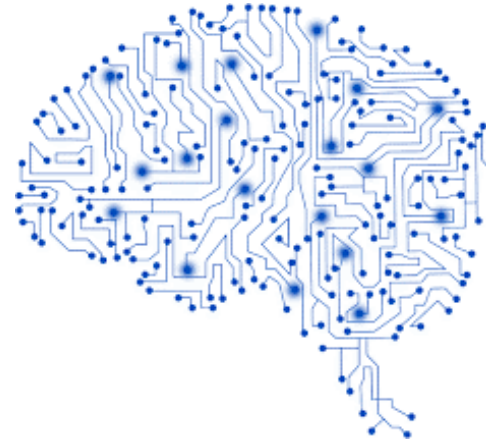




University of Minho
School of Engineering



Dados e Aprendizagem Automática

Support Vector Machines and Librosa

DAA @ MEI-1º/MiEI-4º – 1º Semestre

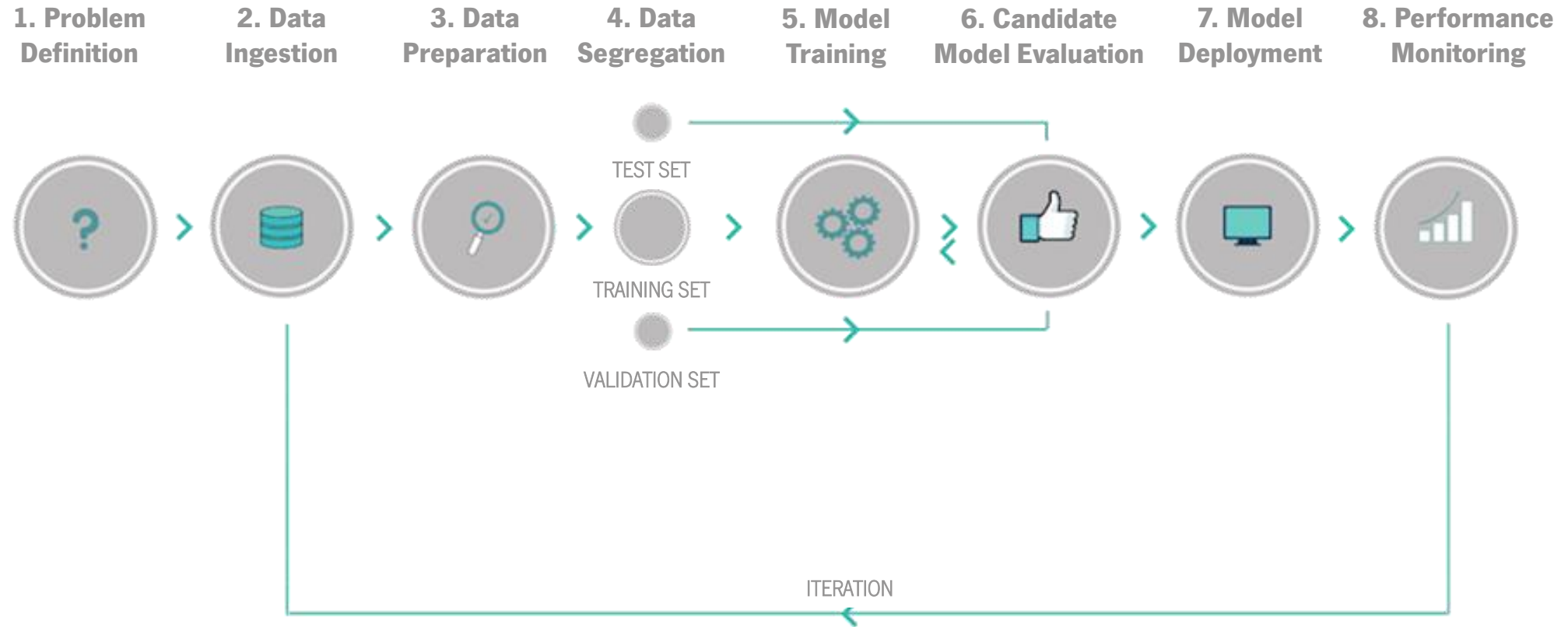
Bruno Fernandes, Dalila Alves, Filipa Ferraz, Victor Alves

Part X

Contents

2

- Librosa
- Support Vector Machines
 - Support Vector Classifier
- Hands On





Librosa

The Data

5

For this class, we will use a music dataset. The [GenAudio dataset](#) is composed by audio files and CSV files with extracted features from the audio files:

- **genres original** - collection of 10 genres with 100 audio files each, all having a length of 30 seconds;
- **CSV files** – extracted features of the audio files. One file has for each song (30 seconds long) a mean and variance computed over multiple features that can be extracted from an audio file. The other file has the same structure, but the songs were split before into 3 seconds audio files (this way increasing 10 times the amount of data we feed into our classification models).

This dataset is frequently used for evaluation in machine listening research for Music Genre Recognition (MGR). The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording.

Librosa

6

In order to work with audio data, we will use `librosa`, a python library used for audio and music analysis. It is a powerful package widely used for audio visualization and for building Music Information Retrieval (MIR) systems.



You may need to install [librosa](#):
`pip install librosa`

Explore the sound files

7

Let's start by selecting one file and explore it:

```
audio_path = 'Data/genres_original/reggae/reggae.00010.wav'
```

Load the audio as a waveform (data) and store the sampling rate (sr):

```
data, sr = librosa.load(audio_path)
print(type(data), type(sr))

print(data.shape, sr)

<class 'numpy.ndarray'> <class 'int'>
(661794,) 22050
```

Initialize the sample rate to 45600 so we can obtain the signal value array:

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

(array([-0.00555292, -0.00768963, -0.00668519, ...,  0.08035275,
        0.0663713 ,  0.03239053], dtype=float32),
45600)
```

Understanding waveform,
sampling rate and other sound
concepts: [here](#)

Explore the sound files

8

Transform the signal using Short-Time Fourier Transform (STFT):

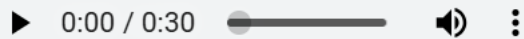
```
stft = librosa.stft(data)
```

Convert an amplitude spectrogram to dB-scaled spectrogram:

```
stft_db = librosa.amplitude_to_db(abs(stft))
```

And play the audio file:

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

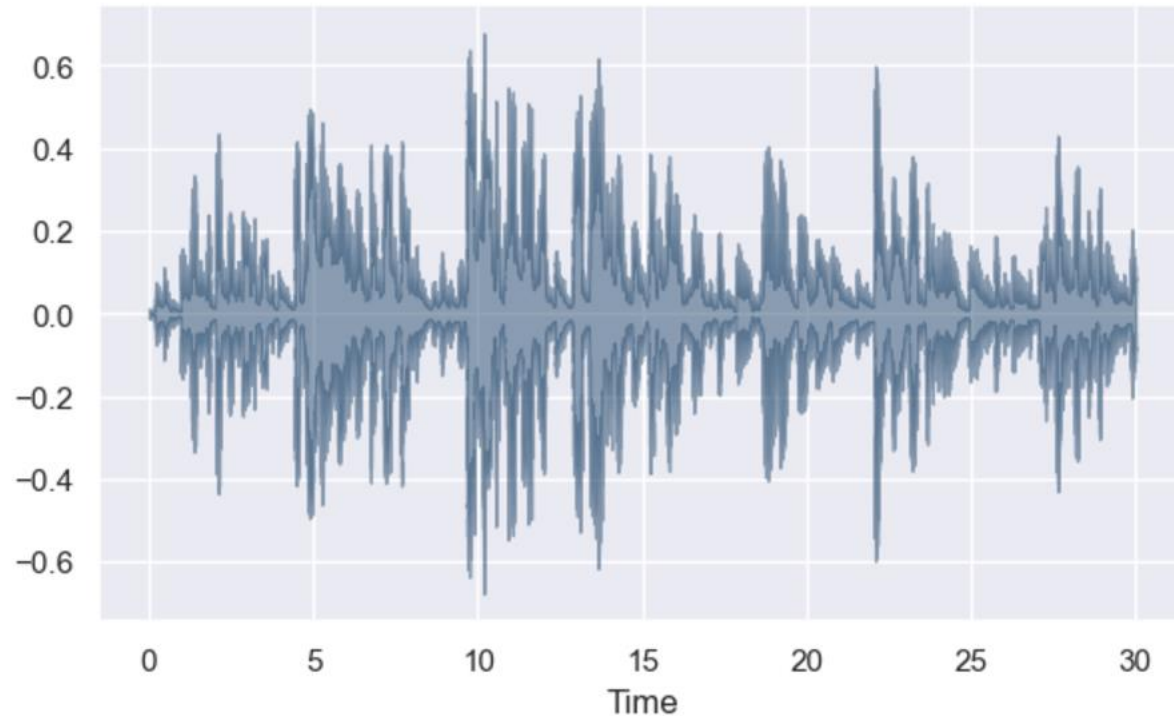


Explore the sound files

9

Let's see the waveform representation of this audio file:

```
plt.figure(figsize=(7, 4))  
librosa.display.waveshow(data, color="#2B4F72", alpha=0.5)  
plt.show()
```



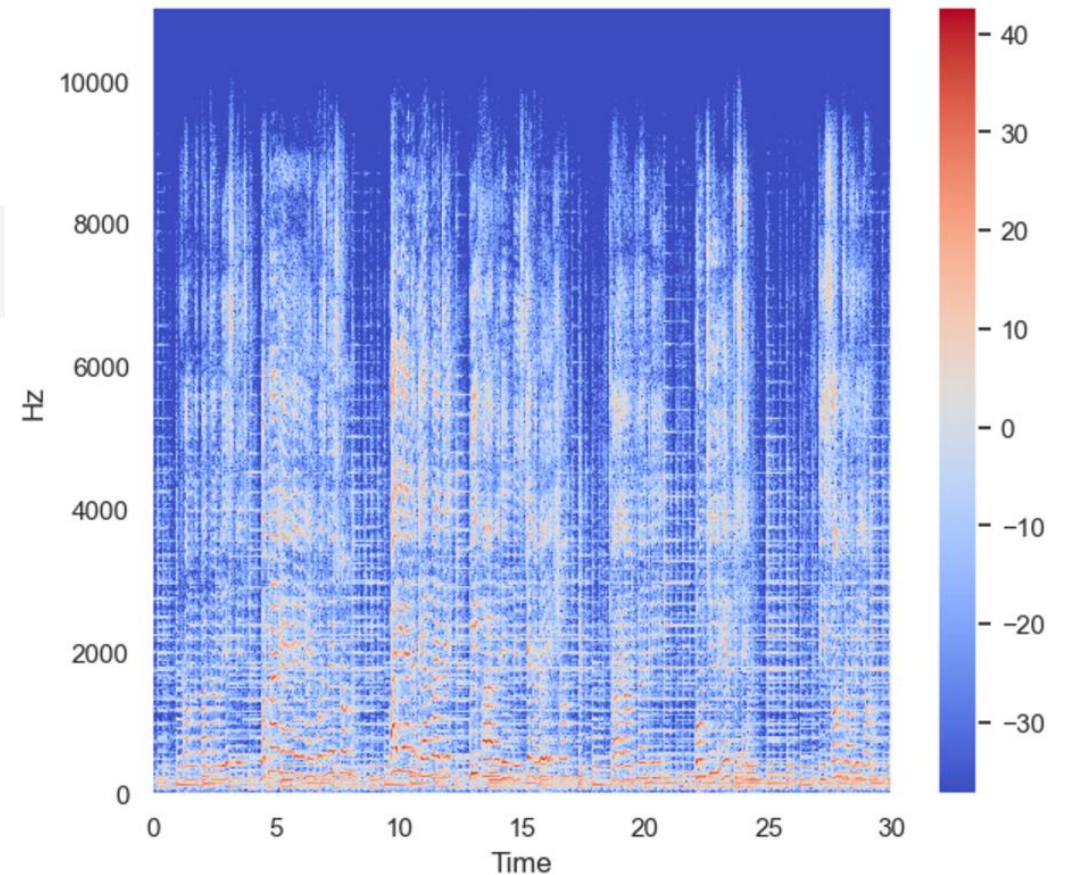
Explore the sound files

10

A *spectrogram* is a visual representation of the signal loudness over time at different frequencies included in a certain waveform. We can examine increase or decrease of energy over period of time. Spectrograms are also known as sonographs, voiceprints, and voicegrams.

Let's obtain the spectrogram of this audio:

```
plt.figure(figsize=(7, 6))
librosa.display.specshow(stft_db, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```



Data Pre-Processing

11

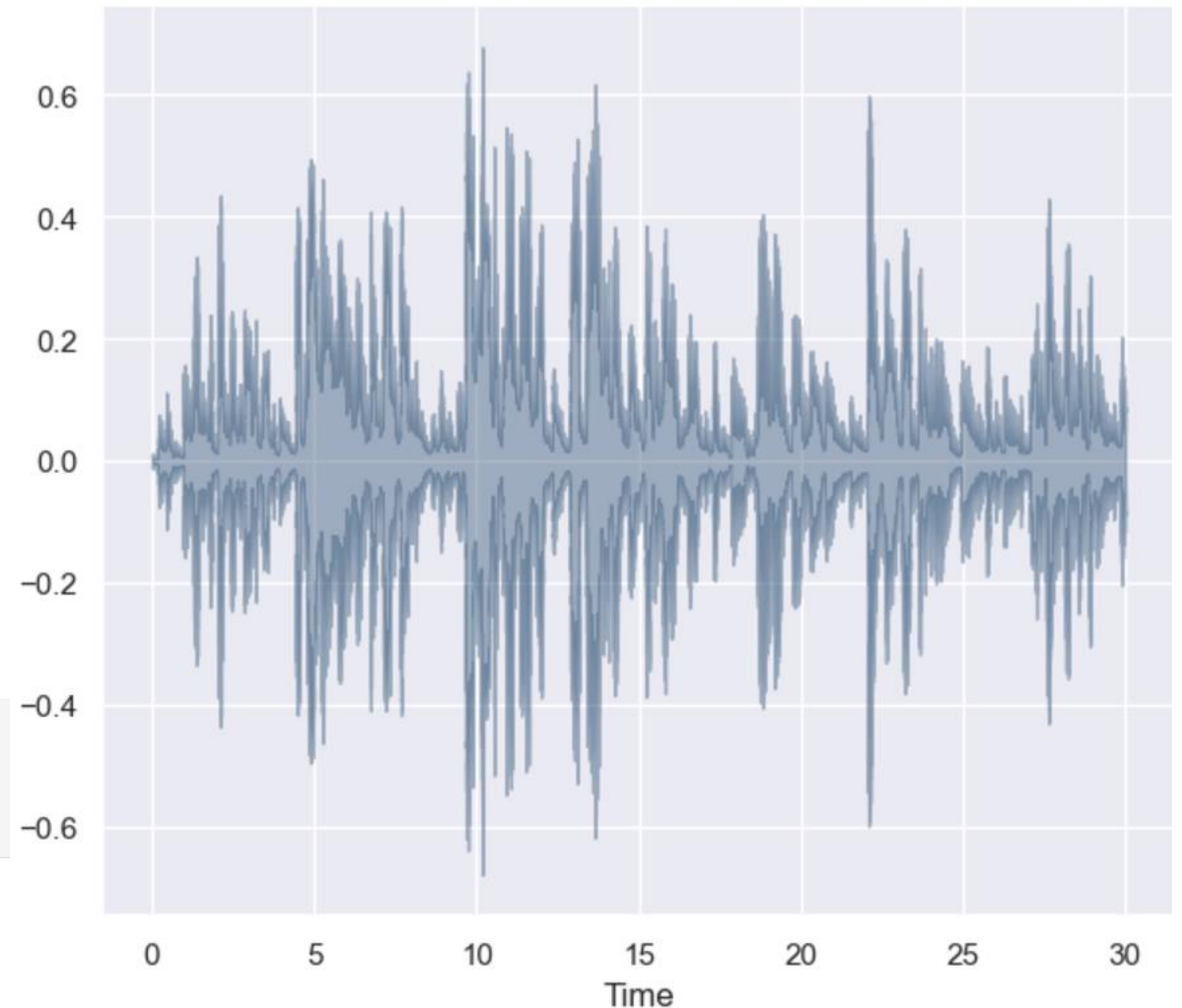
Extracting Audio Features

Each audio signal consists of various audio features however we must extract features that are relevant to the problem that we are solving. We will analyze some features:

- **Spectral Roll-Off**

It computes the roll-off frequency for each frame in each signal. The frequency under which some percentage (*cut-off*) of the total energy of a spectrum is obtained. It can be used to differentiate between the harmonic and noisy sounds.

```
spectral_rolloff = librosa.feature.spectral_rolloff(y=data, sr=sr)[0]  
  
plt.figure(figsize=(7, 6))  
librosa.display.waveshow(data, sr=sr, alpha=0.4, color="#2B4F72")
```



Data Pre-Processing

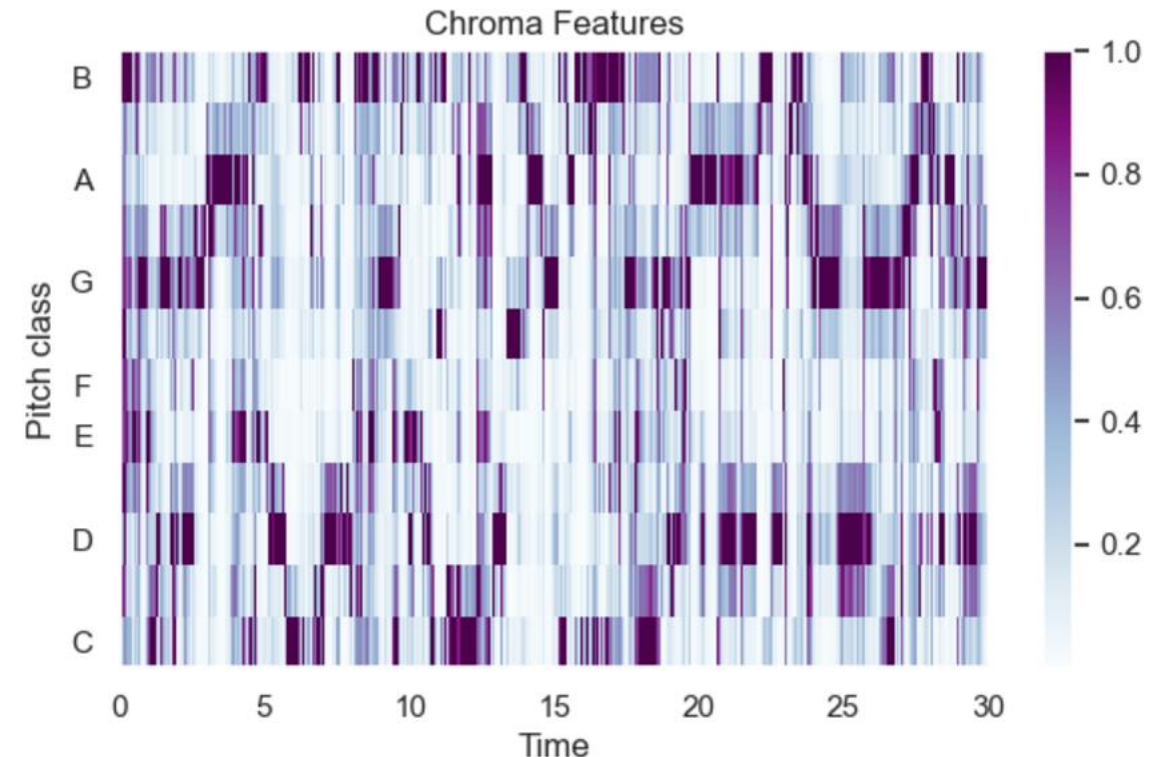
12

- **Chroma Feature**

It closely relates with the 12 different pitch classes. Chroma based features are also called as pitch class profiles. It is the powerful tool for analyzing and categorizing them. Harmonic and melodic characteristics of music are captured by them. It computes the chromogram from a waveform or power spectrogram.

```
chroma = librosa.feature.chroma_stft(y=data, sr=sr)
```

```
plt.figure(figsize=(7, 4))  
lplt.specshow(chroma, sr=sr, x_axis="time", y_axis="chroma", cmap="BuPu")  
plt.colorbar()  
plt.title("Chroma Features")  
plt.show()
```



Data Pre-Processing

13

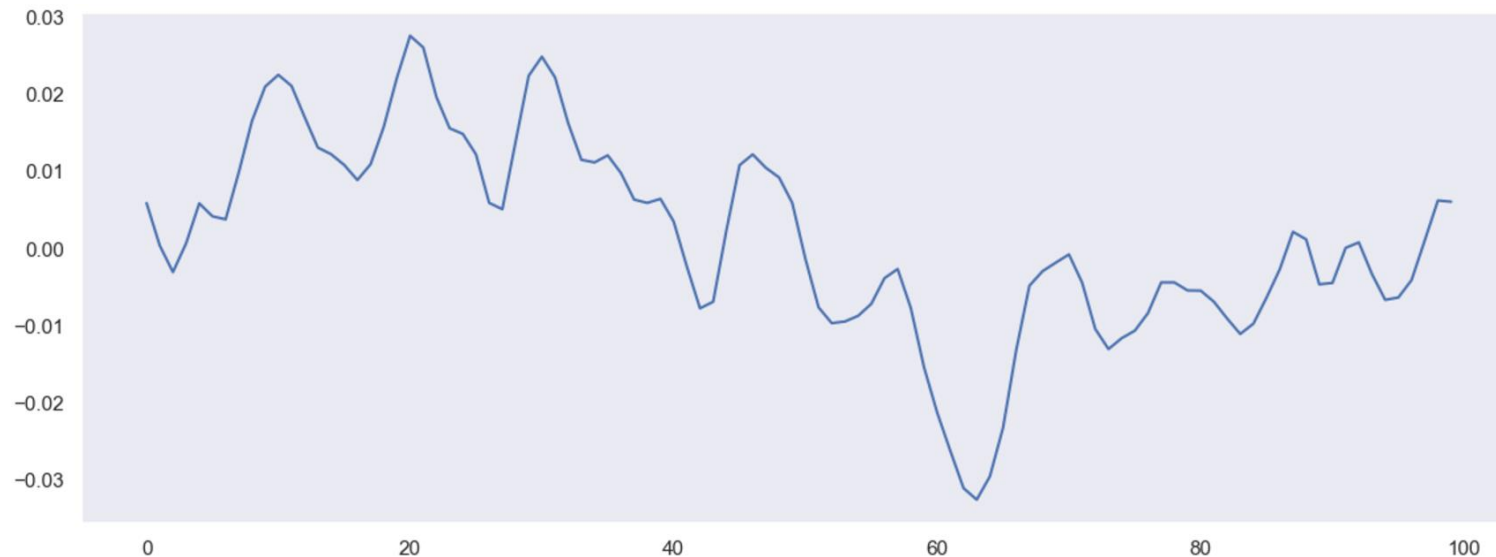
- **Zero-Crossing Rate**

The Zero-Crossing Rate (ZCR) is the rate of sign-changes along with a signal, i.e., the rate at which the signal changes from positive to negative or back - the number of times the signal crosses x-axis. This feature is heavily used in both speech recognition and music information retrieval. It usually has higher values for highly percussive sounds like those in metal and rock.

```
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(data[n0:n1], color="#2B4F72")
plt.grid()

zcr = librosa.zero_crossings(data[n0:n1], pad=False)
print(sum(zcr))
```

