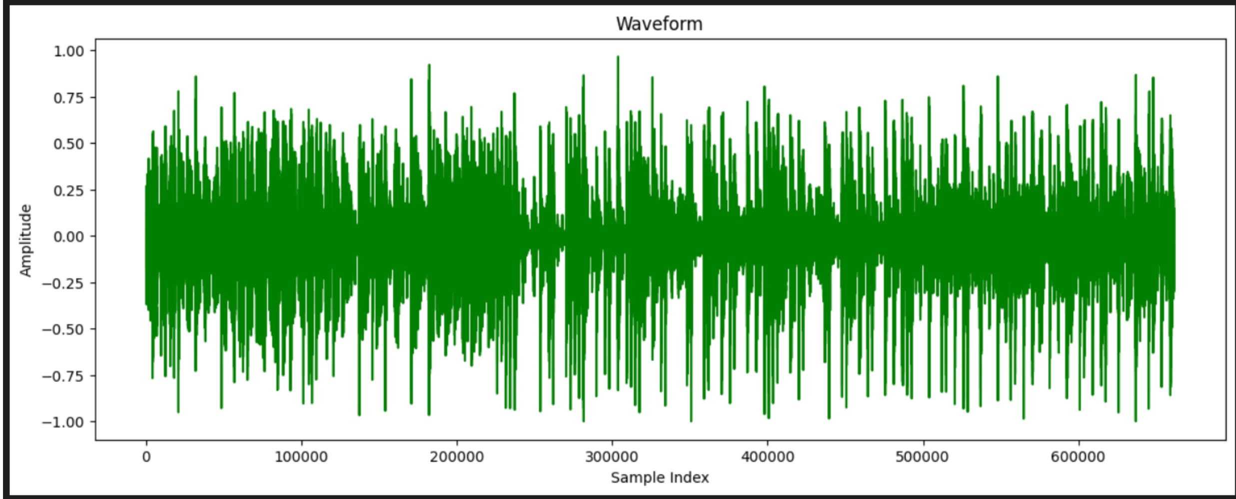
```python
path = 'genres_original/classical/classical.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='Purple')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("CLASSICAL")
```

CLASSICAL



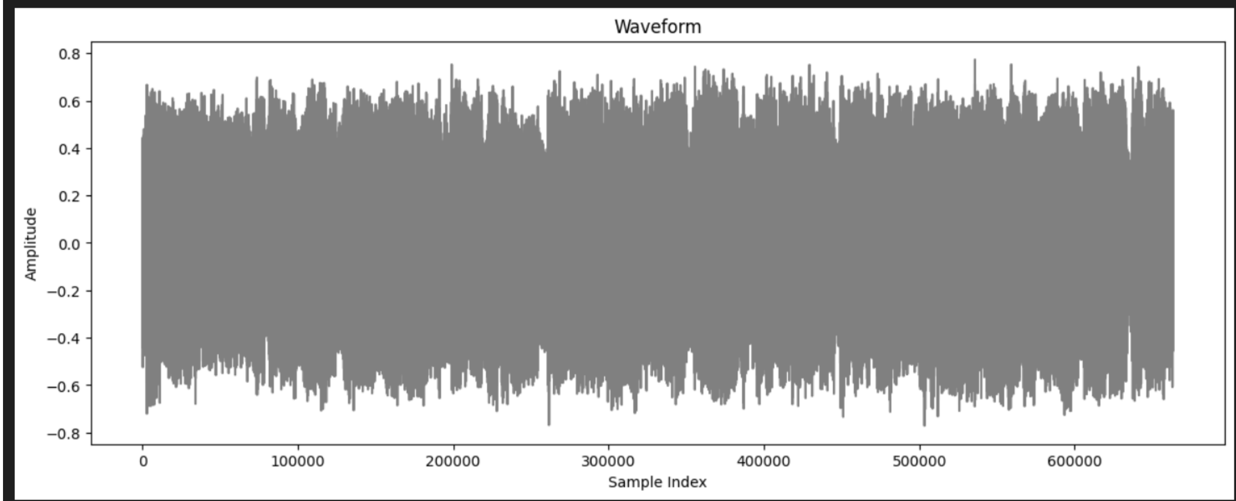

```python
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/hiphop/hiphop.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='Green')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("HIPHOP")
```

HIPHOP




```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/country/country.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='Grey')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("Country")
```

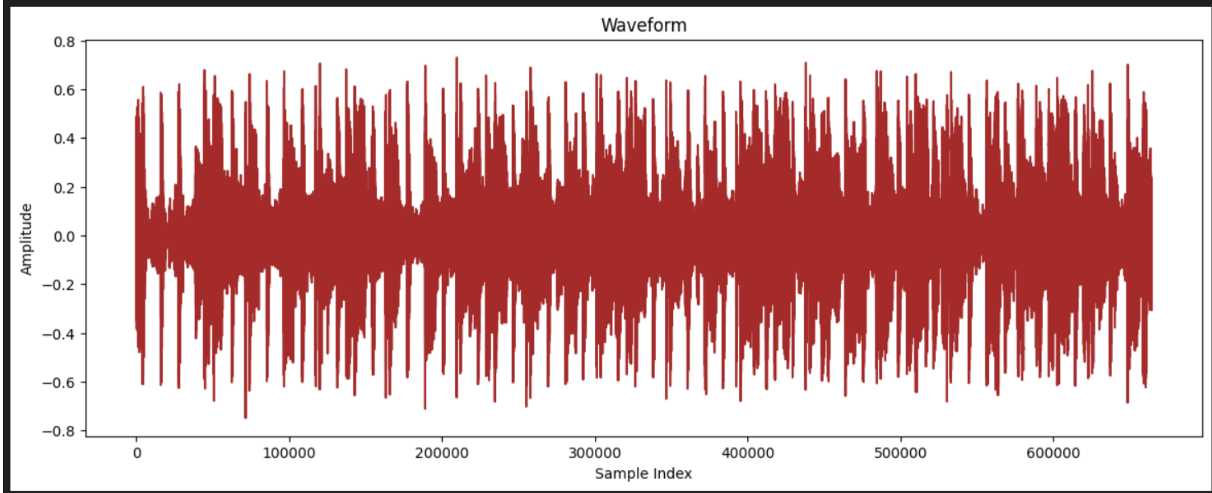
Country




```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd
```

