Project Title : Music Genre Classification using Machine Learning

(Project 3) ¶

- Shiva Sai Soma - 11600652
- Yeshwanth Buggaveeti - 11546178
- Pavan Sai Gundaram - 11510519
- Bakht Singh Basaram - 11546730

Data Set Information:

The Audio Files for Music Genre Classification are downloaded from https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification (https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification). Later using the raw audio files we have extracted time domain and frequency domain features and created our own dataset.

These features of the dataset are:

- Amplitude Envelope
- Root Mean Square Energy
- Zero Crossing Rate
- Spectrogram
- Mel Spectrogram
- Mel Frequency Ceptral Coefficients
- Spectral Centroid
- Spectral Bandwidth
- Tempo
- Chroma - STFT
- Spectral Rolloff
- Labels - 10 music genres

References for code and :

1. For Feature Extraction https://github.com/musikalkemist/AudioSignalProcessingForML (https://github.com/musikalkemist/AudioSignalProcessingForML)
2. For understanding the theory https://youtube.com/playlist?list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0 (https://youtube.com/playlist?list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0)
3. Some part of the code(model creation using optuna) is reused from https://www.kaggle.com/code/calebreigada/liver-disease-analysis-eda-smote-optuna-shap/notebook (https://www.kaggle.com/code/calebreigada/liver-disease-analysis-eda-smote-optuna-shap/notebook)

In [700]:
```
# importing required libraries

import os, sys
import numpy as np
import pandas as pd
import librosa
import IPython.display as ipd
from tqdm import tqdm
import matplotlib.pyplot as plt
import librosa.display
import glob
from itertools import chain
import warnings
warnings.filterwarnings('ignore')
```

Feature Extraction from Audio Files Starts here:

In [701]:
```
# Getting the files from local machine

dir_list = ["blues","classical","country","disco","hiphop","jazz","metal","
basedir = os.getcwd()
file_paths = []
file_names = []

for dirs in dir_list:
    path_of_the_directory = basedir+ "/genres_original"+"/"+dirs
    #print("Files and directories in a specified path:")
    list_of_files = sorted( filter( os.path.isfile,
                        glob.glob(path_of_the_directory + '/**/*', recursiv
    file_name = sorted( filter( lambda x: os.path.isfile(os.path.join(path_
                              os.listdir(path_of_the_directory) ) )

    file_names.append(file_name)
    file_paths.append(list_of_files)
```

In [453]:
```
# convert 2d list to 1d
file_names_list = list(chain.from_iterable(file_names))
file_paths_list = list(chain.from_iterable(file_paths))
```

In [454]:
```
# creating target column
labels = []
for names in file_names_list:
    names_splitted = names.split('.')
    labels.append(names_splitted[0])
```

In [455]:
```
# create a dataset
filenames_df = pd.DataFrame(file_names_list,columns =['filename'])
labels_df = pd.DataFrame(labels,columns =['labels'])
```

In [705]:
```
# loading an audio file
ipd.Audio(file_paths_list[602])
```

Out[705]:

0:07 / 0:30

In [457]:
```
# know the files that are not supported and drop them from the list
files_not_supported = []
index = 0
for files in file_paths_list:
    try:
        signal, sr = librosa.load(files)
    except Exception:
        print(index)
        files_not_supported.append(files)
        filenames_df.drop(index, inplace = True )
        labels_df.drop(index, inplace = True )
        del file_paths_list[index]
    index += 1
```

```
554
```

In [458]:
```
data = filenames_df
```

In [ ]:
```
```

In [459]:
```
# Amplitude Envelope - Function

def feature_amplitude_envelope(signal, frame_size, hop_length):
    """Calculate the amplitude envelope of a signal with a given frame size
    amplitude_envelope = []

    # calculate amplitude envelope for each frame
    for i in range(0, len(signal), hop_length):
        amplitude_envelope_current_frame = max(signal[i:i+frame_size])
        amplitude_envelope.append(amplitude_envelope_current_frame)

    return np.array(amplitude_envelope)
```

In [460]:
```
# Feature Extraction - Amplitude Envelope

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_amplitude_envelope = []
#files_not_supported = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    ae = feature_amplitude_envelope(signal, FRAME_SIZE, HOP_LENGTH)
    list_amplitude_envelope.append(ae.mean())
```

In [461]:
```
# feature - amplitude envelope dataframe
data['amplitude_envelope_mean'] = pd.DataFrame(list_amplitude_envelope)
```

In [462]:
```
# Root Mean Square Energy - Function

def feature_rmse(signal, frame_size, hop_length):
    rmse = []

    # calculate rmse for each frame
    for i in range(0, len(signal), hop_length):
        rmse_current_frame = np.sqrt(sum(signal[i:i+frame_size]**2) / frame
        rmse.append(rmse_current_frame)
    return np.array(rmse)
```

In [463]:
```
# Feature Extraction - Root Mean Square Energy

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_rmse = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    rmse = feature_rmse(signal, FRAME_SIZE, HOP_LENGTH)
    list_rmse.append(rmse.mean())
```

In [464]:
```
# feature - rmse dataframe
data['rmse_mean'] = pd.DataFrame(list_rmse)
```

In [525]:
```
# Feature Extraction - Zero Crossing Rate

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_zcr = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    zcr = librosa.feature.zero_crossing_rate(signal, frame_length=FRAME_SIZ
    list_zcr.append(zcr.mean())
```

In [526]:
```
# feature - rmse dataframe
data['zcr_mean'] = pd.DataFrame(list_zcr)
```

In [467]:
```
# Feature Extraction - Spectrogram

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_spectrogram = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    S_scale = librosa.stft(signal, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)
    Y_scale = np.abs(S_scale) ** 2
    Y_log_scale = librosa.power_to_db(Y_scale)
    list_spectrogram.append(Y_log_scale.mean())
```

In [468]:
```
# feature - Spectrogram dataframe
data['spectrogram_mean'] = pd.DataFrame(list_spectrogram)
```

In [469]:
```
# Feature Extraction - Mel Spectrogram

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_mel_spectrogram = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    mel_spectrogram = librosa.feature.melspectrogram(signal, sr=sr, n_fft=F
    log_mel_spectrogram = librosa.power_to_db(mel_spectrogram)
    list_mel_spectrogram.append(log_mel_spectrogram.mean())
```

In [498]:
```
# feature - Mel Spectrogram dataframe
data['mel_spectrogram_mean'] = pd.DataFrame(list_mel_spectrogram)
```

In [499]:
```
# Feature Extraction - MFCC

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_mfcc1 = []
list_mfcc2 = []
list_mfcc3 = []
list_mfcc4 = []
list_mfcc5 = []
list_mfcc6 = []
list_mfcc7 = []
list_mfcc8 = []
list_mfcc9 = []
list_mfcc10 = []
list_mfcc11 = []
list_mfcc12 = []
list_mfcc13 = []
list_mfcc14 = []
list_mfcc15 = []
list_mfcc16 = []
list_mfcc17 = []
list_mfcc18 = []
list_mfcc19 = []
list_mfcc20 = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    mfccs = librosa.feature.mfcc(y=signal, n_mfcc=20, sr=sr)
    list_mfcc1.append(mfccs[0].mean())
    list_mfcc2.append(mfccs[1].mean())
    list_mfcc3.append(mfccs[2].mean())
    list_mfcc4.append(mfccs[3].mean())
    list_mfcc5.append(mfccs[4].mean())
    list_mfcc6.append(mfccs[5].mean())
    list_mfcc7.append(mfccs[6].mean())
    list_mfcc8.append(mfccs[7].mean())
    list_mfcc9.append(mfccs[8].mean())
    list_mfcc10.append(mfccs[9].mean())
    list_mfcc11.append(mfccs[10].mean())
    list_mfcc12.append(mfccs[11].mean())
    list_mfcc13.append(mfccs[12].mean())
    list_mfcc14.append(mfccs[13].mean())
    list_mfcc15.append(mfccs[14].mean())
    list_mfcc16.append(mfccs[15].mean())
    list_mfcc17.append(mfccs[16].mean())
    list_mfcc18.append(mfccs[17].mean())
    list_mfcc19.append(mfccs[18].mean())
    list_mfcc20.append(mfccs[19].mean())
    """
    delta_mfccs = librosa.feature.delta(mfccs,order=1)
    list_mfcc_der1.append(delta_mfccs.mean())
    delta2_mfccs = librosa.feature.delta(mfccs,order=2)
    list_mfcc_der2.append(delta2_mfccs.mean())"""
```