10



Data Pre-Processing

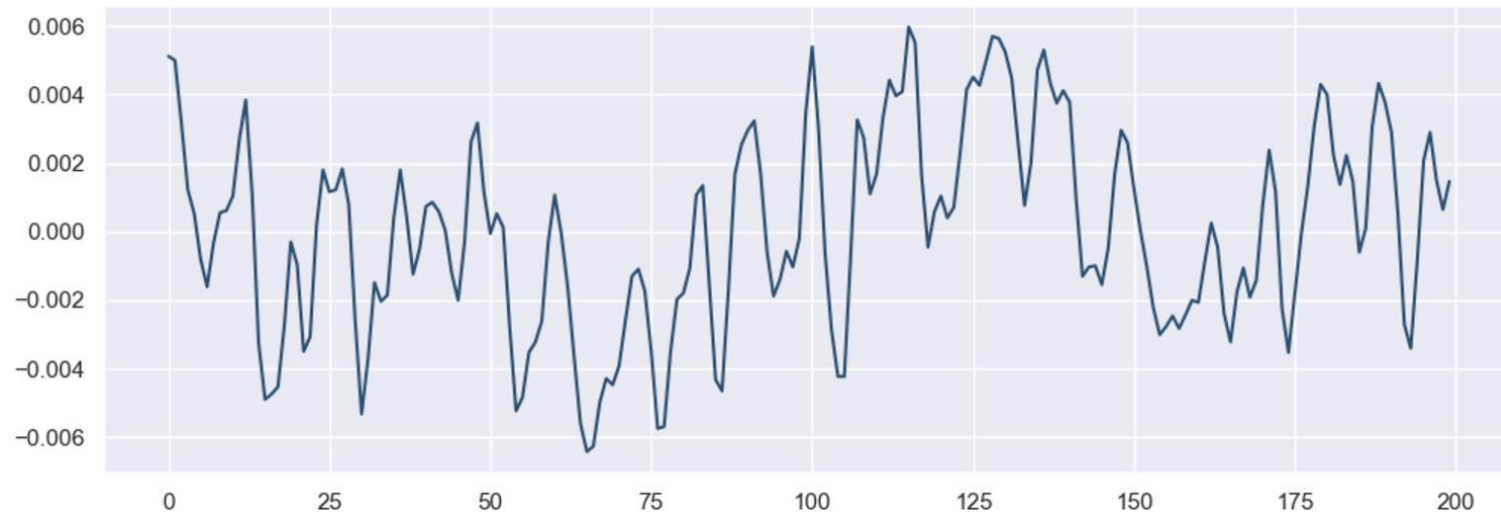
14

Changing the start and the end:

```
start = 1000
end = 1200
plt.figure(figsize=(12, 4))
plt.plot(data[start:end], color="#2B4F72")

zcr = librosa.zero_crossings(data[start:end], pad=False)
print(sum(zcr))
```

36



Data Pre-Processing

15

- **Spectral Centroid**

It indicates where the "center of mass" for a sound is located and is calculated as the weighted mean of the frequencies present in the sound. Consider two songs, one from a blues genre and the other belonging to metal. Now, as compared to the blues genre song, which is the same throughout its length, the metal song has more frequencies towards the end. So spectral centroid for blues song will lie somewhere near the middle of its spectrum while that for a metal song would be towards its end. To compute the spectral centroid, each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame.

```
spectral_centroids = librosa.feature.spectral_centroid(y=data, sr=sr)[0]  
spectral_centroids.shape
```

```
(1293,)
```

Convert frame counts to time (seconds):

```
frames = range(len(spectral_centroids))  
t = librosa.frames_to_time(frames)
```

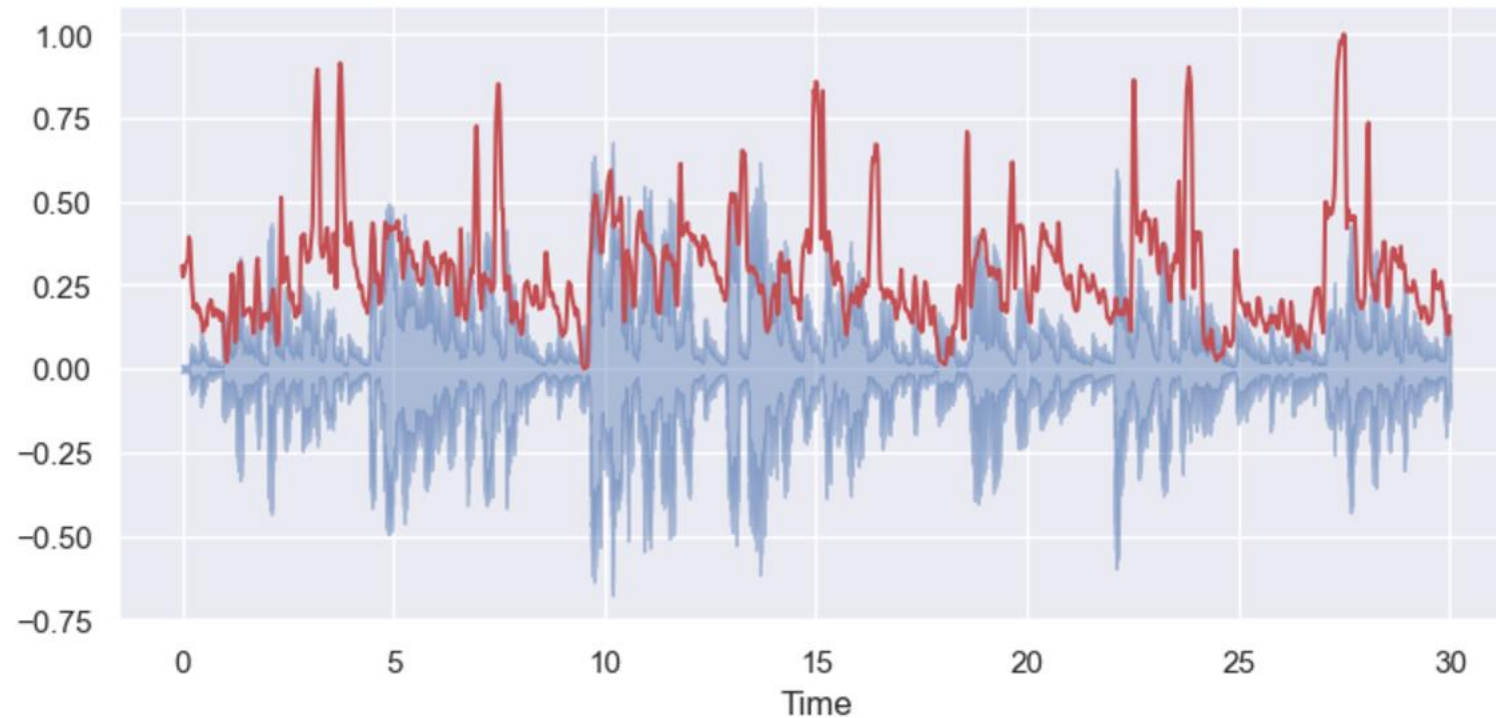

Data Pre-Processing

16

Transform features by scaling each feature to a given range:

```
def normalize(data, axis=0):  
    return sklearn.preprocessing.minmax_scale(data, axis=axis)
```

```
librosa.display.waveshow(data, sr=sr, alpha=0.4)  
plt.plot(t, normalize(spectral_centroids), color='r')
```



Exploratory Data Analysis

17

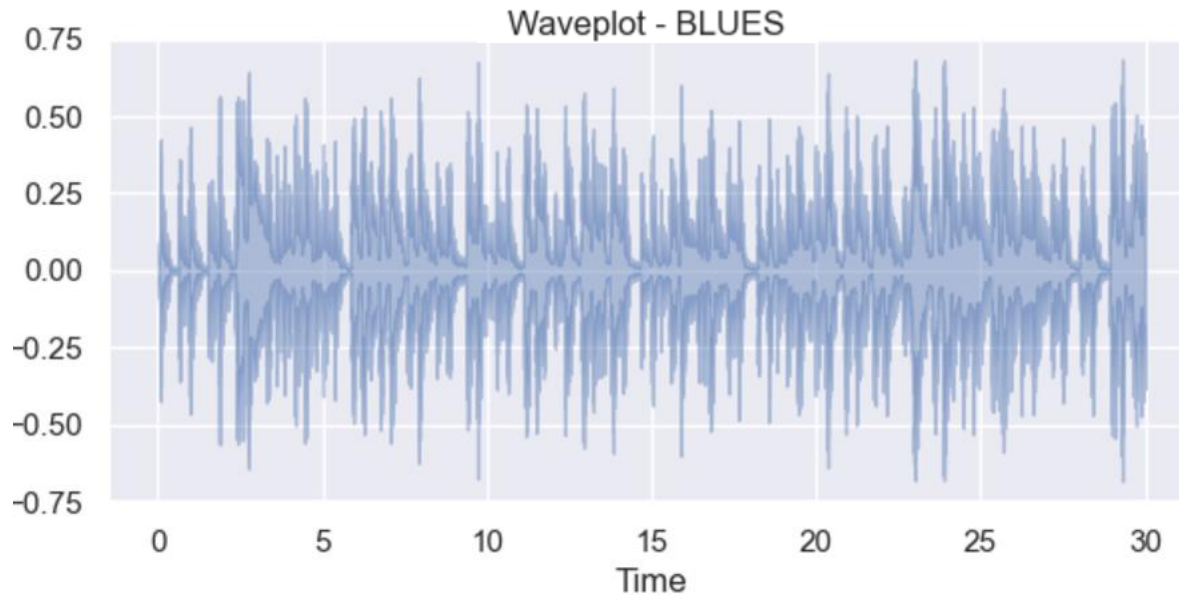
ipd.Audio(audio1)

▶ 0:00 / 0:30 ———— 🔊 ⋮

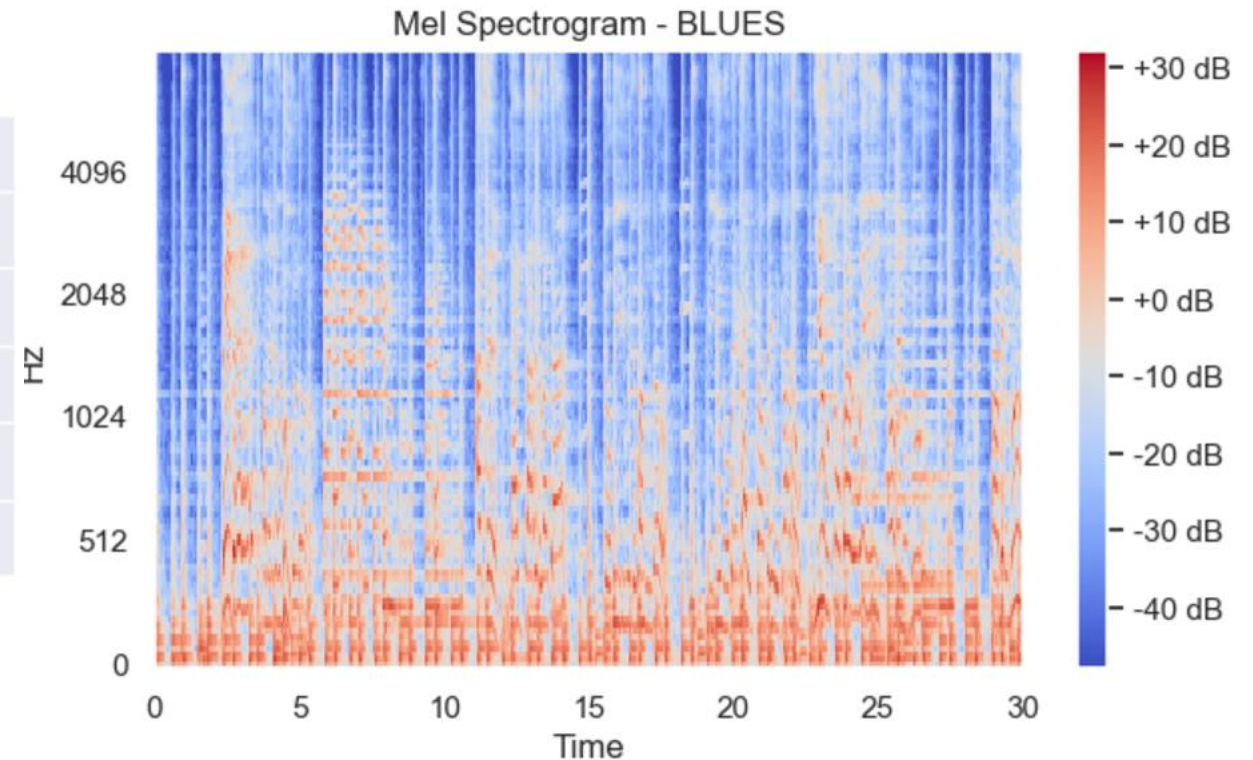
Let's visualize the audio files, wave plots and spectrograms for all the 10 genre classes:

1. Blues

```
audio1 = 'Data/genres_original/blues/blues.00001.wav'  
data, sr = librosa.load(audio1)  
plt.figure(figsize=(7, 3))  
librosa.display.waveshow(data, sr=sr, alpha=0.4,  
plt.title('Waveplot - BLUES')
```



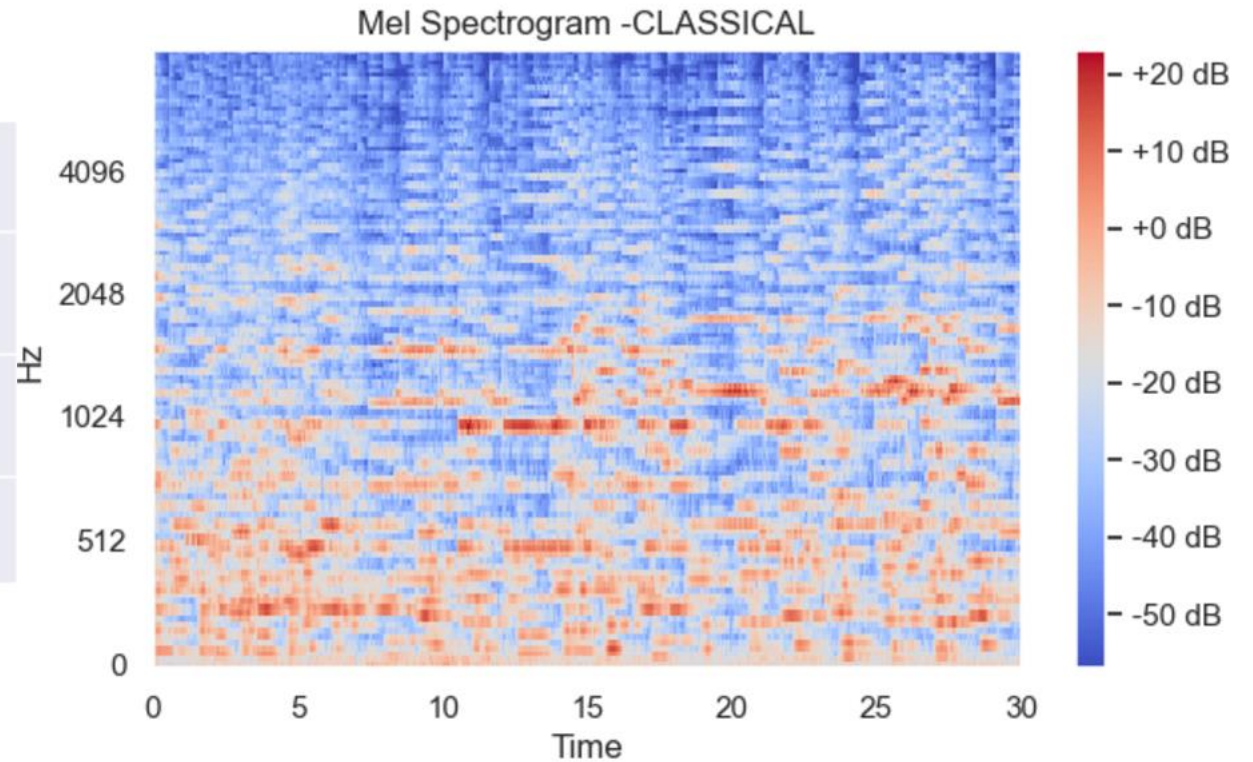
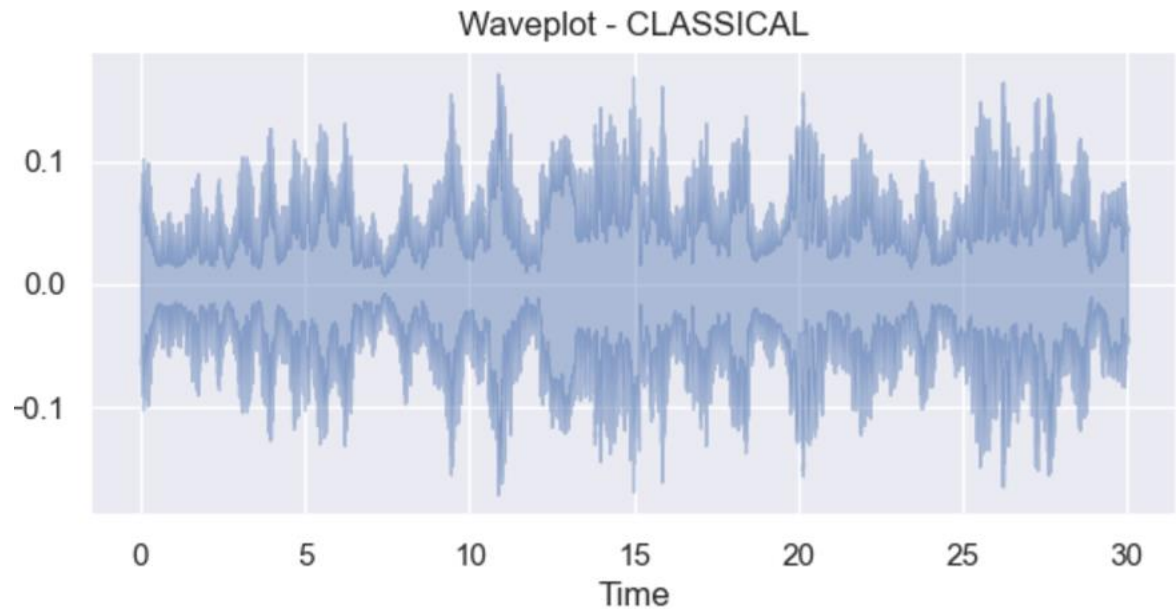
```
plt.figure(figsize=(7, 4))  
spectrogram = librosa.feature.melspectrogram(y=data, sr=sr, n_mels=128, fmax=8000)  
spectrogram = librosa.power_to_db(spectrogram)  
librosa.display.specshow(spectrogram, y_axis='mel', fmax=8000, x_axis='time')  
plt.title('Mel Spectrogram - BLUES')  
plt.colorbar(format='%+2.0f dB')
```



Exploratory Data Analysis

18

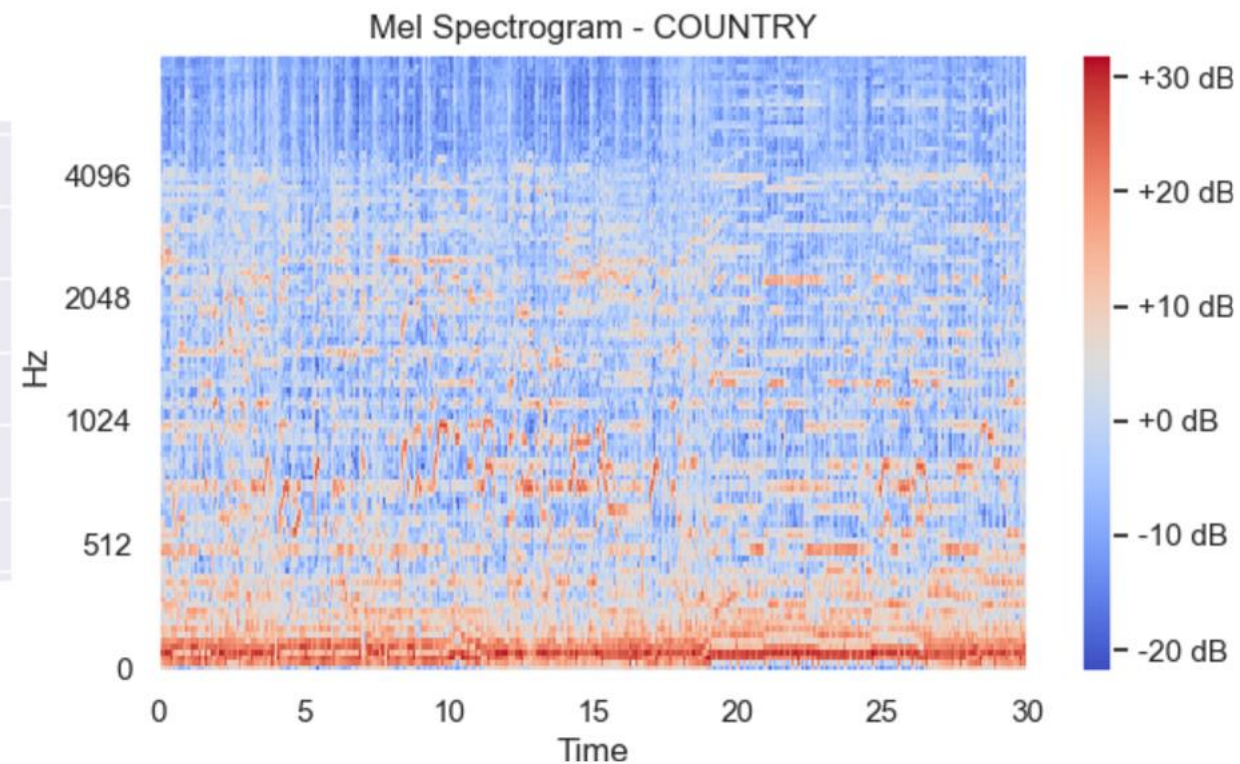
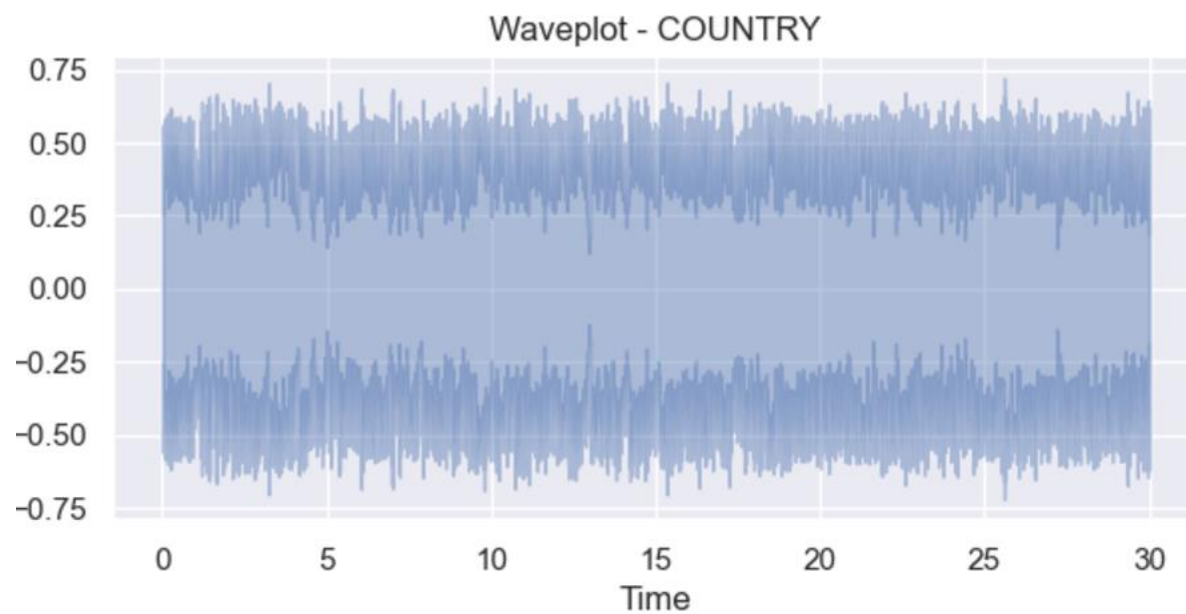
2. Classical