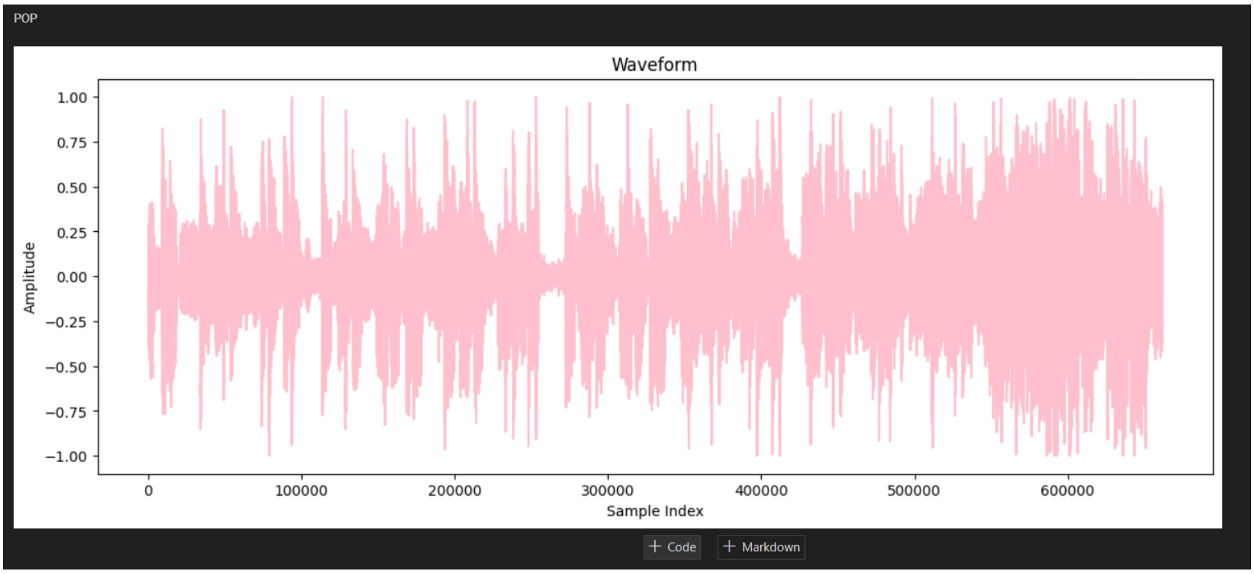
```
path = 'genres_original/disco/disco.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='Brown')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("DISCO")
```

DISCO

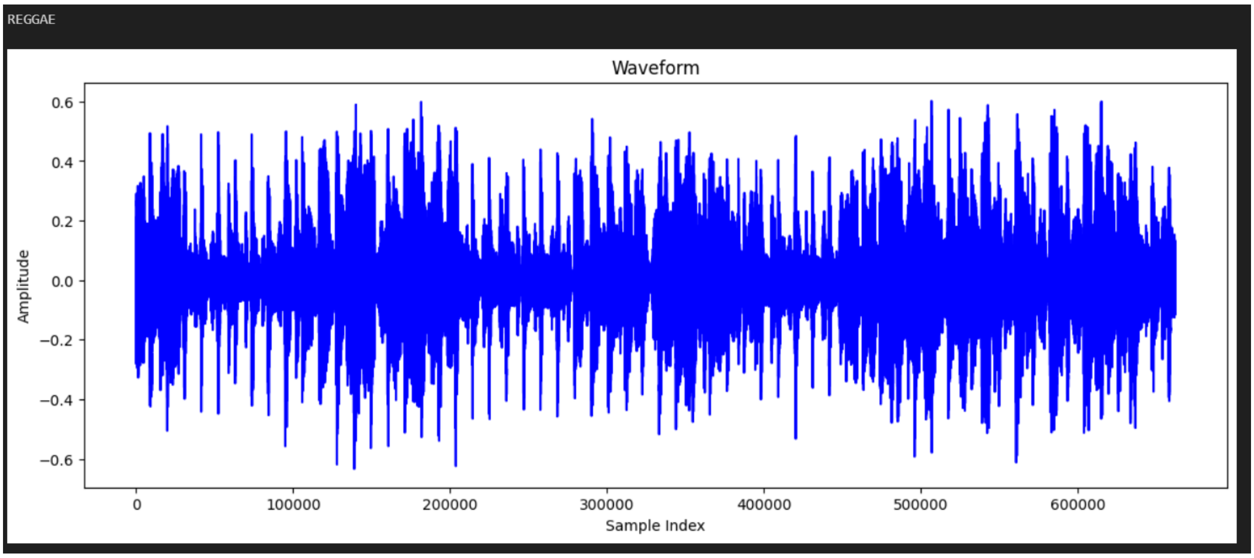


+ Code + Markdown

```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/pop/pop.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='pink')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("POP")
```



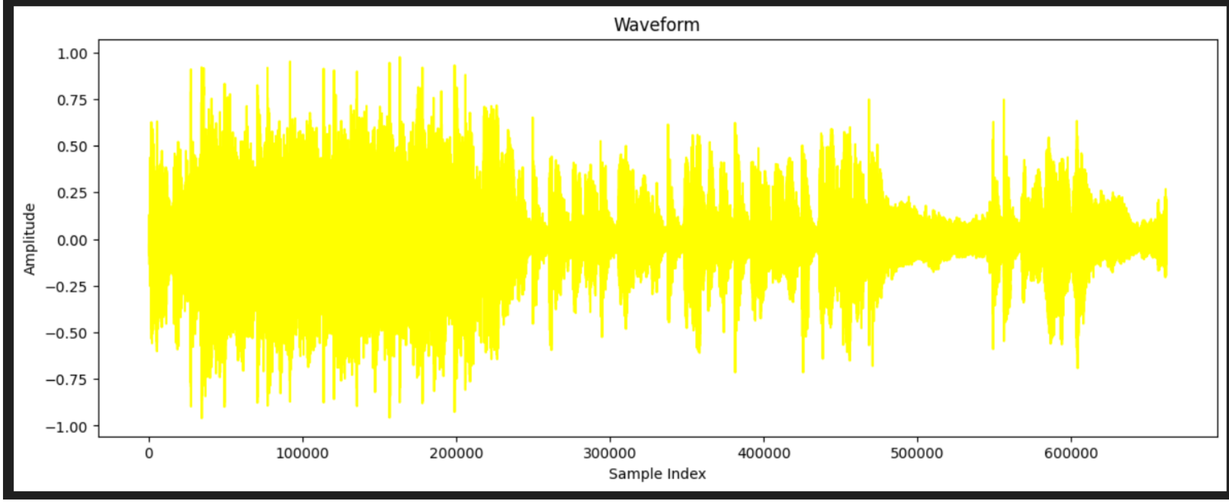

```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd

path = 'genres_original/reggae/reggae.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='blue')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("REGGAE")
```