In [500]:
```
# feature - Mfcc coeff's
data['mfcc_mean1'] = pd.DataFrame(list_mfcc1)
data['mfcc_mean2'] = pd.DataFrame(list_mfcc2)
data['mfcc_mean3'] = pd.DataFrame(list_mfcc3)
data['mfcc_mean4'] = pd.DataFrame(list_mfcc4)
data['mfcc_mean5'] = pd.DataFrame(list_mfcc5)
data['mfcc_mean6'] = pd.DataFrame(list_mfcc6)
data['mfcc_mean7'] = pd.DataFrame(list_mfcc7)
data['mfcc_mean8'] = pd.DataFrame(list_mfcc8)
data['mfcc_mean9'] = pd.DataFrame(list_mfcc9)
data['mfcc_mean10'] = pd.DataFrame(list_mfcc10)
data['mfcc_mean11'] = pd.DataFrame(list_mfcc11)
data['mfcc_mean12'] = pd.DataFrame(list_mfcc12)
data['mfcc_mean13'] = pd.DataFrame(list_mfcc13)
data['mfcc_mean14'] = pd.DataFrame(list_mfcc14)
data['mfcc_mean15'] = pd.DataFrame(list_mfcc15)
data['mfcc_mean16'] = pd.DataFrame(list_mfcc16)
data['mfcc_mean17'] = pd.DataFrame(list_mfcc17)
data['mfcc_mean18'] = pd.DataFrame(list_mfcc18)
data['mfcc_mean19'] = pd.DataFrame(list_mfcc19)
data['mfcc_mean20'] = pd.DataFrame(list_mfcc20)
```

In [501]:
```
# Feature Extraction - Spectral Centroid

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_sc = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    signal_sc = librosa.feature.spectral_centroid(y=signal, sr=sr, n_fft=FR
    list_sc.append(signal_sc.mean())
```

In [502]:
```
# feature - spectral centroid
data['spectral_centroid_mean'] = pd.DataFrame(list_sc)
```

In [503]:
```
# Feature Extraction - Spectral Bandwidth

FRAME_SIZE = 2048
HOP_LENGTH = 512
list_spec_bw = []

for files in file_paths_list:
    signal, sr = librosa.load(files)
    signal_spec_bw = librosa.feature.spectral_bandwidth(y=signal, sr=sr, n_
    list_spec_bw.append(signal_spec_bw.mean())
```

In [504]:
```
# feature - spectral bandwidth
data['spectral_bandwidth_mean'] = pd.DataFrame(list_spec_bw)
```

In [505]:
```
# Feature Extraction - Tempo

list_tempo = []
for files in file_paths_list:
    signal, sr = librosa.load(files)
    onset_env = librosa.onset.onset_strength(y=signal, sr=sr)
    tempo = librosa.beat.tempo(onset_envelope=onset_env, sr=sr)
    list_tempo.append(tempo)
```

In [506]:
```
# feature - tempo
data['tempo'] = pd.DataFrame(list_tempo)
```

In [508]:
```
#  Feature Extraction - Chroma

list_chroma = []
for files in file_paths_list:
    signal, sr = librosa.load(files)
    chroma = librosa.feature.chroma_stft(y=signal, sr=sr)
    list_chroma.append(chroma.mean())
```

In [509]:
```
# feature - chroma
data['chroma_stft'] = pd.DataFrame(list_chroma)
```

In [510]:
```
# Feature Extraction - spectral roll off

list_spectral_rolloff = []
for files in file_paths_list:
    signal, sr = librosa.load(files)
    spectral_rolloff = librosa.feature.spectral_rolloff(y=signal, sr=sr)
    list_spectral_rolloff.append(spectral_rolloff.mean())
```

In [511]:
```
# feature - spectral rolloff
data['spectral_rolloff'] = pd.DataFrame(list_spectral_rolloff)
```

In [512]:
```
data['labels'] = labels_df['labels']
```

In [530]:
```
#data.to_csv('music_classification_feature_extraction_dataset.csv',index=Fa
```

In [531]:
```
#data.to_excel('music_classification_feature_extraction_dataset.xlsx',index
```

In [601]:
```
data = pd.read_csv('music_classification_feature_extraction_dataset.csv')
```

In [602]:
```
data.head()
```

Out[602]:

:0	spectral_centroid_mean	spectral_bandwidth_mean	tempo	chroma_stft	spectral_rolloff	labels
i7	1784.122641	2002.412407	123.046875	0.350129	3805.723030	blues
i4	1530.261767	2038.987608	107.666016	0.340849	3550.713616	blues
:8	1552.832481	1747.754087	161.499023	0.363538	3042.410115	blues
i6	1070.153418	1596.422565	172.265625	0.404854	2184.879029	blues
i5	1835.128513	1748.410759	135.999178	0.308526	3579.957471	blues

In [603]:
```
data.isna().sum()
```

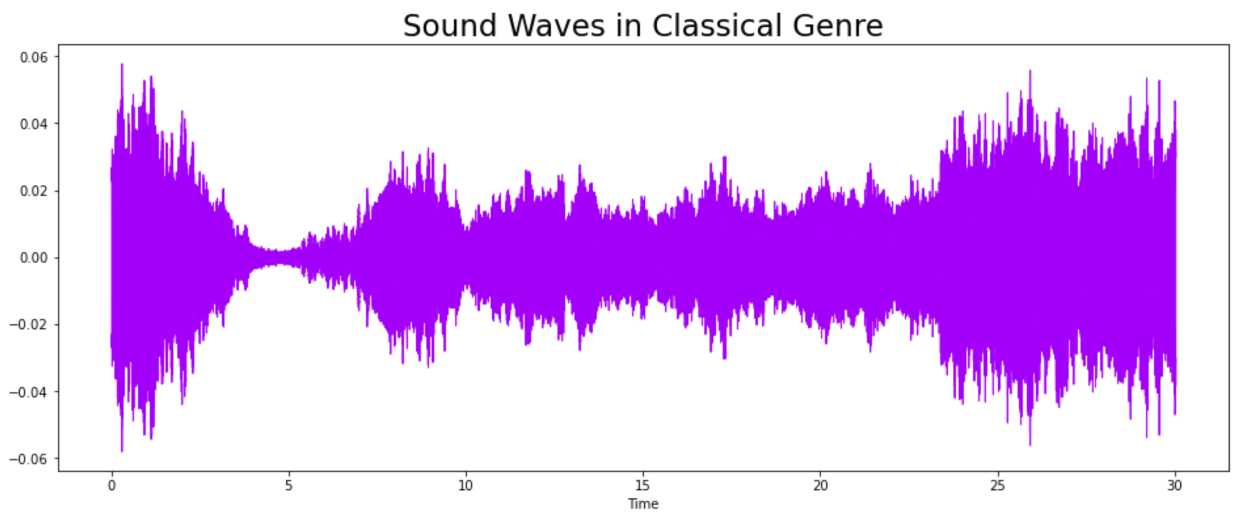
Out[603]:
```
amplitude_envelope_mean    0
rmse_mean                  0
zcr_mean                   0
spectrogram_mean           0
mel_spectrogram_mean       0
mfcc_mean1                 0
mfcc_mean2                 0
mfcc_mean3                 0
mfcc_mean4                 0
mfcc_mean5                 0
mfcc_mean6                 0
mfcc_mean7                 0
mfcc_mean8                 0
mfcc_mean9                 0
mfcc_mean10                0
mfcc_mean11                0
mfcc_mean12                0
mfcc_mean13                0
mfcc_mean14                0
mfcc_mean15                0
mfcc_mean16                0
mfcc_mean17                0
mfcc_mean18                0
mfcc_mean19                0
mfcc_mean20                0
spectral_centroid_mean     0
spectral_bandwidth_mean    0
tempo                      0
chroma_stft                0
spectral_rolloff           0
labels                     0
dtype: int64
```

