

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [ ]: # Usual Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import os
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn

# Librosa (the mother of audio files)
import librosa
import librosa.display
import IPython.display as ipd
import warnings
warnings.filterwarnings('ignore')
```

Initial Audio Analysis

```
In [ ]: #loading audio file
y, sr = librosa.load('/content/drive/MyDrive/genres_original/blues/blues.00000

print('\nNumerical Features :', y)
print('\n shape of the converted Audio files :', np.shape(y))
print('\n Sample Rate (Hz):', sr)

print('Length of the Audio File in seconds:', len(y)/sr)
```

Numerical Features : [0.00732422 0.01660156 0.00762939 ... -0.05560303 -0.06106567
-0.06417847]

shape of the converted Audio files : (661794,)

Sample Rate (Hz): 22050

Length of the Audio File in seconds: 30.013333333333332

```
In [ ]: #Trimming silence
audio_file, _ = librosa.effects.trim(y)

print('Audio File:', audio_file, '\n')
print('Audio File shape:', np.shape(audio_file))
```

Audio File: [0.00732422 0.01660156 0.00762939 ... -0.05560303 -0.06106567
-0.06417847]

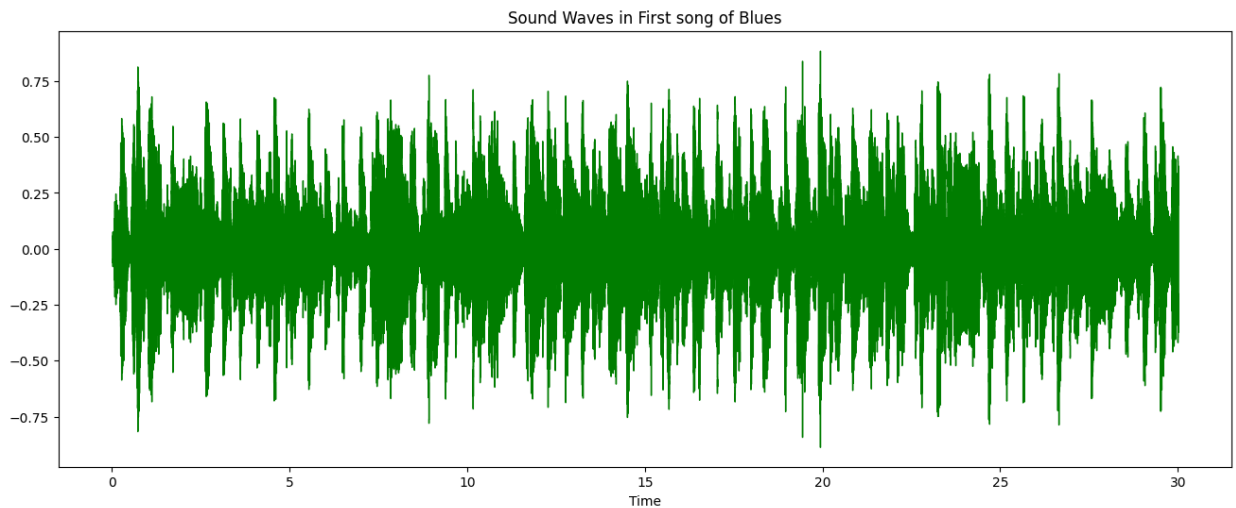
Audio File shape: (661794,)

Visual representation of Audio File

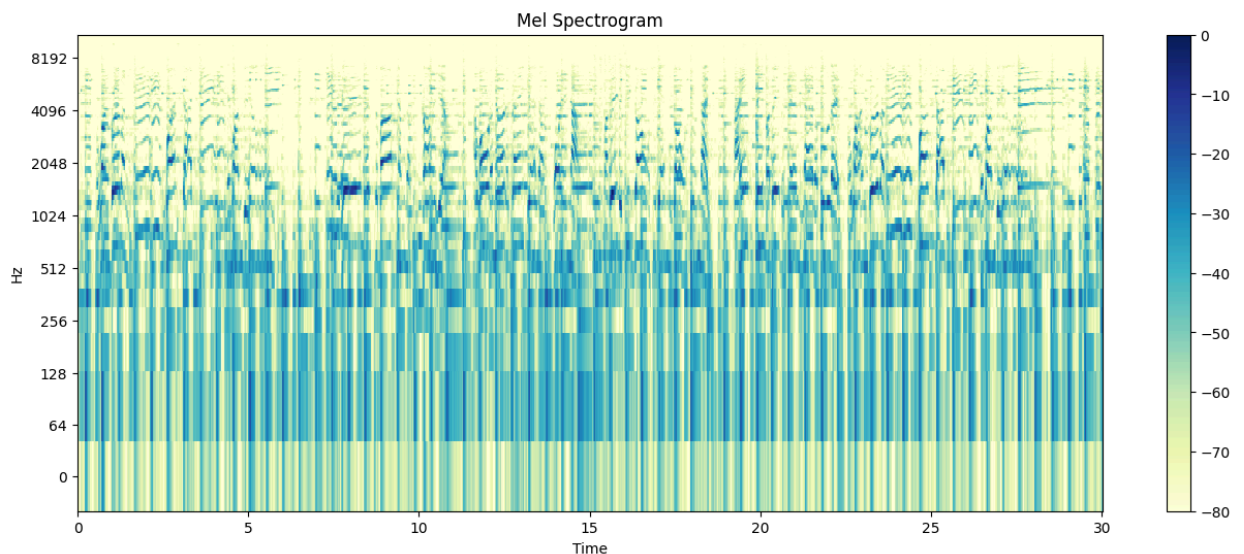
```
In [ ]: filelocation='/content/drive/MyDrive/genres_original'
hop_length=512
```