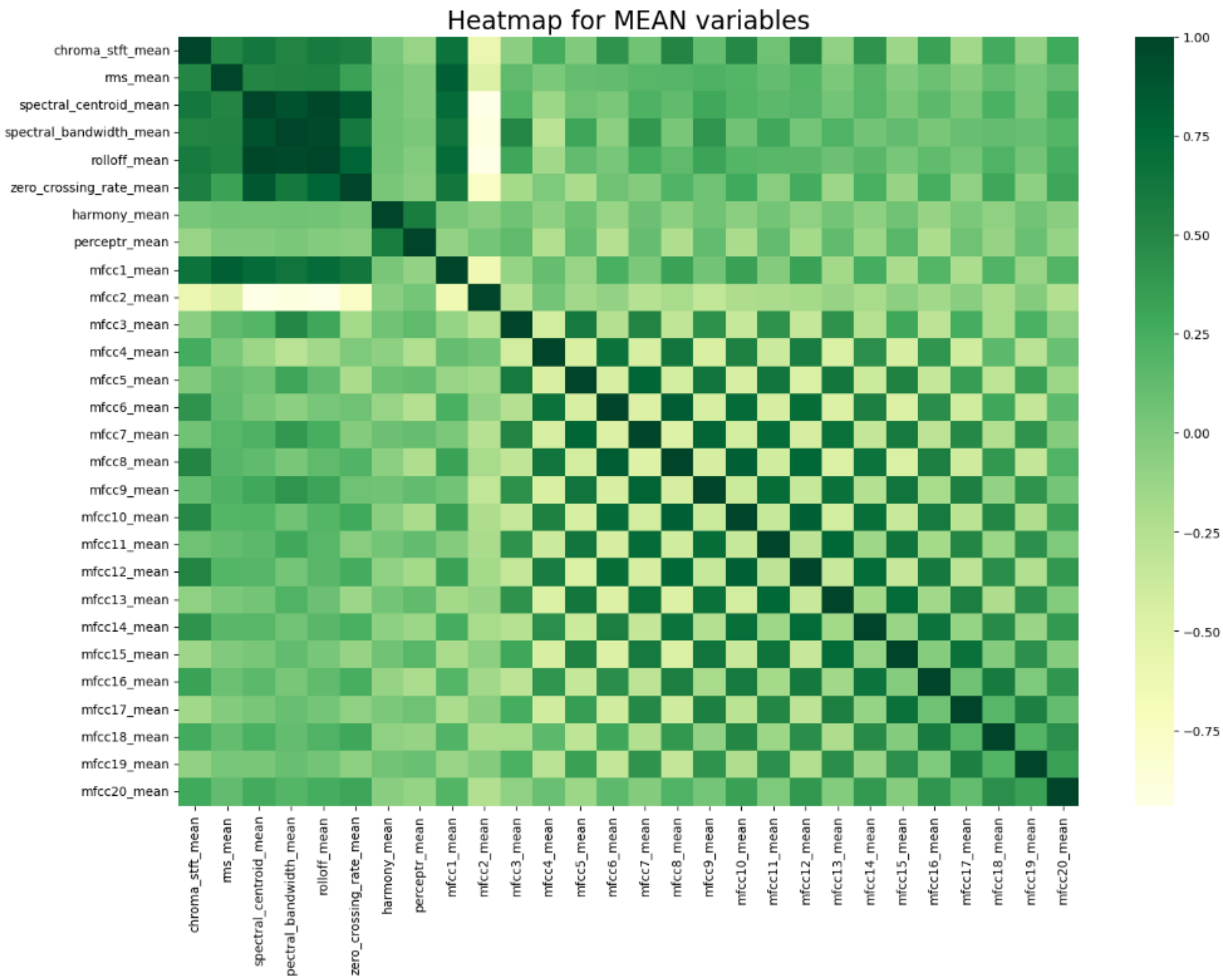

```
import librosa
import matplotlib.pyplot as plt
import IPython.display as ipd
```

```
path = 'genres_original/rock/rock.00000.wav'
x, sr = librosa.load(path)

plt.figure(figsize=(14, 5))
plt.plot(x, color='Yellow')
plt.title('Waveform')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')

ipd.Audio(path)

print("ROCK")
```

ROCK




```
# HEATMAP FOR MEAN VARIABLES
import numpy as np
import seaborn as sns

# Computing the Correlation Matrix
spike_cols = [col for col in music_data.columns if 'mean' in col]

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(16, 11));

# Draw the heatmap
sns.heatmap(music_data[spike_cols].corr(), cmap='YlGn')

plt.title('Heatmap for MEAN variables', fontsize = 20)
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10);
```




```
import scipy
import sys
import os
import pickle
import librosa.display
#import IPython.display import Audio
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
```

```
music_data.shape
```

```
(1000, 60)
```

```
music_data.dtypes
```

```
filename                     object
length                        int64
chroma_stft_mean            float64
chroma_stft_var             float64
rms_mean                    float64
rms_var                     float64
spectral_centroid_mean      float64
spectral_centroid_var       float64
spectral_bandwidth_mean     float64
spectral_bandwidth_var      float64
rolloff_mean                float64
rolloff_var                 float64
zero_crossing_rate_mean     float64
zero_crossing_rate_var      float64
harmony_mean                float64
harmony_var                 float64
perceptr_mean               float64
perceptr_var                float64
tempo                       float64
mfcc1_mean                  float64
mfcc1_var                   float64
mfcc2_mean                  float64
mfcc2_var                   float64
mfcc3_mean                  float64
mfcc3_var                   float64
...
mfcc19_var                  float64
mfcc20_mean                 float64
mfcc20_var                  float64
label                        object
dtype: object
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
audio_recording="genres_original/pop/pop.00000.wav"
data,sr=librosa.load(audio_recording)
print(type(data),type(sr))
```

```
data,sr=librosa.load(audio_recording)  # It loads and decodes the audio as a time series y.

#sr = sampling rate of y. It is the number of samples per second. 20 kHz is the audible range for human beings. So it is used as the default value for sr. In this code we are using sr as 45600Hz.
```

```
librosa.load(audio_recording)
```

```
(array([-0.0887146 , -0.09524536, -0.10275269, ...,  0.04016113,
         0.03860474,  0.02639771], dtype=float32),
 22050)
```

```
librosa.load(audio_recording, sr=45600)
```

```
(array([-0.08442919, -0.10617629, -0.09555261, ...,  0.01802196,
         0.01832535,  0.        ], dtype=float32),
 45600)
```

```
import IPython
IPython.display.Audio(data, rate=sr)
```

```
audio_recording2="genres_original/rock/rock.00000.wav"
data2,sr=librosa.load(audio_recording2)
print(type(data2),type(sr))
```

```
data2,sr=librosa.load(audio_recording2)
```

```
librosa.load(audio_recording2)
```

```
(array([-0.03344727, -0.05490112, -0.05435181, ..., -0.08416748,
         0.02886963,  0.1296997 ], dtype=float32),
 22050)
```