In [604]:
```
data.columns
```

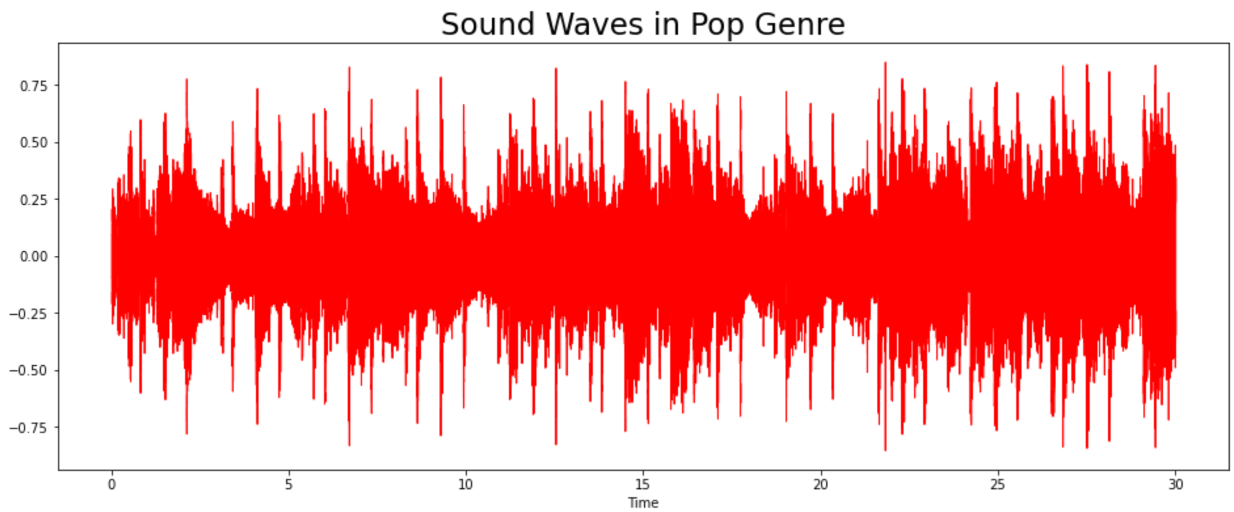
Out[604]:
```
Index(['amplitude_envelope_mean', 'rmse_mean', 'zcr_mean', 'spectrogram_m
ean',
       'mel_spectrogram_mean', 'mfcc_mean1', 'mfcc_mean2', 'mfcc_mean3',
       'mfcc_mean4', 'mfcc_mean5', 'mfcc_mean6', 'mfcc_mean7', 'mfcc_mean
8',
       'mfcc_mean9', 'mfcc_mean10', 'mfcc_mean11', 'mfcc_mean12',
       'mfcc_mean13', 'mfcc_mean14', 'mfcc_mean15', 'mfcc_mean16',
       'mfcc_mean17', 'mfcc_mean18', 'mfcc_mean19', 'mfcc_mean20',
       'spectral_centroid_mean', 'spectral_bandwidth_mean', 'tempo',
       'chroma_stft', 'spectral_rolloff', 'labels'],
      dtype='object')
```

In [605]:
```
#data.dropna(inplace = True)
```

In [675]:
```
classical, sr = librosa.load(file_paths_list[150])
plt.figure(figsize = (16, 6))
librosa.display.waveshow(y = classical, sr = sr, color = "#A300F9");
plt.title("Sound Waves in Classical Genre", fontsize = 23);
```

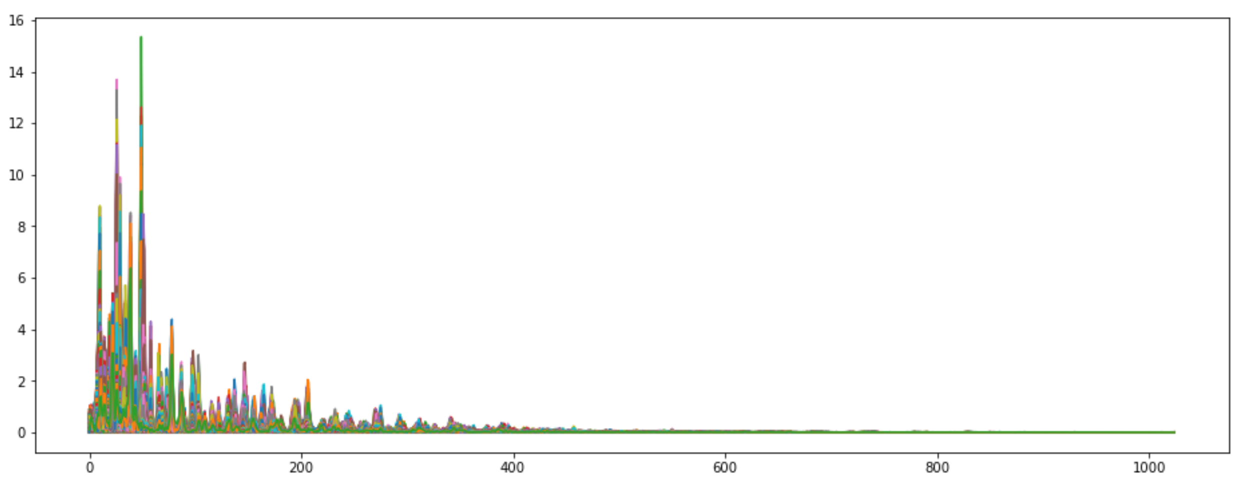


In [673]:
```
signal, sr = librosa.load(file_paths_list[702])
plt.figure(figsize = (16, 6))
librosa.display.waveshow(y = signal, sr = sr, color = "red");
plt.title("Sound Waves in Pop Genre", fontsize = 23);
```

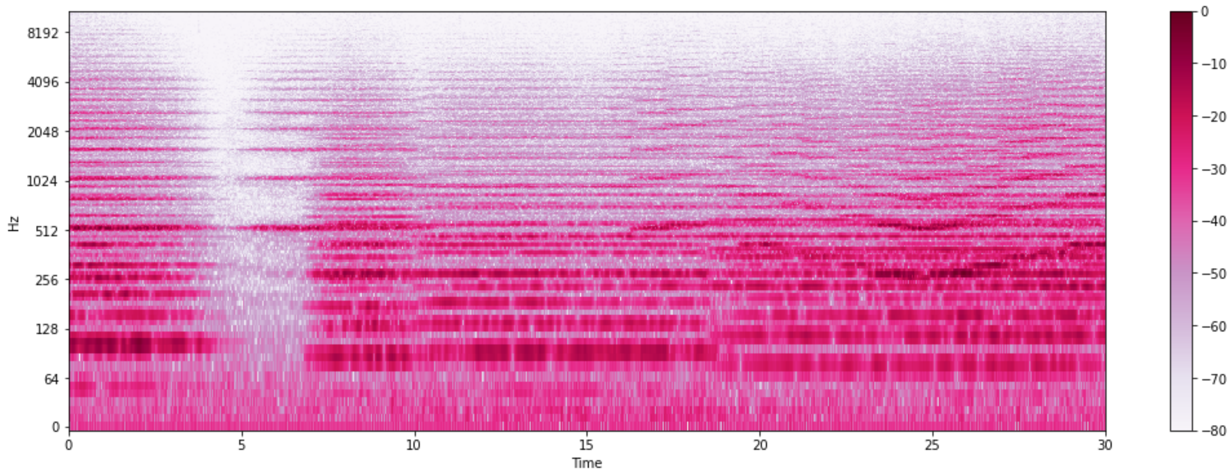


In [678]:
```
# Converting a Time domain classical signal to Frequency Domain by applying
n_fft = 2048
hop_length = 512

# Short-time Fourier transform (STFT)
signal_D = np.abs(librosa.stft(y=classical, n_fft = n_fft, hop_length = hop

print('Shape of D object:', np.shape(signal_D))
plt.figure(figsize = (16, 6))
plt.plot(signal_D);
```