```
genre_names = ["blues", "classical", "country", "disco", "hiphop", "jazz", "me
```

```
In [ ]: plt.figure(figsize = (16, 6))
librosa.display.waveshow(y = audio_file, sr = sr, color = "green");
plt.title("Sound Waves in First song of Blues");
```



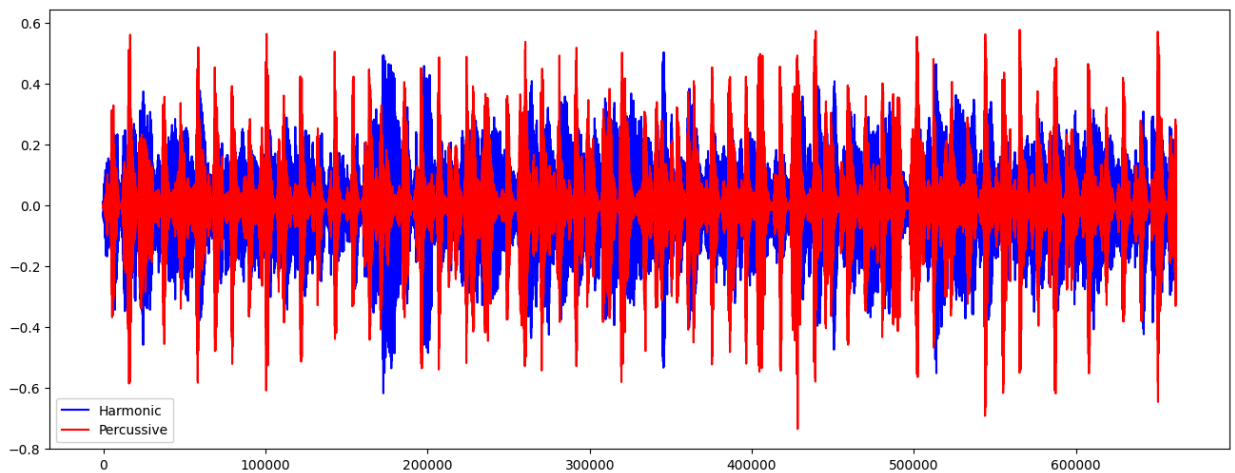
```
In [ ]: #Mel spectrogram
S = librosa.feature.melspectrogram(y=y, sr=sr)
S_DB = librosa.amplitude_to_db(S, ref=np.max)
plt.figure(figsize=(16, 6))
librosa.display.specshow(S_DB, sr=sr, hop_length=512, x_axis='time', y_axis='l
plt.colorbar()
plt.title("Mel Spectrogram",)
plt.show()
```



Harmonic and percussive components