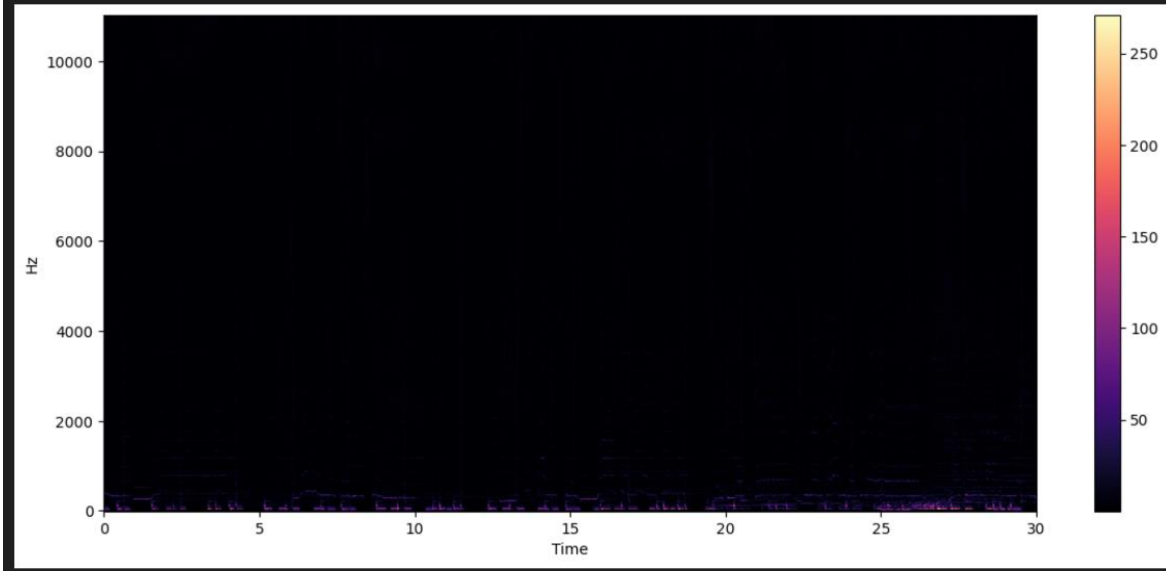
```
librosa.load(audio_recording, sr=45600)
```

```
(array([-0.08442919, -0.10617629, -0.09555261, ...,  0.01802196,
         0.01832535,  0.        ], dtype=float32),
 45600)
```

```
import IPython
IPython.display.Audio(data2, rate=sr)
```

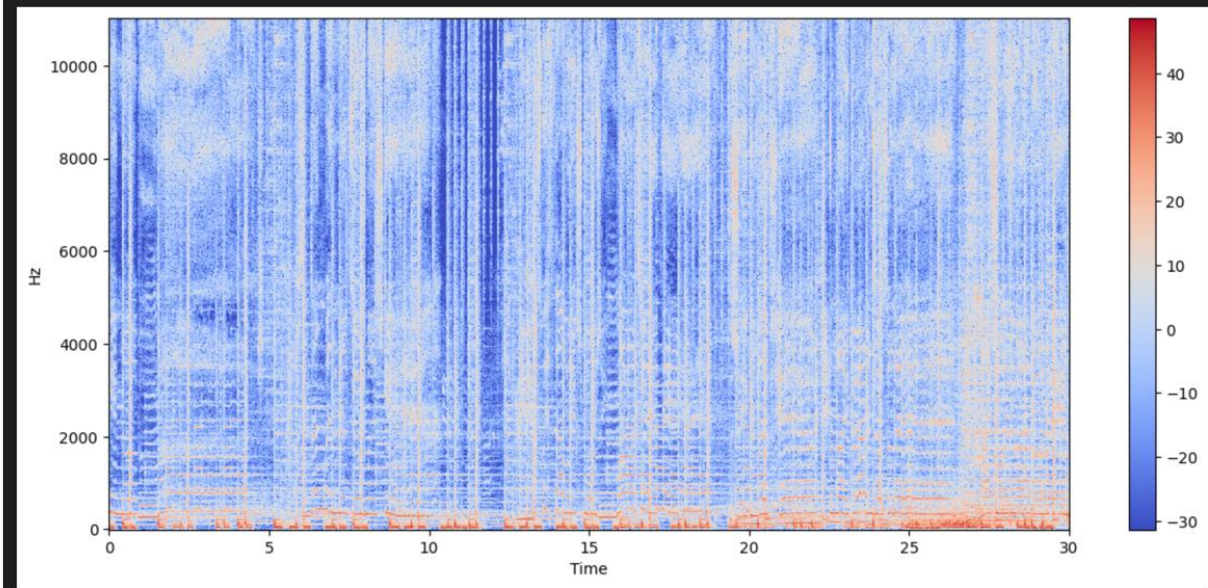
```
#SPECTOGRAMS
stft = librosa.stft(data)
stft_db = librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_1744\2619410286.py:5: UserWarning: Trying to display complex-valued input. Showing magnitude instead.
  librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')

<matplotlib.colorbar.Colorbar at 0x14ae327b8f0>
```