Exploratory Data Analysis

19

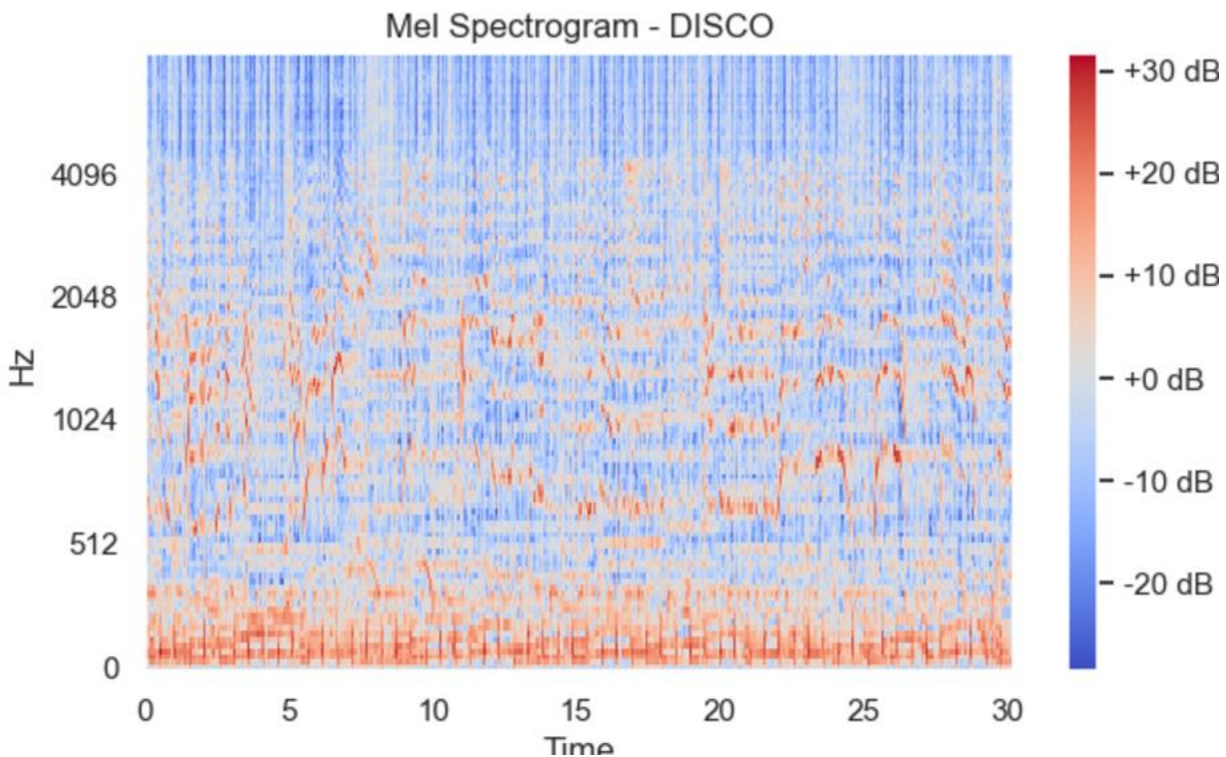
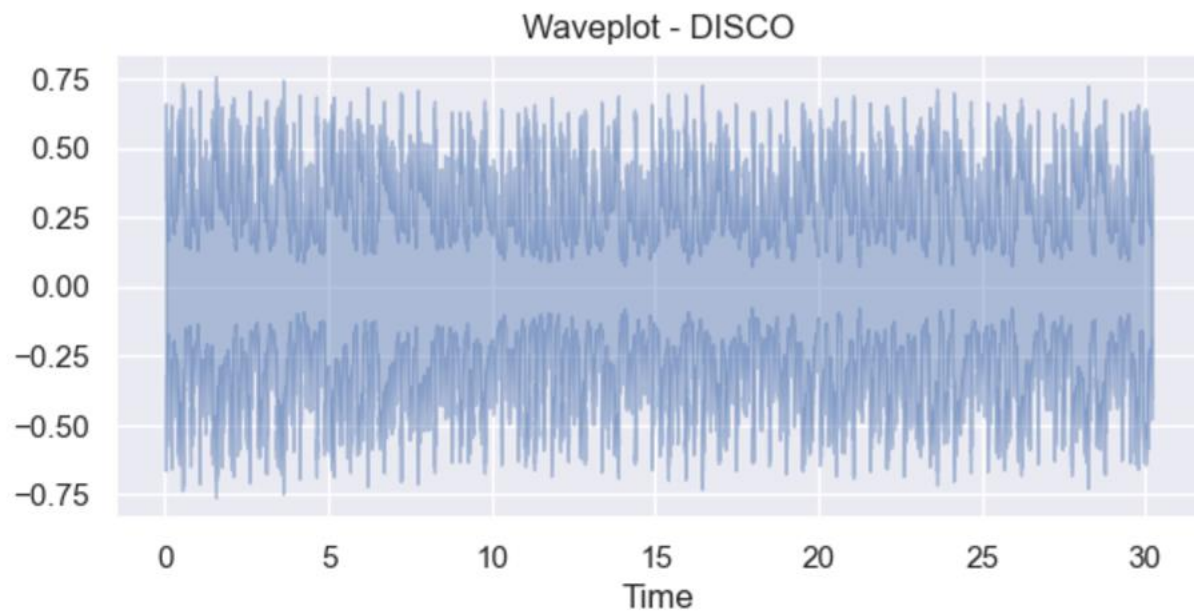
3. Country



Exploratory Data Analysis

20

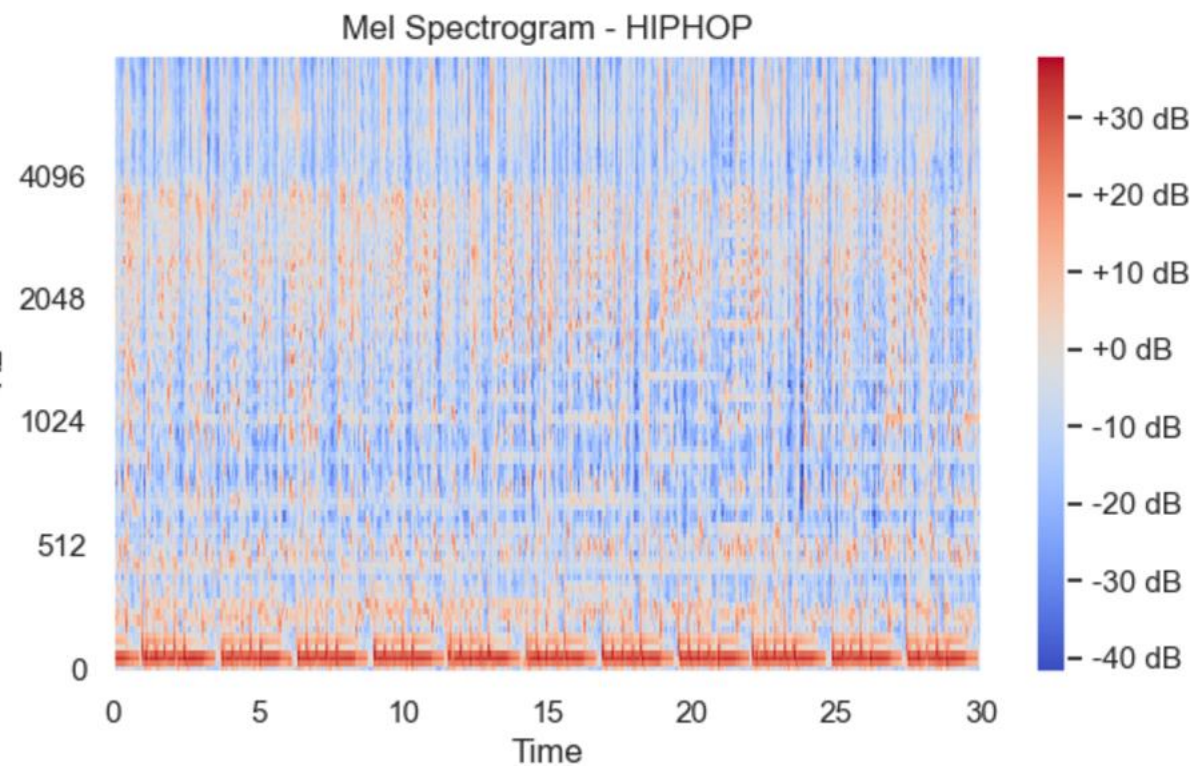
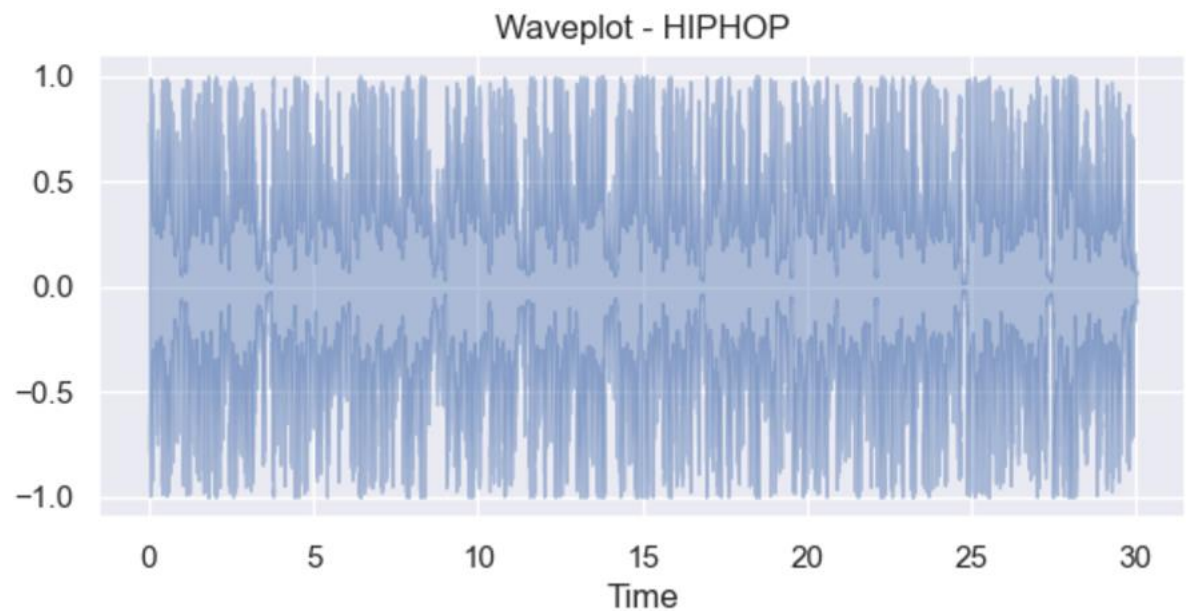
4. Disco



