Music is an essential part of our lives and, music streaming companies like Spotify are nowadays using machine learning to create recommendations for us. Music genres play a big role in creating these recommendations. In this story, we will build a model for the classification of music tracks into their respective genres. For this tutorial, we‘ll use*librosa,* a library for music and audio analysis. We will also use the [GTZAN Dataset](https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification) from Kaggle for training our classifier.

**Importing Libraries**

import os  
import pandas as pd  
import numpy as np  
import IPython  
import librosa  
import librosa.display  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn import preprocessing  
from sklearn.preprocessing import minmax\_scale  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score

**Different Genres**

Let’s see the entire list of genres present in the dataset.

general\_path = './Data'  
print(list(os.listdir('Data/genres\_original/')))



**Taking a Single Audio**

We’ll take a single audio file for exploration.

file = './Data/genres\_original/classical/classical.00050.wav'  
signal , sr = librosa.load(file , sr = 22050)

**Signal Visualization**

Let’s use a wave plot to visualize the audio file or signal.

plt.figure(figsize=(15,5))  
librosa.display.waveplot(signal , sr = sr)  
plt.xlabel('Time')  
plt.ylabel('Amplitude')  
plt.title("Classical music signal")  
plt.show()

A blue sound wave on a white background

Description automatically generated

**Playing the Audio**

To play the audio in the Jupyter Notebook, we’ll use the following code.

IPython.display.Audio(signal, rate=sr)

A black line on a white background

Description automatically generated

Output

**Fourier Transform**

Fourier Transform is a function that takes a signal in the time domain as input and outputs its decomposition into frequencies. Let’s plot a graph using it to see the distribution of frequencies.

n\_fft = 2048  
hop\_length = 512D = np.abs(librosa.stft(signal, n\_fft = n\_fft, hop\_length = hop\_length))plt.figure(figsize = (15, 5))  
plt.plot(D)  
plt.title('Fourier Transform')  
plt.show()

A white screen with a black border

Description automatically generated

**Spectrogram**

A spectrogram is a representation of the loudness of a signal over time at various frequencies present in a particular waveform. Let’s plot a spectrogram for our audio file.

DB = librosa.amplitude\_to\_db(D, ref = np.max)plt.figure(figsize = (15, 5))  
librosa.display.specshow(DB, sr = sr, hop\_length = hop\_length, x\_axis = 'time', y\_axis = 'log', cmap = 'cool')plt.colorbar()  
plt.title('Spectrogram')  
plt.show()

A blue and pink background

Description automatically generated

**Harmonics & Perceptual**

Harmonics are unwanted higher frequencies superimposed on the fundamental waveform creating a distorted wave pattern. Perceptual represents the sound rhythm and emotion. Let’s plot both of them on a graph.

a,b = librosa.effects.hpss(signal)plt.figure(figsize = (15, 5))  
plt.plot(a, color = '#FF5E33');  
plt.plot(b, color = '#FFD433');  
plt.title('Harmonics and Perceptrual')  
plt.show()

A sound wave with orange and yellow lines

Description automatically generated

**Spectral Centroid**

The spectral centroid indicates where the center of mass of the spectrum is located.

spectral\_centroids = librosa.feature.spectral\_centroid(signal, sr=sr)[0]plt.figure(figsize=(15, 5))  
frames = range(len(spectral\_centroids))  
t = librosa.frames\_to\_time(frames)  
librosa.display.waveplot(signal, sr=sr, alpha=0.4)plt.plot(t, minmax\_scale(spectral\_centroids,axis=0), color='r')  
plt.title('Spectral Centroid')  
plt.show()

A graph of a heart rate

Description automatically generated with medium confidence

**Chromogram**

We will create a chromogram in which the entire spectrum will be projected onto 12 bins representing the 12 distinct semitones of the musical octave.

hop\_length = 5000chromagram = librosa.feature.chroma\_stft(signal, sr=sr, hop\_length=hop\_length)plt.figure(figsize=(15, 5))librosa.display.specshow(chromagram, x\_axis='time', y\_axis='chroma', hop\_length=hop\_length, cmap='YlGnBu')plt.title('Chromogram')  
plt.show()

A blue and white rectangles

Description automatically generated

**30 Second Features**

The *‘features\_30\_sec.csv’*file contains audio features over a duration of 30 seconds. We will use it for studying beats per minute.

data = pd.read\_csv('./Data/features\_30\_sec.csv')  
data.head()

A screenshot of a computer

Description automatically generated

Output (Truncated)

**Beats per Minute**

Let’s see the beats per minute for different genres.

x = data[["label", "tempo"]]f, ax = plt.subplots(figsize=(15, 5));sns.boxplot(x = "label", y = "tempo", data = x, palette = 'husl');plt.title('BPM for Genres', fontsize = 20)  
plt.xticks(fontsize = 14)  
plt.yticks(fontsize = 10);  
plt.xlabel("Genre", fontsize = 15)  
plt.ylabel("BPM", fontsize = 15)  
plt.show()

A diagram of different colored squares

Description automatically generated

**Loading the Data**

The *‘features\_3\_sec.csv’*filecontains audio features over a duration of 3 seconds. We will use it to train our model.

data = pd.read\_csv('./Data/features\_3\_sec.csv')  
data = data.iloc[0:, 1:]

A screenshot of a computer

Description automatically generated

Output (Truncated)

**Preprocessing the Data**

We will preprocess the data to make it suitable for our model. We’ll use *MinMaxScaler* for this purpose.

y = data['label']  
X = data.loc[:, data.columns != 'label']cols = X.columnsmin\_max\_scaler = preprocessing.MinMaxScaler()  
np\_scaled = min\_max\_scaler.fit\_transform(X)  
X = pd.DataFrame(np\_scaled, columns = cols)

**Splitting the Data**

Let’s split the data for training and testing purposes.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

**Model Creation & Prediction**

It’s time to create our model. We will use Random Forest Classifier to built the model. We’ll fit the model using the training data and predict the testing data. Our model’s accuracy turns out to be 81.38 %, which is great!

model = RandomForestClassifier(n\_estimators=1000, max\_depth=10, random\_state=0)model.fit(X\_train, y\_train)preds = model.predict(X\_test)  
print('Accuracy:', round(accuracy\_score(y\_test, preds)))



Output