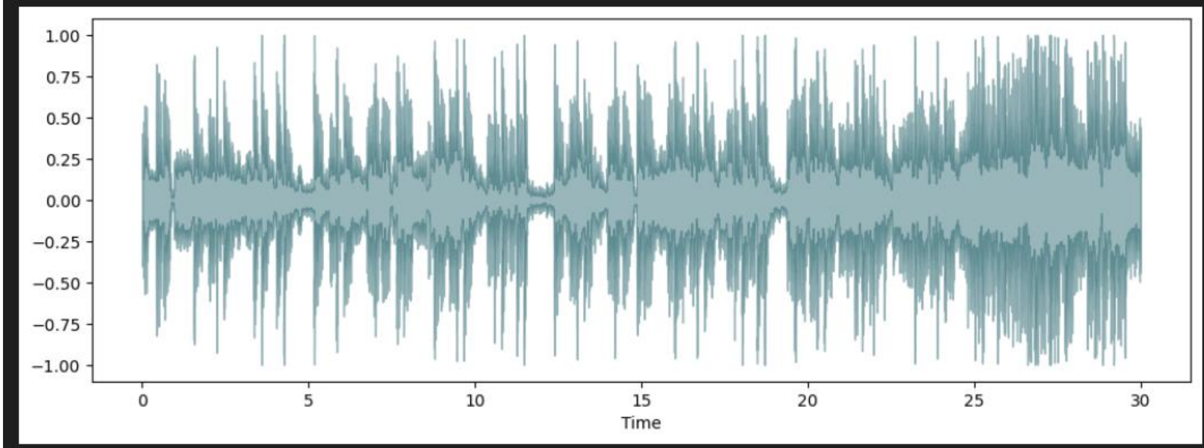

```
#SPECTOGRAMS
stft = librosa.stft(data)
stft_db = librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft_db,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x14a80fe8ce0>
```

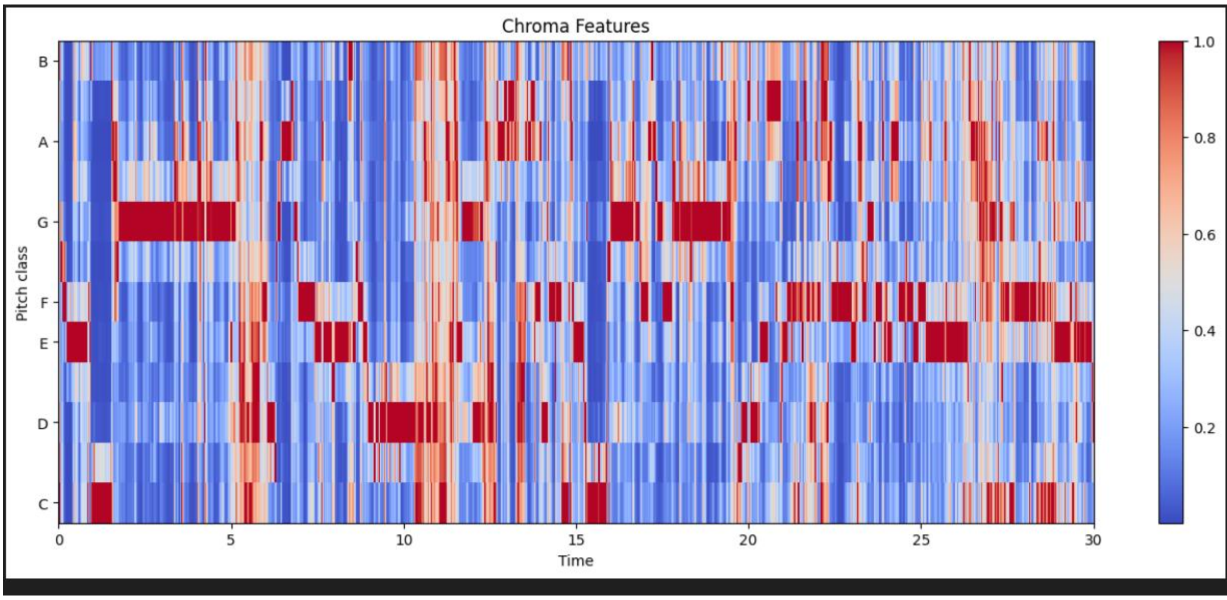



```
# Spectral Roll-Off
from sklearn.preprocessing import normalize

spectral_rolloff = librosa.feature.spectral_rolloff(y=data + 0.01, sr=sr)[0]
plt.figure(figsize=(12,4))
librosa.display.waveshow(data,sr=sr,alpha=0.4,color="#004953")
```