Exploratory Data Analysis

21

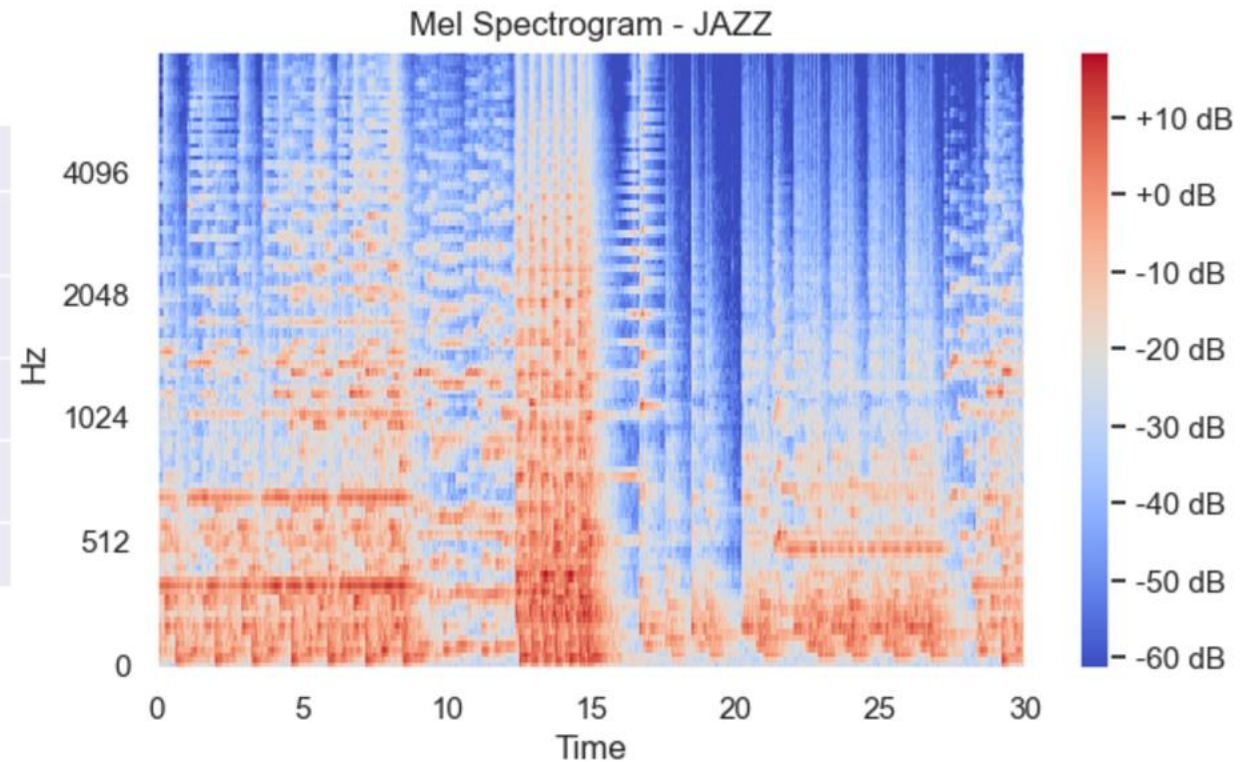
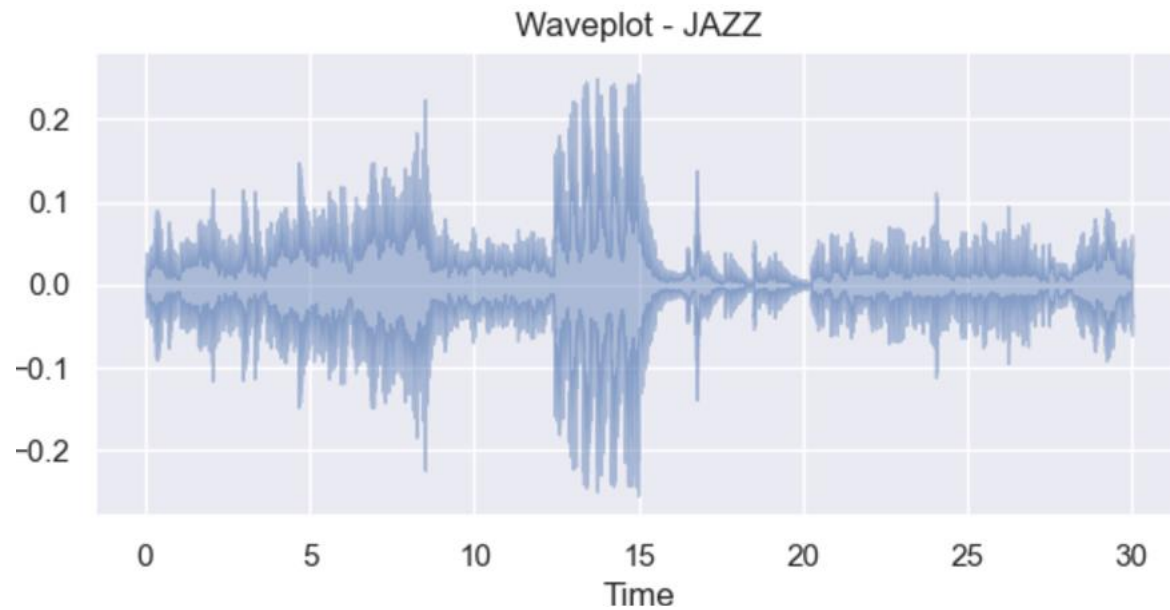
5. Hip-Hop