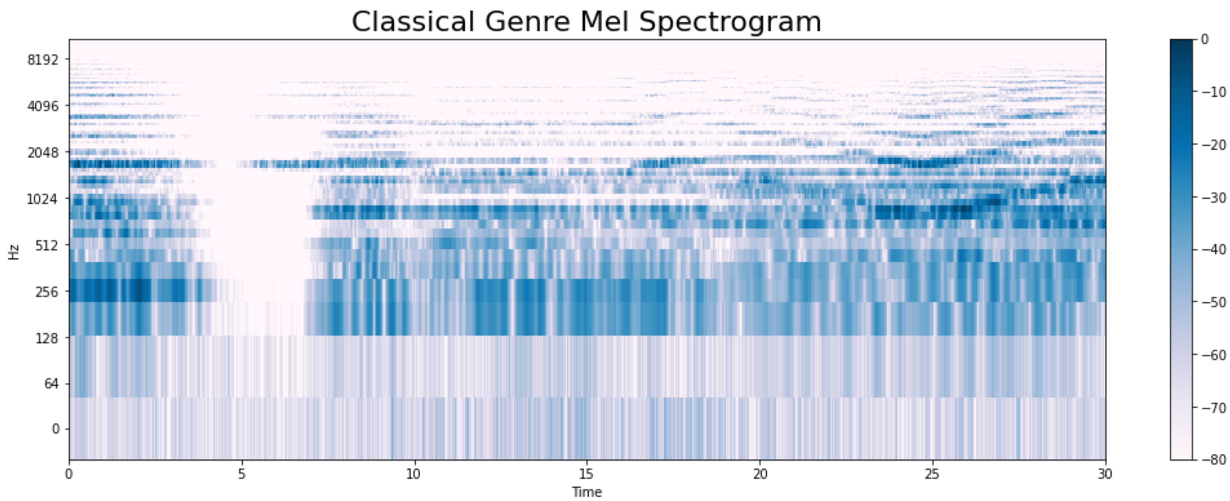
```
Shape of D object: (1025, 1293)
```



In [683]:
```
# Convert an amplitude spectrogram to Decibels-scaled spectrogram.
DB = librosa.amplitude_to_db(signal_D, ref = np.max)

# Creating the Spectogram
plt.figure(figsize = (18, 6))
librosa.display.specshow(DB, sr = sr, hop_length = hop_length, x_axis = 'ti
                         cmap = 'PuRd')
plt.colorbar();
```

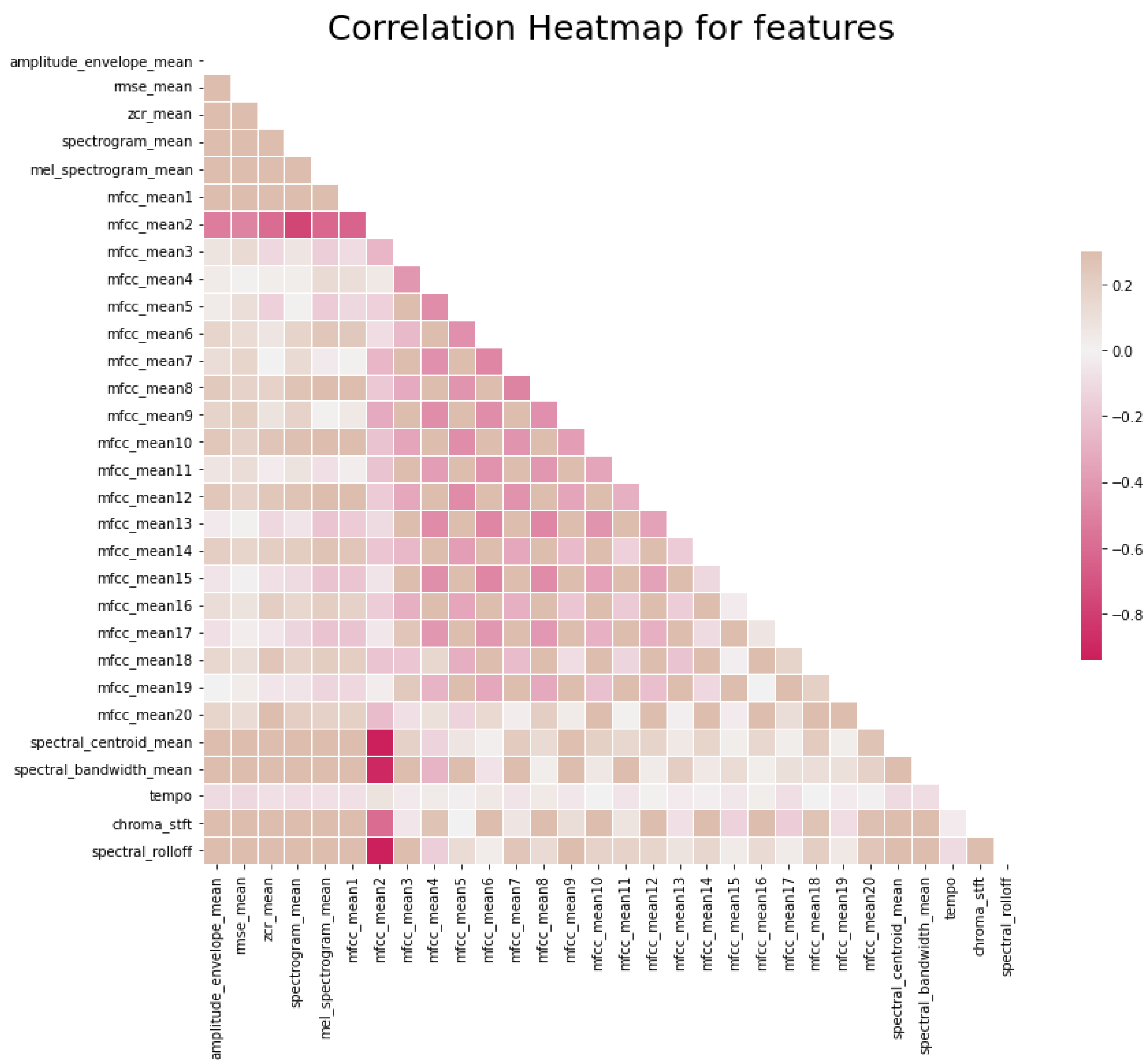


In [684]:
```
y, _ = librosa.effects.trim(classical)

S_Mel = librosa.feature.melspectrogram(y, sr=sr)
S_DB = librosa.amplitude_to_db(S_Mel, ref=np.max)
plt.figure(figsize = (18, 6))
librosa.display.specshow(S_DB, sr=sr, hop_length=hop_length, x_axis = 'time
                        cmap = 'PuBu');
plt.colorbar();
plt.title("Classical Genre Mel Spectrogram", fontsize = 22);
```



In [606]:
```
from sklearn import preprocessing

data = data.iloc[0:, 0:]
y = data['labels']
X = data.drop(columns=['labels'], axis=1)

#### NORMALIZE X ####
cols = X.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(X)
X = pd.DataFrame(np_scaled, columns = cols)
```

In [699]:
```
# Computing the Correlation Matrix
spike_cols = [col for col in X.columns]
corr = data[spike_cols].corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=np.bool))

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

# Generate a custom diverging colormap
cmap = sns.diverging_palette(0, 25, as_cmap=True, s = 90, l = 45, n = 5)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

plt.title('Correlation Heatmap for features', fontsize = 25)
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10);
plt.savefig("Corr_Heatmap.jpg")
```



In [607]:
```
#!pip install shap
```

In [693]:
```
#models to try
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

#graph of model
import graphviz

#metrics
from sklearn import metrics

#shap
import shap
```

In [609]:
```
X
```

Out[609]:

	amplitude_envelope_mean	rmse_mean	zcr_mean	spectrogram_mean	mel_spectrogram_mean
0	0.383173	0.318555	0.242545	0.647535	0.776501
1	0.291588	0.230971	0.135778	0.523950	0.615278
2	0.516567	0.433603	0.215844	0.651353	0.822990
3	0.350993	0.345574	0.045909	0.519804	0.607916
4	0.334860	0.219619	0.315353	0.566349	0.728934
...	...	...	...	...	...
993	0.246465	0.189066	0.494012	0.580589	0.719148
994	0.246065	0.181360	0.266989	0.585607	0.729288
995	0.264225	0.194580	0.300344	0.601801	0.759021
996	0.237707	0.199792	0.395857	0.483570	0.590430
997	0.151675	0.125277	0.106855	0.433233	0.597990

998 rows × 30 columns

In [610]:
```
X['zcr_mean'].value_counts()
```

Out[610]:
```
0.296344    2
0.307134    2
0.669183    2
0.197732    2
0.572590    2
           ..
0.601165    1
0.401505    1
0.406392    1
0.629081    1
0.106855    1
Name: zcr_mean, Length: 982, dtype: int64
```

In [611]:
```
y_float = np.where(y=='blues', 0.0, y)
y_float = np.where(y_float=='classical', 1.0, y_float)
y_float = np.where(y_float=='country', 2.0, y_float)
y_float = np.where(y_float=='disco', 3.0, y_float)
y_float = np.where(y_float=='hiphop', 4.0, y_float)
y_float = np.where(y_float=='jazz', 5.0, y_float)
y_float = np.where(y_float=='metal', 6.0, y_float)
y_float = np.where(y_float=='pop', 7.0, y_float)
y_float = np.where(y_float=='reggae', 8.0, y_float)
y_float = np.where(y_float=='rock', 9.0, y_float).astype('float64')
```

In [612]:
```
#!pip install imblearn
```

In [613]:
```
from imblearn.over_sampling import SMOTE
```

In [614]:
```
#splits data into train, validation, and test set
X_train, X_test, y_train, y_test = train_test_split(X, y_float, test_size=0
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0
#SMOTE for class balancing
sm = SMOTE(random_state=8)

#create new training set with SMOTE object
X_bal, y_bal = sm.fit_resample(X_train, y_train)
```

A voting classifier is a machine learning model that gains experience by training on a collection of several models and forecasts an output (class) based on the class with the highest likelihood of being the output.
To predict the output class based on the highest majority of votes, it simply averages the results of each classifier that was passed into the voting classifier. The concept is to build a single model that learns from these models and predicts output based on their aggregate majority of voting for each output class, rather than developing separate dedicated models and calculating the accuracy for each of them.