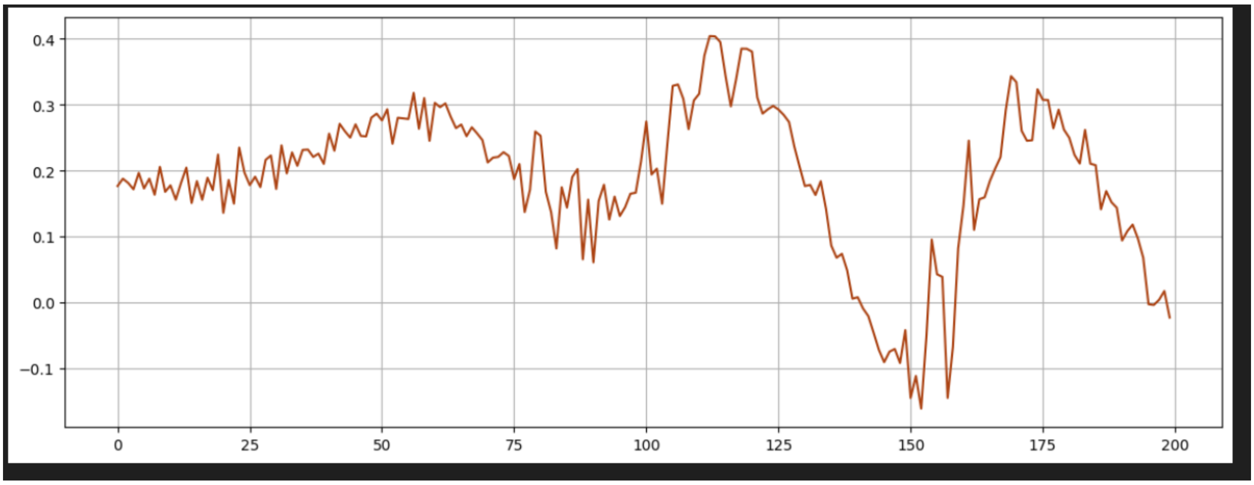
```
<librosa.display.AdaptiveWaveplot at 0x14ace3fd940>
```




```
# CHROMA FEATURE
import librosa.display as lplt
chroma = librosa.feature.chroma_stft(y=data,sr=sr)
plt.figure(figsize=(16,6))
lplt.specshow(chroma,sr=sr, x_axis='time', y_axis='chroma', cmap='coolwarm')
plt.colorbar()
plt.title("Chroma Features")
plt.show()
```




```
# ZERO CROSSING RATE

start=1000
end=1200
plt.figure(figsize=(14,5))
plt.plot(data[start:end],color="#ac4313")
plt.grid()
```




```
zero_cross_rate = librosa.zero_crossings(data[start:end], pad=False)
print("The number of zero-crossings is :",sum(zero_cross_rate))
```

```
# USE OF LABEL ENCODER CLASS IS USED TO CONVER CATEGORIAL TEXT DATA INTO MODEL
UNDERSTANABLE NUMERICAL DATA
```

```
class_list = music_data.iloc[:, -1]
convertor=LabelEncoder()
```

```
#fit_transform(): Fit label encoder and return encoded labels.

y=convertor.fit_transform(class_list)
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
...
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 9, 9])
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
print(music_data.iloc[:, :-1])
```

```
           filename  length  chroma_stft_mean  chroma_stft_var  rms_mean  \
0    blues.00000.wav  661794          0.350088         0.088757  0.130228
1    blues.00001.wav  661794          0.340914         0.094980  0.095948
2    blues.00002.wav  661794          0.363637         0.085275  0.175570
3    blues.00003.wav  661794          0.404785         0.093999  0.141093
4    blues.00004.wav  661794          0.308526         0.087841  0.091529
..               ...     ...               ...              ...       ...
995   rock.00095.wav  661794          0.352063         0.080487  0.079486
996   rock.00096.wav  661794          0.398687         0.075086  0.076458
997   rock.00097.wav  661794          0.432142         0.075268  0.081651
998   rock.00098.wav  661794          0.362485         0.091506  0.083860
999   rock.00099.wav  661794          0.358401         0.085884  0.054454

      rms_var  spectral_centroid_mean  spectral_centroid_var  \
0    0.002827             1784.165850          129774.064525
1    0.002373             1530.176679          375850.073649
2    0.002746             1552.811865          156467.643368
3    0.006346             1070.106615          184355.942417
4    0.002303             1835.004266          343399.939274
..        ...                     ...                    ...
995  0.000345             2008.149458          282174.689224
996  0.000588             2006.843354          182114.709510
997  0.000322             2077.526598          231657.968040
998  0.001211             1398.699344          240318.731073
999  0.000336             1609.795082          422203.216152
...
998    -5.041897   47.227180    -3.590644   41.299088
999    -2.025783   72.189316     1.155239   49.662510

[1000 rows x 59 columns]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
from sklearn.preprocessing import StandardScaler
import numpy as np

# Select only the numeric columns for scaling
numeric_columns = music_data.select_dtypes(include=[np.number]).columns

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the selected numeric columns
X = scaler.fit_transform(music_data[numeric_columns])

# Now X_scaled contains the scaled numeric data
```

```python
X
```

```
array([[-0.13282213, -0.35013678,  0.31258717, ..., -0.30059734,
         0.60406407, -0.51298758],
       [-0.13282213, -0.46248155,  1.11757233, ..., -0.40708699,
         0.42412706, -0.53842129],
       [-0.13282213, -0.18422456, -0.13770124, ..., -0.52729705,
        -0.29618888, -0.8749539 ],
       ...,
       [-0.13282213,  0.65463736, -1.43198917, ..., -0.63865065,
        -0.26361549, -0.89060474],
       [-0.13282213, -0.19833855,  0.66814351, ..., -0.5114848 ,
        -0.65064889, -0.63768256],
       [-0.13282213, -0.2483391 , -0.05894495, ...,  0.16033426,
         0.5868411 , -0.4526752 ]])
```

```
def features_extractor(file):
      audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
      mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
      mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

      return mfccs_scaled_features
```

```
metadata.drop(labels=552, axis=0, inplace=True)
```

```
from tqdm import tqdm
### Now we iterate through every audio file and extract features
### using Mel-Frequency Cepstral Coefficients
extracted_features=[]
for index_num,row in tqdm(metadata.iterrows()):
    try:
        final_class_labels=row["label"]
        file_name = os.path.join(os.path.abspath(audio_dataset_path),
final_class_labels+'/',str(row["filename"]))
        data=features_extractor(file_name)
        extracted_features.append([data, final_class_labels])
    except Exception as e:
        print(f"Error: {e}")
        continue
```

```
553it [03:34,  3.17it/s]<ipython-input-27-a2cfbf48bd37>:2: UserWarning: PySoundFile failed. Trying audioread instead.
  audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
/usr/local/lib/python3.10/dist-packages/librosa/core/audio.py:183: FutureWarning: librosa.core.audio.__audioread_load
        Deprecated as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
  y, sr_native = __audioread_load(path, offset, duration, dtype)
554it [03:35,  2.28it/s]
Error:
999it [06:24,  2.60it/s]
```

```
### converting extracted_features to Pandas dataframe
```

```
extracted_features_df=pd.DataFrame (extracted_features, columns=['feature',
'class'])
extracted_features_df.head()
```

```
X=np.array (extracted_features_df ['feature'].tolist())
y=np.array(extracted_features_df ['class'].tolist())
```

```
X.shape
```

```
(998, 40)
```

```
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing  import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical (labelencoder.fit_transform(y))
```

```
y.shape
```

```
(998, 10)
```

```
from sklearn.model_selection import train_test_split
X_train,
X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

```
X_train
```

```
array([[-1.04723763e+02,  8.77537155e+01, -3.32488594e+01, ...,
        -2.38248801e+00, -1.36347139e+00, -7.22123563e-01],
       [-2.59909851e+02,  1.23193169e+02, -6.39508581e+00, ...,
        -6.73697710e+00, -3.90829515e+00,  3.18117642e+00],
       [-1.15755066e+02,  6.70791245e+01,  1.88346851e+00, ...,
        -3.43661404e+00, -1.73870683e+00, -4.68738191e-02],
       ...,
       [-1.25020428e+01,  9.13173676e+01, -2.30759563e+01, ...,
        -4.04763937e+00, -1.77685583e+00, -1.75431299e+00],
       [-2.37930984e+01,  8.29835587e+01,  2.32049227e+00, ...,
         1.40550292e+00,  4.16220367e-01, -3.45980115e-02],
       [-9.63196945e+01,  9.09497147e+01, -3.22195396e+01, ...,
        -2.41483903e+00, -1.62698299e-01, -1.84749973e+00]], dtype=float32)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(798, 40)
(200, 40)
(798, 10)
(200, 10)
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.15.0
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn import metrics
```