Exploratory Data Analysis

22

6. Jazz

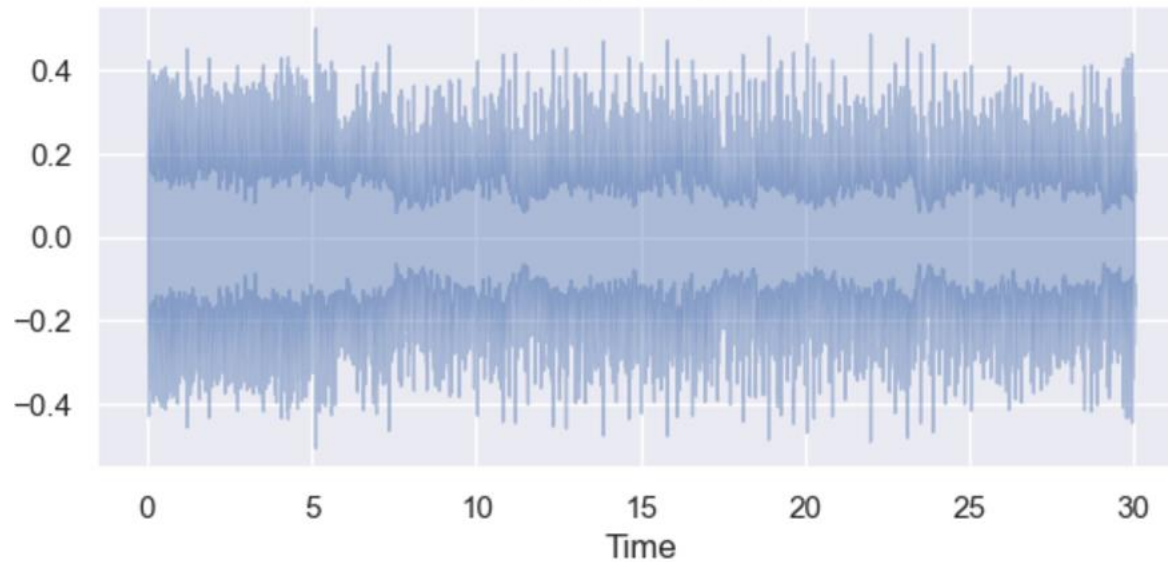


Exploratory Data Analysis

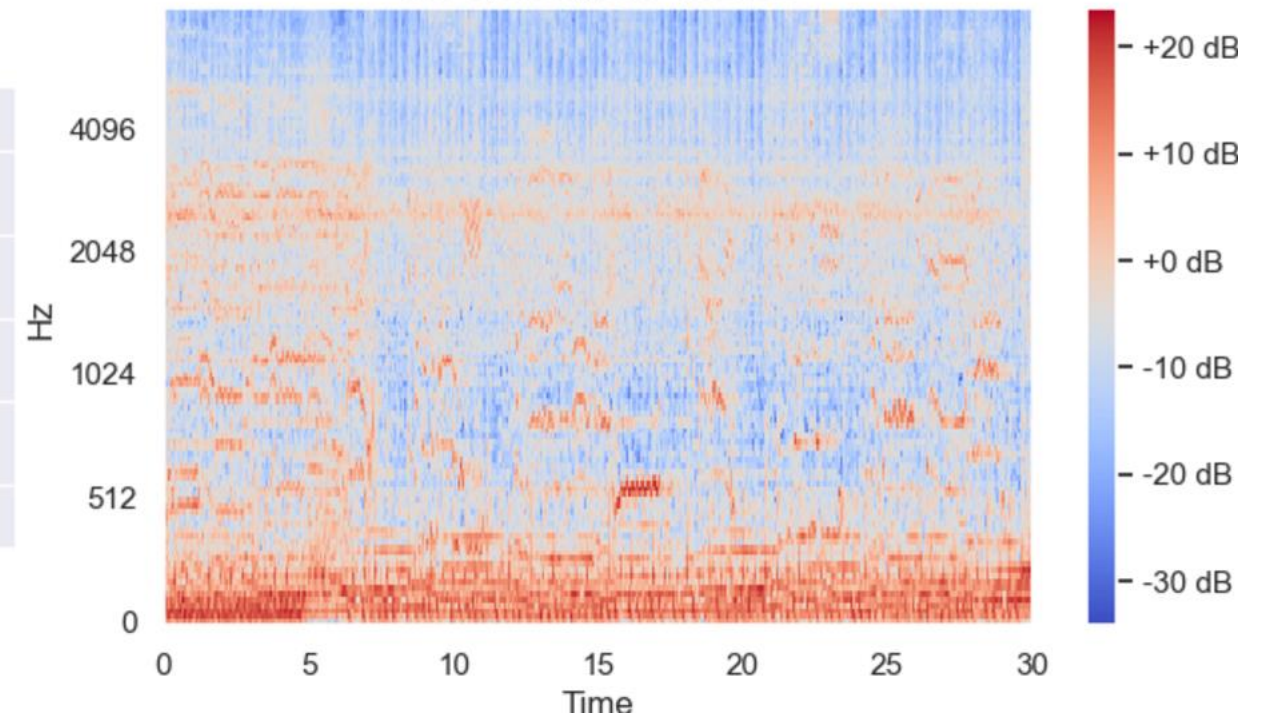
23

7. Metal

Waveplot - METAL



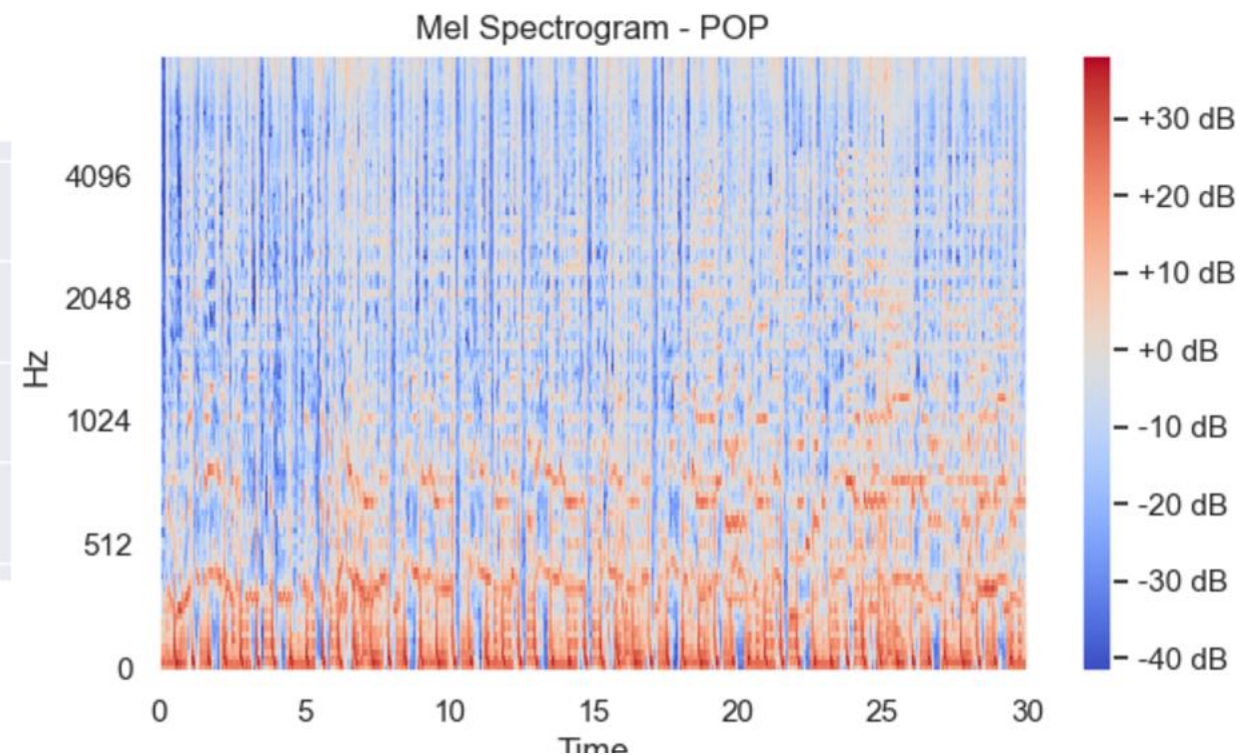
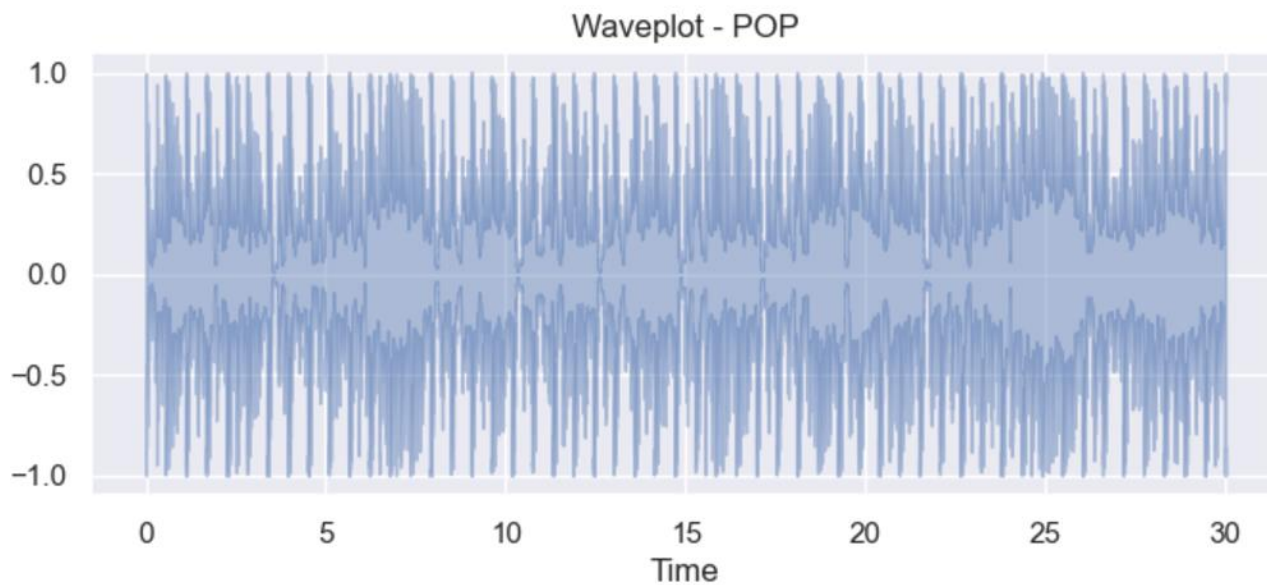
Mel Spectrogram - METAL



Exploratory Data Analysis

24

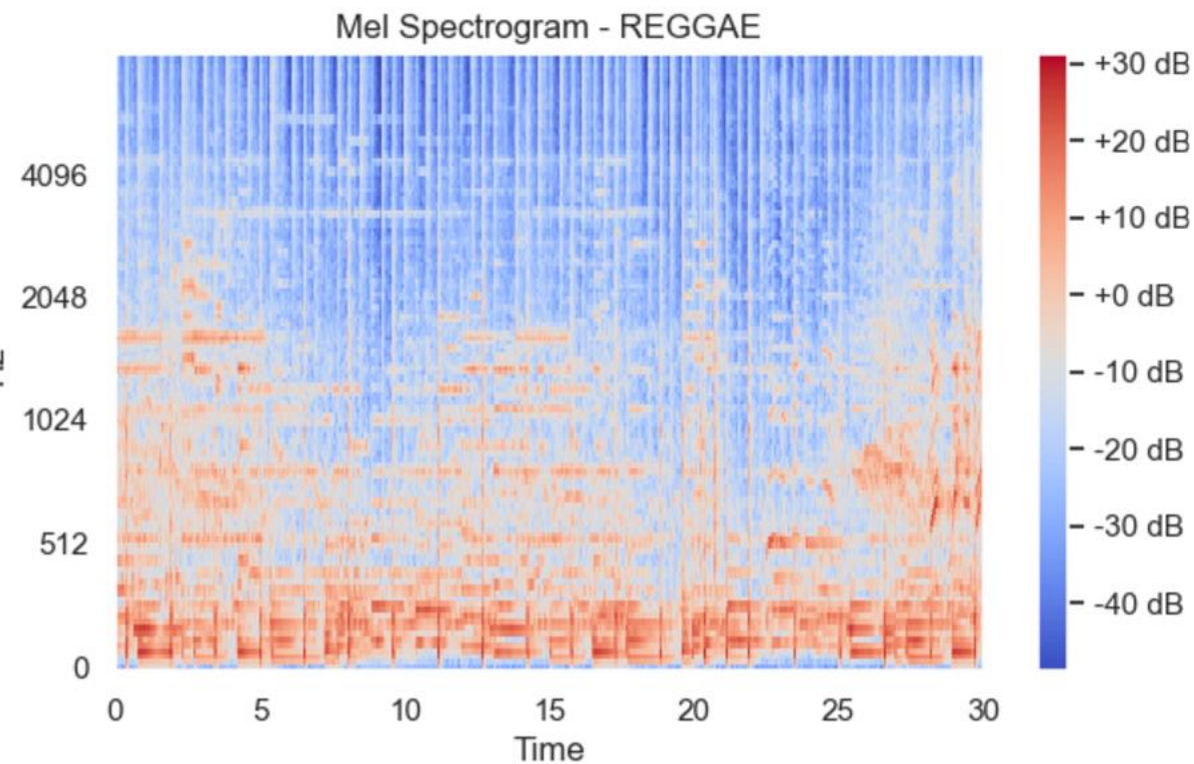
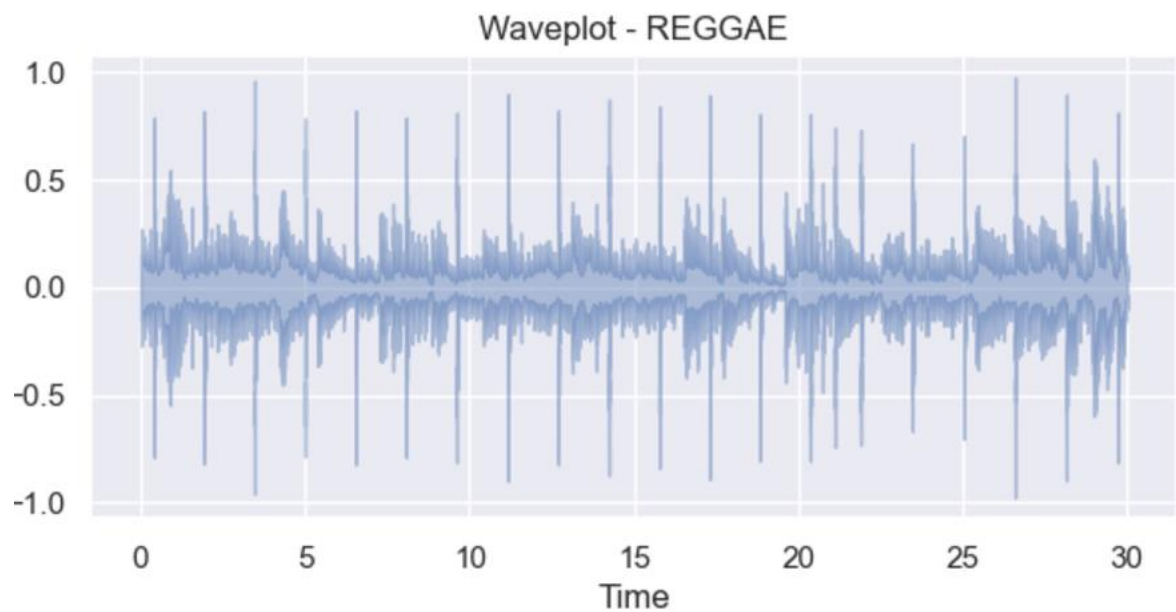
8. Pop