```
In [ ]: harm,perc = librosa.effects.hpss(audio_file)
plt.figure(figsize = (16, 6))
plt.plot(harm, color='blue',label='Harmonic');
plt.plot(perc, color='red',label='Percussive')
```

```
plt.legend()
plt.show()
```



```
In [ ]: #Tempo analysis
tempo, _ = librosa.beat.beat_track(y=y, sr=sr)
print('Beats per Minute is ',tempo)
```

Beats per Minute is 123.046875

Feature Engineering

```
In [ ]: #Extracting MFCCs
mfcc = librosa.feature.mfcc(y=y, sr=sr, hop_length=512, n_mfcc=13)
mfcc.shape
mfcc.T[0:1200,:].shape
```

Out[]: (1200, 13)

```
In [ ]: mfcc
```

```
Out[ ]: array([[ -2.40635422e+02, -2.11214355e+02, -1.93908890e+02, ...,
        -1.09999146e+02, -8.68144302e+01, -8.40735855e+01],
       [ 9.96476364e+01,  1.01042831e+02,  1.02243965e+02, ...,
        1.50079346e+02,  1.38948669e+02,  1.38309769e+02],
       [-7.40327501e+00, -8.35852528e+00,  1.91543472e+00, ...,
        -5.07951355e+01, -3.65361443e+01, -2.81363564e+01],
       ...,
       [-2.22809958e+00, -4.09619999e+00, -9.18501282e+00, ...,
        -1.21473036e+01, -9.28038597e+00, -1.04724808e+01],
       [-3.98046923e+00,  1.07179761e+00, -2.12721896e+00, ...,
        6.25275517e+00,  2.70401812e+00,  4.79288101e-02],
       [-9.62531447e-01, -1.38649821e+00, -3.84490538e+00, ...,
        4.95667553e+00, -2.70487618e+00, -6.35826874e+00]], dtype=float32)
```

```
In [ ]: spectral_center = librosa.feature.spectral_centroid(y=y, sr=sr, hop_length=512)
spectral_center.shape
```

Out[]: (1, 1293)

```
In [ ]: chroma = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=512)
chroma.shape
```

Out[]: (12, 1293)

```
In [ ]: spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=512)
spectral_contrast.shape
```

Out[]: (7, 1293)

Batch processing of audio files for feature extraction

```
In [ ]: data=np.zeros((50, 128, 33), dtype=np.float64)
data.shape
```

Out[]: (50, 128, 33)

```
In [ ]: x=librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
x.shape
```

Out[]: (128, 1293)

```
In [ ]: q=librosa.feature.melspectrogram(y=y, sr=sr)
q.shape
```

Out[]: (128, 1293)

```
In [ ]: genres_dir = "/content/drive/MyDrive/genres_original"
data = np.zeros((999,512,33), dtype=np.float64)
target=[]
# List of genre names
genre_names = ["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "rock"]
i=0
for genre in genre_names:
    genre_path = os.path.join(genres_dir, genre)