```
num_labels=y.shape[1]
```

```
model=Sequential()
model.add(Dense (1024, input_shape=(40,), activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense (32, activation="relu"))
model.add(Dropout(0.3))

###final layer
model.add(Dense(num_labels, activation="softmax"))
```

```
model.summary()
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 1024)              41984

 dropout_18 (Dropout)        (None, 1024)              0

 dense_19 (Dense)            (None, 512)               524800

 dropout_19 (Dropout)        (None, 512)               0

 dense_20 (Dense)            (None, 256)               131328

 dropout_20 (Dropout)        (None, 256)               0

 dense_21 (Dense)            (None, 128)               32896

 dropout_21 (Dropout)        (None, 128)               0

 dense_22 (Dense)            (None, 64)                8256

 dropout_22 (Dropout)        (None, 64)                0

 dense_23 (Dense)            (None, 32)                2080
...
Total params: 741674 (2.83 MB)
Trainable params: 741674 (2.83 MB)
Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
optimizer='adam')
```

```
import time
t = time.localtime()
current_time = time.strftime("%H:%M:%S", t)
```

```
## Trianing my model
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 100
num_batch_size = 32
```

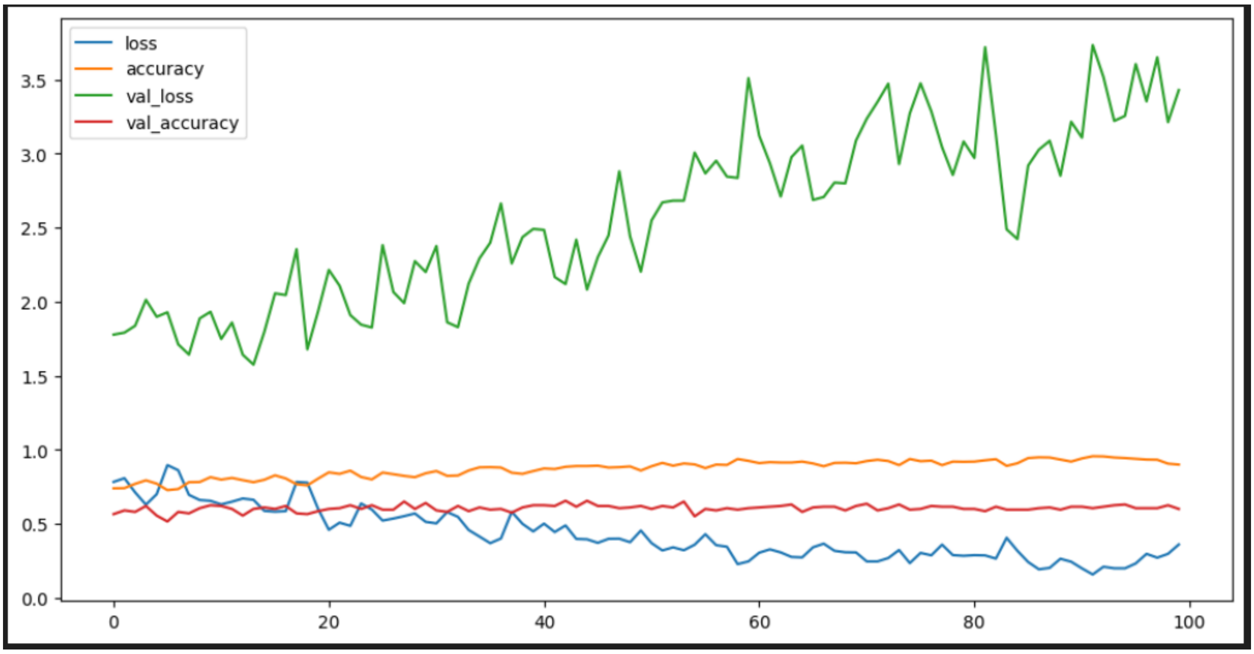
```python
checkpointer =
ModelCheckpoint(filepath=f'saved_models/audio_classification_{current_time}.hdf5
', verbose=1, save_best_only=True)
start=datetime.now()
history=model.fit(X_train, y_train, batch_size=num_batch_size,
epochs=num_epochs, validation_data=(X_test, y_test), callbacks=[checkpointer],
verbose=1)
duration=datetime.now() - start
print("Training completed in time:", duration)
```

```
Epoch 1/100
23/25 [==========================>...] - ETA: 0s - loss: 0.7900 - accuracy: 0.7364
Epoch 1: val_loss improved from inf to 1.77780, saving model to saved_models/audio_classification_17:41:23.hdf5
25/25 [==============================] - 1s 58ms/step - loss: 0.7827 - accuracy: 0.7393 - val_loss: 1.7778 - val_accuracy: 0.5650
Epoch 2/100
24/25 [===========================>..] - ETA: 0s - loss: 0.8141 - accuracy: 0.7383
Epoch 2: val_loss did not improve from 1.77780
25/25 [==============================] - 1s 26ms/step - loss: 0.8076 - accuracy: 0.7406 - val_loss: 1.7911 - val_accuracy: 0.5900
Epoch 3/100
24/25 [===========================>..] - ETA: 0s - loss: 0.7090 - accuracy: 0.7708
Epoch 3: val_loss did not improve from 1.77780
25/25 [==============================] - 1s 29ms/step - loss: 0.7118 - accuracy: 0.7694 - val_loss: 1.8378 - val_accuracy: 0.5800
Epoch 4/100
24/25 [===========================>..] - ETA: 0s - loss: 0.6306 - accuracy: 0.7904
Epoch 4: val_loss did not improve from 1.77780
25/25 [==============================] - 1s 29ms/step - loss: 0.6289 - accuracy: 0.7932 - val_loss: 2.0130 - val_accuracy: 0.6200
Epoch 5/100
25/25 [==============================] - ETA: 0s - loss: 0.6997 - accuracy: 0.7707
Epoch 5: val_loss did not improve from 1.77780
25/25 [==============================] - 0s 19ms/step - loss: 0.6997 - accuracy: 0.7707 - val_loss: 1.8984 - val_accuracy: 0.5550
Epoch 6/100
25/25 [==============================] - ETA: 0s - loss: 0.8961 - accuracy: 0.7268
Epoch 6: val_loss did not improve from 1.77780
25/25 [==============================] - 0s 18ms/step - loss: 0.8961 - accuracy: 0.7268 - val_loss: 1.9289 - val_accuracy: 0.5150
Epoch 7/100
...
25/25 [==============================] - ETA: 0s - loss: 0.3596 - accuracy: 0.8997
Epoch 100: val_loss did not improve from 1.57508
25/25 [==============================] - 1s 35ms/step - loss: 0.3596 - accuracy: 0.8997 - val_loss: 3.4301 - val_accuracy: 0.6000
Training completed in time: 0:00:59.097544
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
model.evaluate(X_test, y_test, verbose=0)
```