In [615]:
```
# OPTUNA objective function

def objective(trial):
    #logistic regression
    lr_penalty = trial.suggest_categorical('lr_penalty', ['l1', 'l2', 'elas
    lr_l1_ratio = None
    if lr_penalty == 'l1':
        lr_solver = trial.suggest_categorical('lr_solver1', ['liblinear', '
    elif lr_penalty == 'l2':
        lr_solver = trial.suggest_categorical('lr_solver2', ['newton-cg', '
    else:
        lr_solver = 'saga'
        lr_l1_ratio = trial.suggest_uniform('lr_l1_ratio', 0.0, 1.0)

    lr_tol = trial.suggest_uniform('lr_tol', 1e-5, 1e-2)
    lr_C = trial.suggest_uniform('lr_C', 0.0, 1.0)

    lr = LogisticRegression(
        penalty=lr_penalty,
        tol=lr_tol,
        C=lr_C,
        solver=lr_solver,
        l1_ratio=lr_l1_ratio
    )

    #KNN
    knn_neighbors = trial.suggest_int('knn_neighbors', 2, 100)
    knn_weights = trial.suggest_categorical('knn_weights', ['uniform', 'dis
    knn_p = trial.suggest_categorical('knn_p', [1, 2])

    knn = KNeighborsClassifier(
        n_neighbors=knn_neighbors,
        weights=knn_weights,
        p=knn_p
    )

    #SVM
    svm_C = trial.suggest_uniform('svm_C', 0.0, 1.0)
    svm_kernel = trial.suggest_categorical('svm_kernel', ['poly', 'rbf'])
    svm_degree = 3
    if svm_kernel == 'poly':
        svm_degree = trial.suggest_int('svm_degree', 1, 10)
    svm_tol = trial.suggest_uniform('svm_tol', 1e-5, 1e-2)

    svm = SVC(
        C=svm_C,
        kernel=svm_kernel,
        degree=svm_degree,
        tol=svm_tol
    )

    #random forest
    rf_estimators = trial.suggest_int('rf_estimators', 1, 500)
    rf_criterion = trial.suggest_categorical('rf_criterion', ['entropy', 'g
    rf_max_depth = trial.suggest_int('rf_max_depth', 1, 100)
    rf_min_samples_split = trial.suggest_int('rf_min_samples_split', 2, 50)
```

```
    rf_min_samples_leaf = trial.suggest_int('rf_min_samples_leaf', 1, 25)

    rf = RandomForestClassifier(
        n_estimators=rf_estimators,
        criterion=rf_criterion,
        max_depth=rf_max_depth,
        min_samples_split=rf_min_samples_split,
        min_samples_leaf=rf_min_samples_leaf
    )

    #naive bayes
    nb_smoothing = trial.suggest_uniform('nb_smoothing', 1e-10, 1e-6)
    nb = GaussianNB(var_smoothing=nb_smoothing)

    #ensemble model
    lr_w = trial.suggest_uniform('lr_w', 0.0, 1.0)
    knn_w = trial.suggest_uniform('knn_w', 0.0, 1.0)
    svm_w = trial.suggest_uniform('svm_w', 0.0, 1.0)
    rf_w = trial.suggest_uniform('rf_w', 0.0, 1.0)
    nb_w = trial.suggest_uniform('nb_w', 0.0, 1.0)

    vc = VotingClassifier(estimators=[
        ('lr', lr),
        ('knn', knn),
        ('svm', svm),
        ('rf', rf),
        ('nb', nb)],
        weights=[lr_w, knn_w, svm_w, rf_w, nb_w])

    vc.fit(X_bal, y_bal)
    preds = vc.predict(X_val)

    acc = metrics.accuracy_score(y_val, preds)

    return acc
```

In [616]:
```
#!pip install optuna
```

Verbosity level is just related to logging. In unit tests you find it for the logging of the information. It is more pythonic to use levels as constant names(logging.INFO, logging.DEBUG, optuna.logging.CRITICAL, optuna.logging.FATAL) rather than numbers.

In [617]:
```
import optuna

optuna.logging.set_verbosity(optuna.logging.ERROR)

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

In [618]:

```
#recreates a model from the best hyperparameters:

def create_model(best_params):

    try:
        l1_ratio = best_params['lr_l1_ratio']
    except:
        l1_ratio = None
    try:
        solver = best_params['lr_solver1']
    except:
        try:
            solver = best_params['lr_solver2']
        except:
            solver = 'saga'

    lr = LogisticRegression(
        penalty=best_params['lr_penalty'],
        tol=best_params['lr_tol'],
        C=best_params['lr_C'],
        l1_ratio=l1_ratio,
        solver=solver
    )

    #KNN

    knn = KNeighborsClassifier(
        n_neighbors=best_params['knn_neighbors'],
        weights=best_params['knn_weights'],
        p=best_params['knn_p']
    )

    #SVM
    try:
        svm_degree = best_params['svm_degree']
    except:
        svm_degree=3

    svm = SVC(
        C=best_params['svm_C'],
        kernel=best_params['svm_kernel'],
        degree=svm_degree,
        tol=best_params['svm_tol']
    )

    #random forest
    rf = RandomForestClassifier(
        n_estimators=best_params['rf_estimators'],
        criterion=best_params['rf_criterion'],
        max_depth=best_params['rf_max_depth'],
        min_samples_split=best_params['rf_min_samples_split'],
        min_samples_leaf=best_params['rf_min_samples_leaf']
    )

    #naive bayes
    nb = GaussianNB(var_smoothing=best_params['nb_smoothing'])
```

```
    #ensemble model
    vc = VotingClassifier(estimators=[
        ('lr', lr),
        ('knn', knn),
        ('svm', svm),
        ('rf', rf),
        ('nb', nb)],
      weights=[
          best_params['lr_w'],
          best_params['knn_w'],
          best_params['svm_w'],
          best_params['rf_w'],
          best_params['nb_w']]
     )

    vc.fit(X_bal, y_bal)

    return vc
```

In [619]:

```
#ensemble model with best hyperparameters
model = create_model(study.best_params)
```

In [620]:

```
print(model)
```

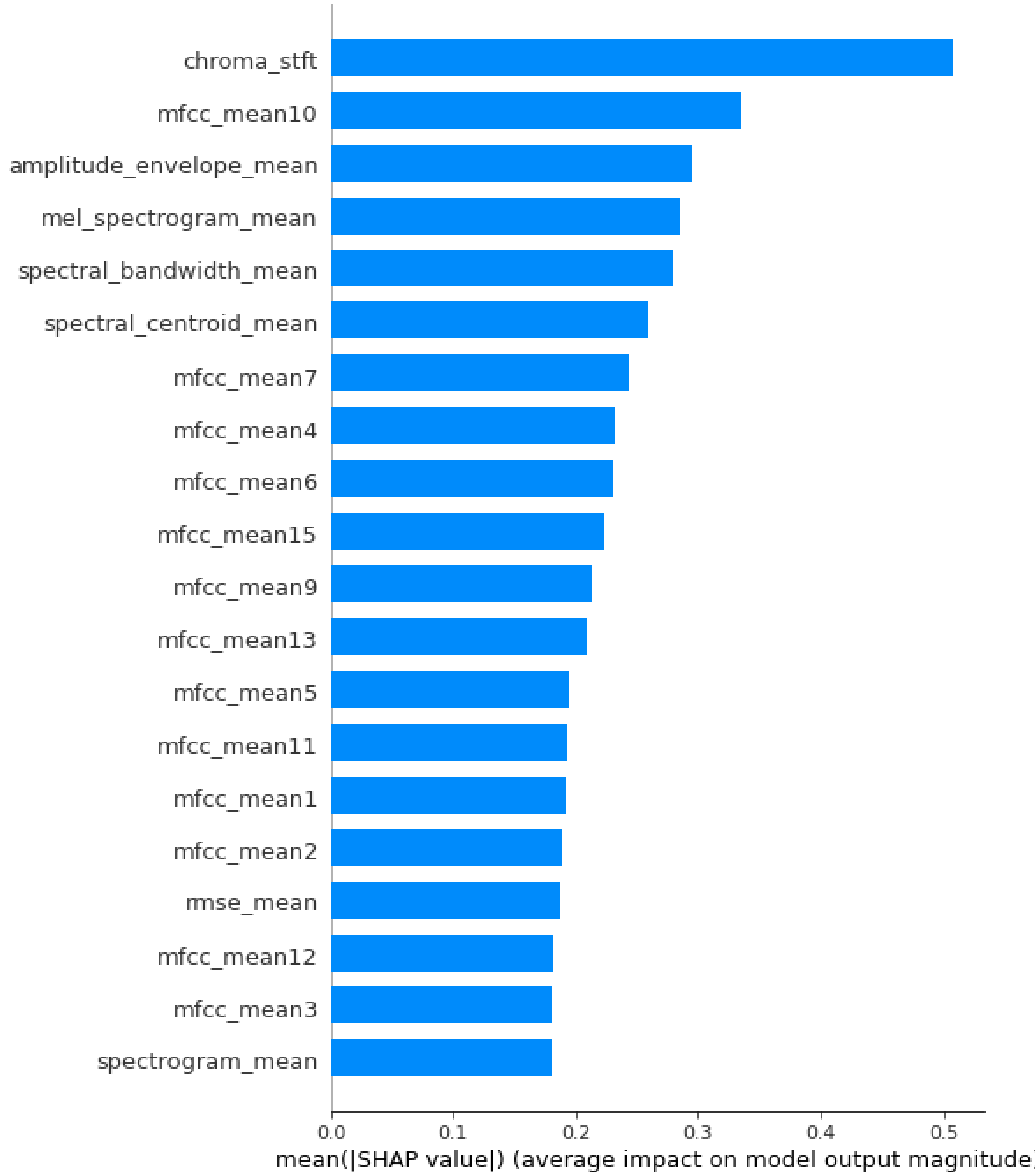
```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(C=0.7085703908772655,
                                                 solver='sag',
                                                 tol=0.00671663551223457
5)),
                             ('knn', KNeighborsClassifier(n_neighbors=
2)),
                             ('svm',
                              SVC(C=0.6755784145957529, kernel='poly',
                                  tol=0.0001469683318744772)),
                             ('rf',
                              RandomForestClassifier(max_depth=94,
                                                     min_samples_leaf=22,
                                                     min_samples_split=3
8,

                                                     n_estimators=455)),
                             ('nb',
                              GaussianNB(var_smoothing=7.550717522272895e
-07))],
                 weights=[0.05282823134718598, 0.31836230122050135,
                          0.5675777326145544, 0.11996455238545856,
                          0.12124174809570648])
```

In [621]:

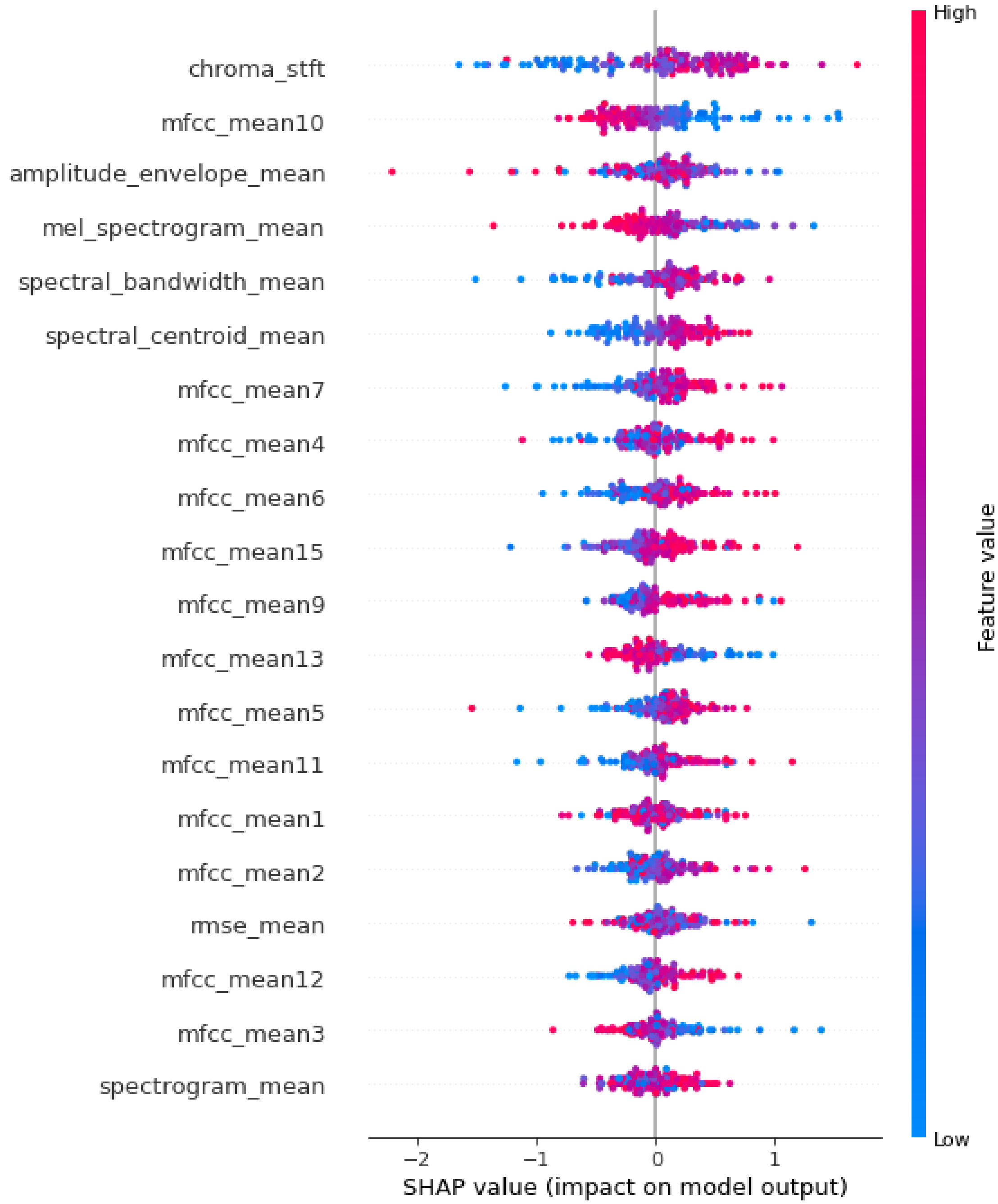
```
#creates shap explainer
feature_names = list(data.columns)[1:2] + list(data.columns)[3:] + [list(da
explainer = shap.Explainer(model.predict, X_train, feature_names=feature_na
shap_values = explainer(X_test)
```

```
Permutation explainer: 151it [09:26,  3.83s/it]
```

In [622]:
```
#shap plots the importance of each feature
shap.summary_plot(shap_values, X_test, plot_type="bar")
```



In [623]:
```
#plots importance of each feature
shap.summary_plot(shap_values, X_test)
```



In [624]:
```
# Accuracy for Training Dataset
test_preds = model.predict(X_train)
print('Test Accuracy:', round(metrics.accuracy_score(y_train, test_preds),
```

```
Test Accuracy: 0.95
```

In [625]:
```
# Accuracy for Test Dataset
test_preds = model.predict(X_test)
print('Test Accuracy:', round(metrics.accuracy_score(y_test, test_preds), 2
```

```
Test Accuracy: 0.69
```

In [689]:
```
print('Confusion Matrix:', metrics.confusion_matrix(y_test, test_preds))
```