    # Loop through each file in the genre folder
    for filename in os.listdir(genre_path):
        file_path = os.path.join(genre_path, filename)
        y, sr = librosa.load(file_path)
        y, _ = librosa.effects.trim(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr, hop_length=512, n_mfcc=13)
        spectral_center = librosa.feature.spectral_centroid(y=y, sr=sr, hop_length=512)
        chroma = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=512)
        spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=512)
        target.append(genre)
        data[i, :, 0:13] = mfcc.T[0:512,:]
        data[i, :, 13:14] = spectral_center.T[0:512, :]
        data[i, :, 14:26] = chroma.T[0:512, :]
        data[i, :, 26:33] = spectral_contrast.T[0:512, :]
        print("Numerical features extracted from audio file %i of %i." % (i + 1, len(genre_names)))
        i+=1
```

localhost:8889/nbconvert/html/Downloads/DATA_606_Project (3).ipynb?download=false

localhost:8889/nbconvert/html/Downloads/DATA_606_Project(3).ipynb?download=false

[illegible]

localhost:8889/nbconvert/html/Downloads/DATA_606_Project (3).ipynb?download=false

localhost:8889/nbconvert/html/Downloads/DATA_606_Project(3).ipynb?download=false

localhost:8889/nbconvert/html/Downloads/DATA_606_Project (3).ipynb?download=false

localhost:8889/nbconvert/html/Downloads/DATA_606_Project (3).ipynb?download=false

[illegible]

[illegible]

[illegible]

localhost:8889/nbconvert/html/Downloads/DATA_606_Project (3).ipynb?download=false

[illegible]

[illegible]

[illegible]

localhost:8889/nbconvert/html/Downloads/DATA_606_Project(3).ipynb?download=false

localhost:8889/nbconvert/html/Downloads/DATA_606_Project(3).ipynb?download=false

```

Numerical features extracted from audio file 961 of 999.
Numerical features extracted from audio file 962 of 999.
Numerical features extracted from audio file 963 of 999.
Numerical features extracted from audio file 964 of 999.
Numerical features extracted from audio file 965 of 999.
Numerical features extracted from audio file 966 of 999.
Numerical features extracted from audio file 967 of 999.
Numerical features extracted from audio file 968 of 999.
Numerical features extracted from audio file 969 of 999.
Numerical features extracted from audio file 970 of 999.
Numerical features extracted from audio file 971 of 999.
Numerical features extracted from audio file 972 of 999.
Numerical features extracted from audio file 973 of 999.
Numerical features extracted from audio file 974 of 999.
Numerical features extracted from audio file 975 of 999.
Numerical features extracted from audio file 976 of 999.
Numerical features extracted from audio file 977 of 999.
Numerical features extracted from audio file 978 of 999.
Numerical features extracted from audio file 979 of 999.
Numerical features extracted from audio file 980 of 999.
Numerical features extracted from audio file 981 of 999.
Numerical features extracted from audio file 982 of 999.
Numerical features extracted from audio file 983 of 999.
Numerical features extracted from audio file 984 of 999.
Numerical features extracted from audio file 985 of 999.
Numerical features extracted from audio file 986 of 999.
Numerical features extracted from audio file 987 of 999.
Numerical features extracted from audio file 988 of 999.
Numerical features extracted from audio file 989 of 999.
Numerical features extracted from audio file 990 of 999.
Numerical features extracted from audio file 991 of 999.
Numerical features extracted from audio file 992 of 999.
Numerical features extracted from audio file 993 of 999.
Numerical features extracted from audio file 994 of 999.
Numerical features extracted from audio file 995 of 999.
Numerical features extracted from audio file 996 of 999.
Numerical features extracted from audio file 997 of 999.
Numerical features extracted from audio file 998 of 999.
Numerical features extracted from audio file 999 of 999.

```

```

In [ ]: y=np.zeros((999,10))
        for i,genre in enumerate(target):
            ind=genre_names.index(genre)
            y[i,ind]=1

```

```

In [ ]: y

```

```

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

```

```

In [ ]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import LSTM, Dense
        from tensorflow.keras.optimizers import Adam
        from tensorflow.keras.models import Sequential

```