```
[3.430067539215088, 0.6000000238418579]
```

```python
pd.DataFrame(history.history).plot(figsize=(12,6))
plt.show()
```




```python
np.argmax(model.predict(X), axis=-1)
```

```
32/32 [==============================] - 1s 7ms/step

array([0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 2, 2, 4, 2, 9, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 7, 5, 2, 2, 2, 2, 5, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 3, 2, 1, 2, 9, 2, 9, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 3, 0, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 5, 2, 2, 2, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 3, 3, 3, 3, 0, 3, 9, 3,
       3, 3, 4, 3, 4, 3, 3, 3, 2, 3, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 7, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 9, 3, 3,
       3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 4, 4, 3, 3, 3, 9, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 9, 3, 0, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 4, 4, 5, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 8, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4,
       5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 8, 4, 4,
       1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 1, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
...
       0, 9, 6, 9, 9, 0, 0, 9, 9, 0, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
       0, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 5, 9,
       9, 9, 9, 9, 9, 9, 9, 9, 3, 9, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 3, 9,
       9, 4, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 3, 9, 9, 0, 9, 9,
       9, 9, 9, 9, 9, 9, 9, 3])
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
filename="/content/drive/MyDrive/Colab Notebooks/AI-ML 
PROJECT/Data/genres_original/classical/classical.00003.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features = mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label = np.argmax(model.predict(mfccs_scaled_features))
print(predicted_label)
predicted_label = np.array([predicted_label])  # Reshape to a 1D array
prediction_class = labelencoder.inverse_transform(predicted_label)
prediction_class
```

```
[-3.26897430e+02  1.27400604e+02 -3.05961761e+01  3.74713974e+01
 -5.29617548e+00  2.08135643e+01 -1.77868533e+00 -4.61545658e+00
 -4.04019880e+00  5.24227428e+00 -6.42448783e-01  2.79456735e+00
  7.22920656e+00  3.97069526e+00 -2.39217043e+00  1.48818469e+00
  5.89341402e-01 -5.93018532e-01  2.34307027e+00  2.63825250e+00
  4.55192041e+00 -3.03187817e-01 -1.59056199e+00  1.71130866e-01
  1.19197810e+00 -3.59523967e-02  2.08741593e+00  3.54622483e-01
  1.03717148e+00 -2.67570233e+00 -6.66255951e+00 -2.44702744e+00
  2.09354901e+00  2.43527636e-01  1.39810920e-01  1.45328581e+00
 -2.91861296e+00 -3.13705468e+00  4.90082932e+00 -4.18077499e-01]
[[-3.26897430e+02  1.27400604e+02 -3.05961761e+01  3.74713974e+01
  -5.29617548e+00  2.08135643e+01 -1.77868533e+00 -4.61545658e+00
  -4.04019880e+00  5.24227428e+00 -6.42448783e-01  2.79456735e+00
   7.22920656e+00  3.97069526e+00 -2.39217043e+00  1.48818469e+00
   5.89341402e-01 -5.93018532e-01  2.34307027e+00  2.63825250e+00
   4.55192041e+00 -3.03187817e-01 -1.59056199e+00  1.71130866e-01
   1.19197810e+00 -3.59523967e-02  2.08741593e+00  3.54622483e-01
   1.03717148e+00 -2.67570233e+00 -6.66255951e+00 -2.44702744e+00
   2.09354901e+00  2.43527636e-01  1.39810920e-01  1.45328581e+00
  -2.91861296e+00 -3.13705468e+00  4.90082932e+00 -4.18077499e-01]]
(1, 40)
1/1 [==============================] - 0s 34ms/step
1

array(['classical'], dtype='<U9')
```

CHAPTER 4

CONCLUSION AND References

4.1 CONCLUSION

- **Successful Implementation of Music Genre Classification:** Learning Python programming and scripting with both Google Colab and Jupyter Notebook, we have set up an effective mechanism that classifies music on the basis of the genre to the user's input. Utilising sports complex algorithms, we have built solutions that can cleanly pick out distinct types of music based on the sound features they possess.
- **Enhanced Understanding of Music Analysis Techniques:** This entire project, to my sastisfaction, has taught me a great deal in the particular area of music analysis namely featuers extraction, spectrogram, and a model building. We are fascinated to develop machine learning and deep learning using these approaches while we do these exercises in a workshop style environment.
- **Potential for Real-World Applications and Further Research:** The key theme of the developed music genre classification system, however, is the world's real-world application, and these can be exemplified by music recommendation systems, organization of content in the digital libraries, genre-based content tagging, and personal navigation tools Moreover this project can become an example in audio signal processing, machine learning, and music analysis, which will serve as a starting point in this field where a lot of creations will come in the future.

4.2 References

- https://www.analyticsvidhya.com/blog/2022/03/music-genre-classification-project-using-machine-learning-techniques/
- https://www.geeksforgeeks.org/music-genre-classifier-using-machine-learning/
- https://github.com/topics/music-genre-classification
- https://scholar.google.co.in/scholar?q=music+genre+classification+AI+ML&hl=en&as_sdt=0&as_vis=1&oi=scholart
- https://data-flair.training/blogs/python-project-music-genre-classification/

BIODATA

NAME: Ashrit Saha

MOBILE NUMBER: 7439933150

E-MAIL: ashrit.saha2021@vitstudent.ac.in

PERMANENT ADDRESS: 10/1/2R Atal Sur Road, Kolkata - 700015

NAME: Biswajyoti Dutta

MOBILE NUMBER: 8927969552

E-MAIL: biswajyoti.dutta2021@vitstudent.ac.in

PERMANENT ADDRESS: South Bharatnagar near Athletic Club, Siliguri - 734004