```
Confusion Matrix: [[ 8  0  1  1  0  0  1  0  0  2]
 [ 0 10  0  0  0  0  0  0  0  0]
 [ 0  0 12  2  1  1  0  1  0  0]
 [ 0  1  1  7  1  0  1  0  1  4]
 [ 0  0  1  2  6  0  0  1  1  3]
 [ 0  1  0  0  0  9  0  0  1  0]
 [ 0  0  1  0  0  0 18  0  0  1]
 [ 0  0  1  0  1  1  0 16  1  0]
 [ 0  0  2  1  1  0  0  1 13  1]
 [ 2  0  1  0  0  0  1  0  1  5]]
```

'blues' - 0.0 'classical'- 1.0 'country' - 2.0 'disco' - 3.0 'hiphop' - 4.0 'jazz' - 5.0 'metal' - 6.0 'pop' - 7.0 'reggae' - 8.0 'rock' - 9.0

In [698]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

         0.0       0.80      0.62      0.70        13
         1.0       0.83      1.00      0.91        10
         2.0       0.60      0.71      0.65        17
         3.0       0.54      0.44      0.48        16
         4.0       0.60      0.43      0.50        14
         5.0       0.82      0.82      0.82        11
         6.0       0.86      0.90      0.88        20
         7.0       0.84      0.80      0.82        20
         8.0       0.72      0.68      0.70        19
         9.0       0.31      0.50      0.38        10

    accuracy                           0.69       150
   macro avg       0.69      0.69      0.68       150
weighted avg       0.71      0.69      0.69       150
```

In [690]:
```
confusion_matrix = metrics.confusion_matrix(y_test, test_preds)
```

In [695]:
```
import sklearn
sklearn.metrics.ConfusionMatrixDisplay(confusion_matrix)
```

Out[695]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd97
c6f0040>
```

In [626]:
```
# print the predicted values
print(test_preds)
```

```
[9. 2. 3. 2. 4. 0. 4. 9. 5. 3. 9. 3. 7. 9. 0. 6. 5. 6. 8. 2. 2. 1. 6. 9.
 7. 7. 7. 7. 6. 5. 7. 4. 8. 9. 3. 1. 8. 0. 2. 9. 9. 2. 6. 6. 0. 7. 1. 8.
 3. 9. 5. 6. 2. 2. 8. 8. 5. 1. 1. 7. 9. 3. 2. 8. 8. 6. 1. 3. 6. 6. 1. 8.
 7. 9. 5. 7. 8. 6. 7. 6. 8. 2. 7. 6. 0. 4. 2. 6. 2. 9. 3. 1. 6. 3. 4. 5.
 2. 1. 3. 7. 7. 7. 7. 0. 6. 3. 3. 9. 9. 6. 4. 2. 0. 0. 9. 4. 0. 2. 8. 7.
 5. 8. 6. 8. 8. 7. 5. 6. 5. 2. 6. 3. 2. 4. 8. 9. 7. 4. 2. 2. 6. 4. 5. 2.
 8. 1. 1. 0. 8. 1.]
```

Testing the model with some Telugu Songs:

In [627]:
```
test_file_names = []
test_file_paths = []

basedir = os.getcwd()
path_of_the_directory = basedir+ "/test_music"
#print("Files and directories in a specified path:")
test_list_of_files = sorted( filter( os.path.isfile,
                        glob.glob(path_of_the_directory + '/**/*', recursive=Tr
test_file_name = sorted( filter( lambda x: os.path.isfile(os.path.join(path
                            os.listdir(path_of_the_directory) ) )

test_file_names.append(test_file_name)
test_file_paths.append(test_list_of_files)
```

In [628]:
```
test_file_paths_list = list(chain.from_iterable(test_file_paths))
```

In [629]:
```
test_file_paths_list
```

Out[629]:
```
['/Users/bakhtsinghbasaram/Downloads/INFO 5502/final_project/Data/test_mu
sic/Jai+Balayya_out.wav',
 '/Users/bakhtsinghbasaram/Downloads/INFO 5502/final_project/Data/test_mu
sic/Nannaya+Raasina_out.wav',
 '/Users/bakhtsinghbasaram/Downloads/INFO 5502/final_project/Data/test_mu
sic/Pilla+Padesaave_out.wav',
 '/Users/bakhtsinghbasaram/Downloads/INFO 5502/final_project/Data/test_mu
sic/Pranam+Pothunna_out.wav']
```

In [631]:
```
# Feature Extraction - Amplitude Envelope

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_amplitude_envelope = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    ae = feature_amplitude_envelope(signal, FRAME_SIZE, HOP_LENGTH)
    test_list_amplitude_envelope.append(ae.mean())
```

In [632]:
```
# feature - amplitude envelope dataframe
test_data = pd.DataFrame(test_list_amplitude_envelope)
```

In [633]:
```
# Feature Extraction - Root Mean Square Energy

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_rmse = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    rmse = feature_rmse(signal, FRAME_SIZE, HOP_LENGTH)
    test_list_rmse.append(rmse.mean())
```

In [634]:
```
# feature - rmse dataframe
test_data['rmse_mean'] = pd.DataFrame(test_list_rmse)
```

In [635]:
```
# Feature Extraction - Zero Crossing Rate

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_zcr = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    zcr = librosa.feature.zero_crossing_rate(signal, frame_length=FRAME_SIZ
    test_list_zcr.append(zcr.mean())
```

In [636]:
```
# feature - zcr dataframe
test_data['zcr_mean'] = pd.DataFrame(test_list_zcr)
```

In [637]:
```
# Feature Extraction - Spectrogram

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_spectrogram = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    S_scale = librosa.stft(signal, n_fft=FRAME_SIZE, hop_length=HOP_LENGTH)
    Y_scale = np.abs(S_scale) ** 2
    Y_log_scale = librosa.power_to_db(Y_scale)
    test_list_spectrogram.append(Y_log_scale.mean())
```

In [638]:
```
# feature - Spectrogram dataframe
test_data['spectrogram_mean'] = pd.DataFrame(test_list_spectrogram)
```

In [639]:
```
# Feature Extraction - Mel Spectrogram

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_mel_spectrogram = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    mel_spectrogram = librosa.feature.melspectrogram(signal, sr=sr, n_fft=F
    log_mel_spectrogram = librosa.power_to_db(mel_spectrogram)
    test_list_mel_spectrogram.append(log_mel_spectrogram.mean())
```

In [640]:
```
# feature - Mel Spectrogram dataframe
test_data['mel_spectrogram_mean'] = pd.DataFrame(test_list_mel_spectrogram)
```

In [641]:
```
# Feature Extraction - MFCC

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_mfcc1 = []
test_list_mfcc2 = []
test_list_mfcc3 = []
test_list_mfcc4 = []
test_list_mfcc5 = []
test_list_mfcc6 = []
test_list_mfcc7 = []
test_list_mfcc8 = []
test_list_mfcc9 = []
test_list_mfcc10 = []
test_list_mfcc11 = []
test_list_mfcc12 = []
test_list_mfcc13 = []
test_list_mfcc14 = []
test_list_mfcc15 = []
test_list_mfcc16 = []
test_list_mfcc17 = []
test_list_mfcc18 = []
test_list_mfcc19 = []
test_list_mfcc20 = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    mfccs = librosa.feature.mfcc(y=signal, n_mfcc=20, sr=sr)
    test_list_mfcc1.append(mfccs[0].mean())
    test_list_mfcc2.append(mfccs[1].mean())
    test_list_mfcc3.append(mfccs[2].mean())
    test_list_mfcc4.append(mfccs[3].mean())
    test_list_mfcc5.append(mfccs[4].mean())
    test_list_mfcc6.append(mfccs[5].mean())
    test_list_mfcc7.append(mfccs[6].mean())
    test_list_mfcc8.append(mfccs[7].mean())
    test_list_mfcc9.append(mfccs[8].mean())
    test_list_mfcc10.append(mfccs[9].mean())
    test_list_mfcc11.append(mfccs[10].mean())
    test_list_mfcc12.append(mfccs[11].mean())
    test_list_mfcc13.append(mfccs[12].mean())
    test_list_mfcc14.append(mfccs[13].mean())
    test_list_mfcc15.append(mfccs[14].mean())
    test_list_mfcc16.append(mfccs[15].mean())
    test_list_mfcc17.append(mfccs[16].mean())
    test_list_mfcc18.append(mfccs[17].mean())
    test_list_mfcc19.append(mfccs[18].mean())
    test_list_mfcc20.append(mfccs[19].mean())
```

In [642]:
```
# feature - Mfcc coeff's
test_data['mfcc_mean1'] = pd.DataFrame(test_list_mfcc1)
test_data['mfcc_mean2'] = pd.DataFrame(test_list_mfcc2)
test_data['mfcc_mean3'] = pd.DataFrame(test_list_mfcc3)
test_data['mfcc_mean4'] = pd.DataFrame(test_list_mfcc4)
test_data['mfcc_mean5'] = pd.DataFrame(test_list_mfcc5)
test_data['mfcc_mean6'] = pd.DataFrame(test_list_mfcc6)
test_data['mfcc_mean7'] = pd.DataFrame(test_list_mfcc7)
test_data['mfcc_mean8'] = pd.DataFrame(test_list_mfcc8)
test_data['mfcc_mean9'] = pd.DataFrame(test_list_mfcc9)
test_data['mfcc_mean10'] = pd.DataFrame(test_list_mfcc10)
test_data['mfcc_mean11'] = pd.DataFrame(test_list_mfcc11)
test_data['mfcc_mean12'] = pd.DataFrame(test_list_mfcc12)
test_data['mfcc_mean13'] = pd.DataFrame(test_list_mfcc13)
test_data['mfcc_mean14'] = pd.DataFrame(test_list_mfcc14)
test_data['mfcc_mean15'] = pd.DataFrame(test_list_mfcc15)
test_data['mfcc_mean16'] = pd.DataFrame(test_list_mfcc16)
test_data['mfcc_mean17'] = pd.DataFrame(test_list_mfcc17)
test_data['mfcc_mean18'] = pd.DataFrame(test_list_mfcc18)
test_data['mfcc_mean19'] = pd.DataFrame(test_list_mfcc19)
test_data['mfcc_mean20'] = pd.DataFrame(test_list_mfcc20)
```

In [643]:
```
# Feature Extraction - Spectral Centroid