```
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

```
In [ ]: from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Flatten
from tensorflow.keras.optimizers import Adam

# Define the number of folds
n_splits = 5

# Initialize StratifiedKFold
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

# Initialize lists to store evaluation metrics for each fold
acc_per_fold = []
loss_per_fold = []

# Iterate over each fold
# Convert multilabel-indicator to single-label format
y_single_label = np.argmax(y, axis=1)

for fold_index, (train_index, val_index) in enumerate(skf.split(data, y_single_label)):
    print(f"Training on Fold {fold_index + 1}")

    # Split data into train and validation sets for this fold
    x_train_fold, x_val_fold = data[train_index], data[val_index]
    y_train_fold, y_val_fold = y[train_index], y[val_index]

    # Define the model
    model = Sequential()
    model.add(Conv1D(filters=256, kernel_size=3, activation='relu', input_shape=(data.shape[1], data.shape[2])))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(LSTM(units=64, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
    model.add(LSTM(units=32, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dropout(0.5))
    model.add(Dense(units=10, activation='softmax'))

    # Compile the model
    model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

    # Train the model
    history = model.fit(x_train_fold, y_train_fold, batch_size=35, epochs=400,
                        validation_data=(x_val_fold, y_val_fold))

    # Evaluate the model on the validation set
    scores = model.evaluate(x_val_fold, y_val_fold, verbose=0)

    # Append evaluation metrics to lists
    acc_per_fold.append(scores[1] * 100) # Accuracy
    loss_per_fold.append(scores[0]) # Loss

    print(f"Validation Accuracy: {scores[1] * 100:.2f}%")
    print(f"Validation Loss: {scores[0]:.4f}")

# Print average metrics across all folds
print(f"\nAverage Validation Accuracy: {sum(acc_per_fold) / len(acc_per_fold):.2f}%")
print(f"Average Validation Loss: {sum(loss_per_fold) / len(loss_per_fold):.4f}")
```

Training on Fold 1

WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.

WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.

```
In [ ]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Flatten
from tensorflow.keras.optimizers import Adam

# Define the model
model = Sequential()

# Add Convolutional layers
model.add(Conv1D(filters=256, kernel_size=3, activation='relu', input_shape=(5,)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))

# Flatten the output for LSTM
#model.add(Flatten())

# Reshape for LSTM input
#model.add(Reshape((32, 120))) # Reshape to (timesteps, features)

# Add LSTM layers
model.add(LSTM(units=64, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
model.add(LSTM(units=32, dropout=0.2, recurrent_dropout=0.2))

# Add Dropout for regularization
model.add(Dropout(0.5))

# Output layer
model.add(Dense(units=10, activation='softmax'))

# Compile the model with a lower learning rate
opt = Adam()
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

# Print model summary
model.summary()
```

```
In [ ]: from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, y, test_size=0.25, random_state=42)
```

```
In [ ]: genre_names = ["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "rock", "soul", "swing"]
```

```
In [ ]: batch_size = 35 # num of training examples per minibatch
num_epochs = 400
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=num_epochs, validation_data=(x_test, y_test))

# Calculate the mean training accuracy
mean_training_accuracy = np.mean(history.history['accuracy'])
print("Mean Training Accuracy:", mean_training_accuracy)
```

```
In [ ]: import math
# score, accuracy = model.evaluate(
#     x_test, y_test, batch_size=batch_size, verbose=1
# )

num_batches = len(x_test) // batch_size
accuracies = []

for i in range(num_batches):
    start = i * batch_size
    end = (i + 1) * batch_size
    batch_x = x_test[start:end]
    batch_y = y_test[start:end]
    _, batch_accuracy = model.evaluate(batch_x, batch_y, verbose=0)
    accuracies.append(batch_accuracy)

mean_test_accuracy = np.mean(accuracies)
print("Mean Test Accuracy:", mean_test_accuracy)
```

```
In [ ]: model.save('m01.h5')
```

```
In [ ]:
```