Exploratory Data Analysis

25

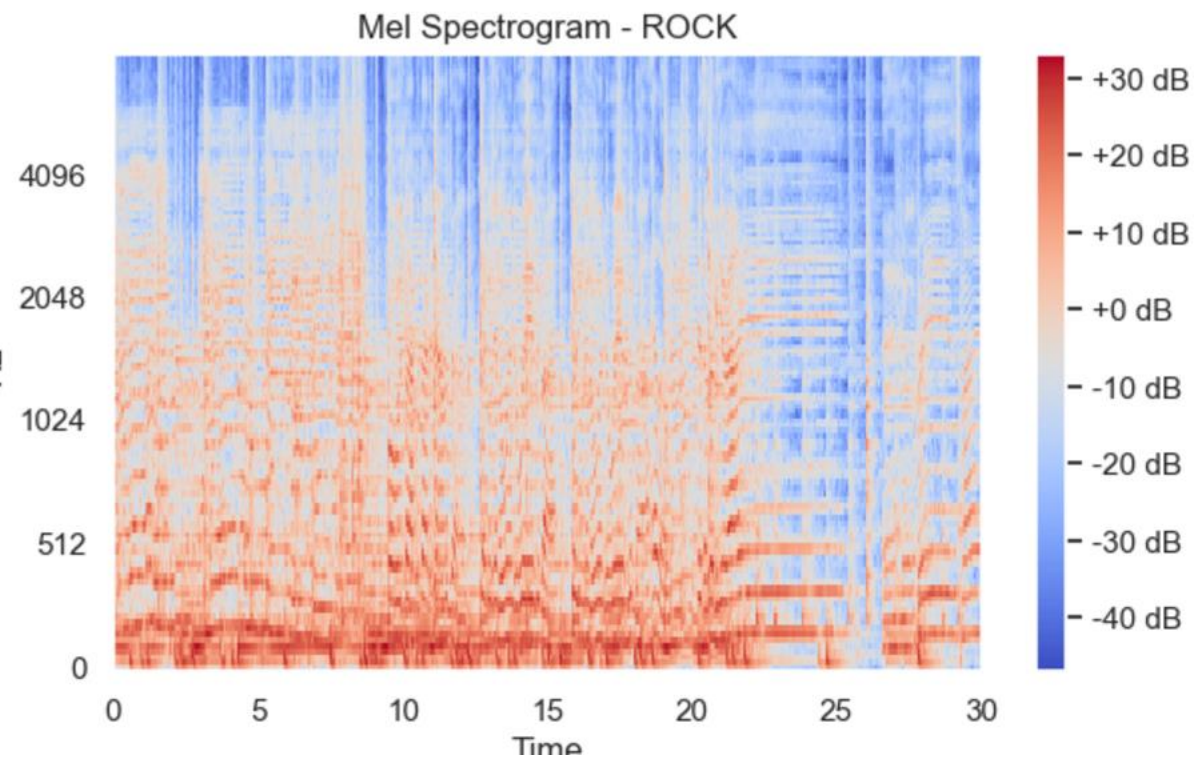
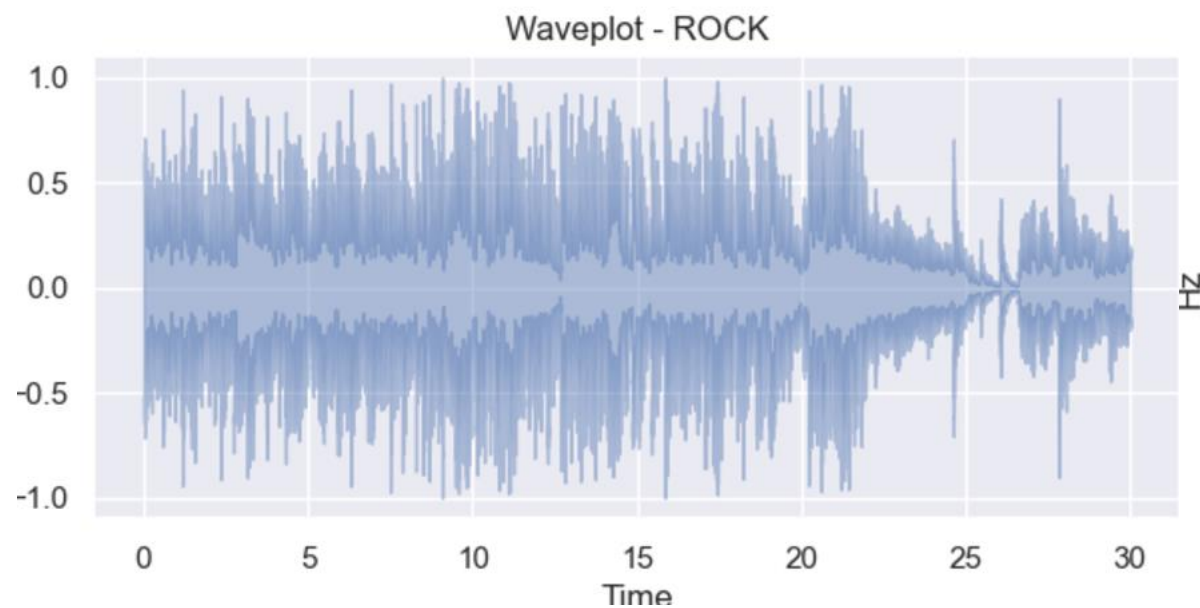
9. Reggae



Exploratory Data Analysis

26

10. Rock



Train and Test Data

27

To create the X and y of this dataset, we need to process the data, extract the features and assign the labels:

```
path = 'Data/genres_original'
```

Process the data:

```
def process_data(data_path):
    X = []
    y = []

    for genre in os.listdir(path):
        genre_path = os.path.join(path, genre)
        if os.path.isdir(genre_path):
            print(f"Processing {genre} files...")

            for file_name in os.listdir(genre_path):
                if file_name.endswith('.wav'):
                    file_path = os.path.join(genre_path, file_name)
                    extracted_features = extract_features(file_path)

                    if extracted_features:
                        X.append(extracted_features)
                        y.append(genre)

    return np.array(X), np.array(y)
```

Extract the features:

```
def extract_features(file_path):
    try:
        audio, sr = librosa.load(file_path)

        mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
        spectral_centroid = librosa.feature.spectral_centroid(y=audio, sr=sr)
        chroma = librosa.feature.chroma_stft(y=audio, sr=sr)

        features = []
        for feature in [mfccs, spectral_centroid, chroma]:
            features.extend([
                np.mean(feature),
                np.std(feature),
                np.max(feature),
                np.min(feature)
            ])

        return features
    except Exception as e:
        print(f"Error extracting features from {file_path}: {str(e)}")
        return None
```

Train and Test Data

28

```
X, y = process_data(path)
```

```
Processing blues files...  
Processing classical files...  
Processing country files...  
Processing disco files...  
Processing hiphop files...  
Processing jazz files...  
Processing metal files...  
Processing pop files...  
Processing reggae files...  
Processing rock files...
```

```
print(X.shape, y.shape)
```

```
(999, 12) (999,)
```

Now we need to split the data:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
print(X_train.shape, y_train.shape)
```

```
(799, 12) (799,)
```

```
print(X_test.shape, y_test.shape)
```

```
(200, 12) (200,)
```



Support Vector Machine

Support Vector Machine (SVM)

30

SVM are a set of **supervised learning** methods used for **classification**, **regression** and **outliers' detection**.

The advantages of SVM are:

- Effective in high dimensional spaces;
- Effective in cases where number of dimensions is greater than the number of samples;
- Uses a subset of training points in the decision function (*support vectors*), so it is also memory efficient;
- Versatile because different kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of SVM include:

- If the number of features is much greater than the number of samples, choosing kernel functions and regularization term helps avoid overfitting;
- Do not directly provide probability estimates - these are calculated using an expensive 5-fold cross-validation.

Support Vector Classifier

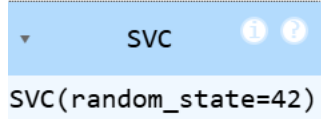
31

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True,
    probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,
    decision_function_shape='ovr', break_ties=False, random_state=None)
```

Let's use default values:

```
classifier = SVC(kernel='rbf', random_state = 42)
```

```
classifier.fit(X_train, y_train)
```



```
SVC
```

```
SVC(random_state=42)
```

Obtain the scores:

```
print("Training set score: {:.3f}".format(classifier.score(X_train, y_train)))
print("Test set score: {:.3f}".format(classifier.score(X_test, y_test)))
```

Training set score: 0.390

Test set score: 0.320

Support Vector Classifier

32

Obtain predictions:

```
y_pred = classifier.predict(X_test)
```

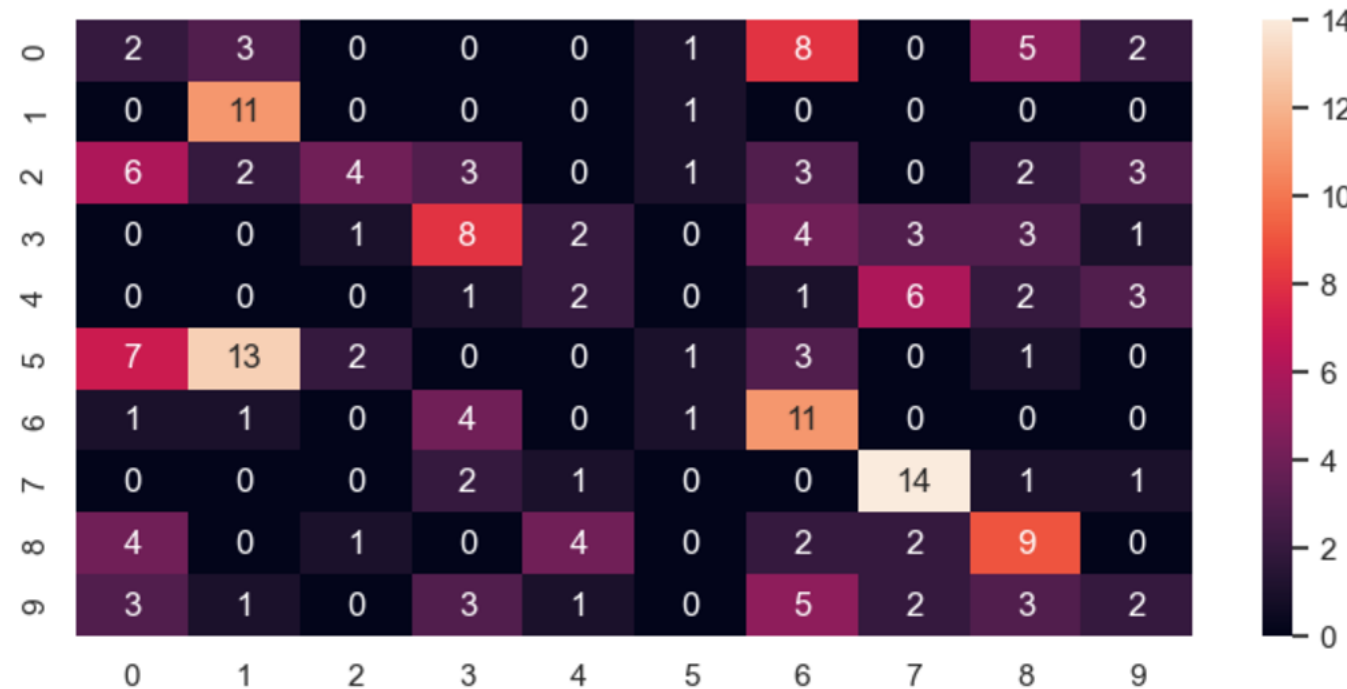
Evaluate the model:

```
cm = confusion_matrix(y_test, y_pred)
```

```
sns.set(rc = {'figure.figsize':(9, 4)})  
sns.heatmap(cm, annot=True)  
print(classification_report(y_test, y_pred))
```

['blues': 0, 'classical': 1, 'country': 2, 'disco': 3,
'hiphop': 4, 'jazz': 5, 'metal': 6, 'pop': 7, 'reggae': 8,
'rock': 9]

	precision	recall	f1-score	support
blues	0.09	0.10	0.09	21
classical	0.35	0.92	0.51	12
country	0.50	0.17	0.25	24
disco	0.38	0.36	0.37	22
hiphop	0.20	0.13	0.16	15
jazz	0.20	0.04	0.06	27
metal	0.30	0.61	0.40	18
pop	0.52	0.74	0.61	19
reggae	0.35	0.41	0.38	22
rock	0.17	0.10	0.12	20
accuracy			0.32	200
macro avg	0.31	0.36	0.30	200
weighted avg	0.31	0.32	0.28	200



Support Vector Classifier

33

Select an audio file:

```
path_test = 'Data/genres_original/hiphop/hiphop.00004.wav'
```

```
audio_test, sr = librosa.load(path_test)
```

```
ipd.Audio(audio_test, rate=sr)
```

▶ 0:00 / 0:30 ———— 🔊 ⋮

How can the model be improved?

Apply the SVC to the file:

```
a_test=[]
mfccs = librosa.feature.mfcc(y=audio_test, sr=sr, n_mfcc=13)
spectral_centroid = librosa.feature.spectral_centroid(y=audio_test, sr=sr)
chroma = librosa.feature.chroma_stft(y=audio_test, sr=sr)
features = []
for feature in [mfccs, spectral_centroid, chroma]:
    features.extend([
        np.mean(feature),
        np.std(feature),
        np.max(feature),
        np.min(feature)
    ])
a_test.append(features)
```

```
print(classifier.predict(np.array(a_test)))
```

```
['reggae']
```



Hands On