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_sc = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    signal_sc = librosa.feature.spectral_centroid(y=signal, sr=sr, n_fft=FR
    test_list_sc.append(signal_sc.mean())
```

In [644]:
```
# feature - spectral centroid
test_data['spectral_centroid_mean'] = pd.DataFrame(test_list_sc)
```

In [645]:
```
# Feature Extraction - Spectral Bandwidth

FRAME_SIZE = 2048
HOP_LENGTH = 512
test_list_spec_bw = []

for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    signal_spec_bw = librosa.feature.spectral_bandwidth(y=signal, sr=sr, n_
    test_list_spec_bw.append(signal_spec_bw.mean())
```

In [646]:
```
# feature - spectral bandwidth
test_data['spectral_bandwidth_mean'] = pd.DataFrame(test_list_spec_bw)
```

In [647]:
```
# Feature Extraction - Tempo

test_list_tempo = []
for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    onset_env = librosa.onset.onset_strength(y=signal, sr=sr)
    tempo = librosa.beat.tempo(onset_envelope=onset_env, sr=sr)
    test_list_tempo.append(tempo)
```

In [648]:
```
# feature - tempo
test_data['tempo'] = pd.DataFrame(test_list_tempo)
```

In [649]:
```
#  Feature Extraction - Chroma

test_list_chroma = []
for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    chroma = librosa.feature.chroma_stft(y=signal, sr=sr)
    test_list_chroma.append(chroma.mean())
```

In [650]:
```

# feature - chroma
test_data['chroma_stft'] = pd.DataFrame(test_list_chroma)
```

In [651]:
```

# Feature Extraction - spectral roll off

test_list_spectral_rolloff = []
for files in test_file_paths_list:
    signal, sr = librosa.load(files)
    spectral_rolloff = librosa.feature.spectral_rolloff(y=signal, sr=sr)
    test_list_spectral_rolloff.append(spectral_rolloff.mean())
```

In [652]:
```
# feature - spectral rolloff
test_data['spectral_rolloff'] = pd.DataFrame(test_list_spectral_rolloff)
```

In [653]:
```
#test_data.to_csv('test_music_classification_feature_extraction_dataset.csv
```

In [654]:
```
Xx = test_data

#### NORMALIZE X ####
cols_test = Xx.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(Xx)
Xx = pd.DataFrame(np_scaled, columns = cols_test)
```

In [655]:
```
X.columns
```

Out[655]:
```
Index(['amplitude_envelope_mean', 'rmse_mean', 'zcr_mean', 'spectrogram_m
ean',
       'mel_spectrogram_mean', 'mfcc_mean1', 'mfcc_mean2', 'mfcc_mean3',
       'mfcc_mean4', 'mfcc_mean5', 'mfcc_mean6', 'mfcc_mean7', 'mfcc_mean
8',
       'mfcc_mean9', 'mfcc_mean10', 'mfcc_mean11', 'mfcc_mean12',
       'mfcc_mean13', 'mfcc_mean14', 'mfcc_mean15', 'mfcc_mean16',
       'mfcc_mean17', 'mfcc_mean18', 'mfcc_mean19', 'mfcc_mean20',
       'spectral_centroid_mean', 'spectral_bandwidth_mean', 'tempo',
       'chroma_stft', 'spectral_rolloff'],
      dtype='object')
```

In [656]:
```
Xx.columns
```

Out[656]:
```
Index([                        0,               'rmse_mean',
                      'zcr_mean',        'spectrogram_mean',
          'mel_spectrogram_mean',              'mfcc_mean1',
                    'mfcc_mean2',              'mfcc_mean3',
                    'mfcc_mean4',              'mfcc_mean5',
                    'mfcc_mean6',              'mfcc_mean7',
                    'mfcc_mean8',              'mfcc_mean9',
                   'mfcc_mean10',             'mfcc_mean11',
                   'mfcc_mean12',             'mfcc_mean13',
                   'mfcc_mean14',             'mfcc_mean15',
                   'mfcc_mean16',             'mfcc_mean17',
                   'mfcc_mean18',             'mfcc_mean19',
                   'mfcc_mean20',  'spectral_centroid_mean',
       'spectral_bandwidth_mean',                   'tempo',
                   'chroma_stft',        'spectral_rolloff'],
      dtype='object')
```

In [657]:
```
#displays the final evaluation
test_preds_x = model.predict(Xx)
#print('Test Accuracy:', round(metrics.accuracy_score(y_test, test_preds_x)
```

'blues' - 0.0 'classical'- 1.0 'country' - 2.0 'disco' - 3.0 'hiphop' - 4.0 'jazz' - 5.0 'metal' - 6.0 'pop' - 7.0 'reggae' - 8.0 'rock' - 9.0

In [658]:
```
ipd.Audio(test_file_paths_list[0])
```

Out[658]:

0:23 / 0:30

In [659]:
```
ipd.Audio(test_file_paths_list[1])
```

Out[659]:

0:20 / 0:30

In [660]:
```
ipd.Audio(test_file_paths_list[2])
```

Out[660]:

0:23 / 0:30

In [661]:
```
ipd.Audio(test_file_paths_list[3])
```

Out[661]:

0:27 / 0:30

In [706]:
```
test_prediction = pd.DataFrame(test_preds_x)
```

Out[706]: array([6., 0., 1., 0.])

In [708]:
```
telugu_songs = ["Jai Balayya","Nannaya Rasina","Pilla Padesaave","Pranam Po
predicted_labels = ["Metal","Blues","Classical","Blues"]
final_result = pd.DataFrame(telugu_songs,columns=["Songs_tested"])
final_result['Test_prediction'] = pd.DataFrame(test_preds_x)
final_result['Predicted_labels'] = pd.DataFrame(predicted_labels)
```

In [709]:
```
final_result
```

Out[709]:

	Songs_tested	Test_prediction	Predicted_labels
0	Jai Balayya	6.0	Metal
1	Nannaya Rasina	0.0	Blues
2	Pilla Padesaave	1.0	Classical
3	Pranam Pothunna	0.0	Blues

In [662]:
```
# print the predicted values
print(test_preds_x)
```

```
[6. 0. 1. 0.]
```

In [663]:
```
test_data
```

Out[663]:

	0	rmse_mean	zcr_mean	spectrogram_mean	mel_spectrogram_mean	mfcc_mean1	mfcc
0	0.560669	0.171383	0.120974	-3.318819	-1.761909	-49.139118	89
1	0.413626	0.153445	0.045321	-16.191626	-10.776351	-169.997925	127
2	0.201786	0.076805	0.100578	-20.083471	-19.903969	-262.905334	83
3	0.584619	0.220116	0.037166	-18.037260	-14.102221	-199.374390	152

4 rows × 30 columns

In [ ]:

```

```