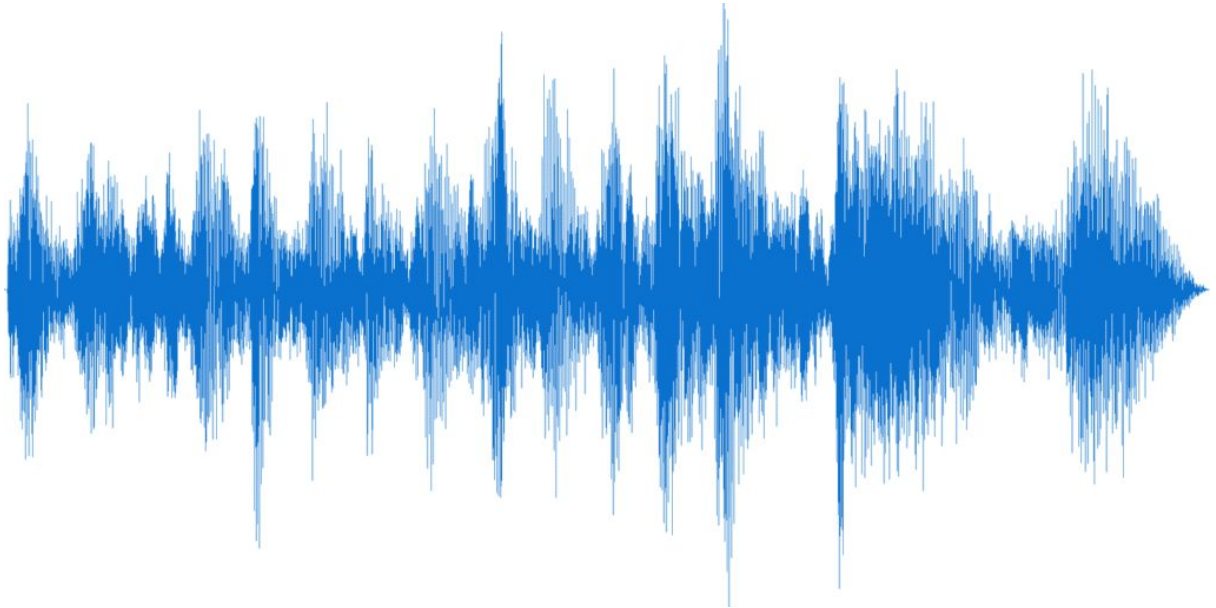
Audio Processing



Bushra Amjad

Audio data analysis

Audio data analysis is about analyzing and understanding audio signals captured by digital devices, with numerous applications in the enterprise, healthcare, productivity, and smart cities.

Applications include

- customer satisfaction analysis from customer support calls,
- media content analysis and retrieval,
- medical diagnostic aids
- patient monitoring
- assistive technologies for people with hearing impairments
- audio analysis for public safety.

Existing products

- virtual assistants Alexa
- Siri
- Google Home

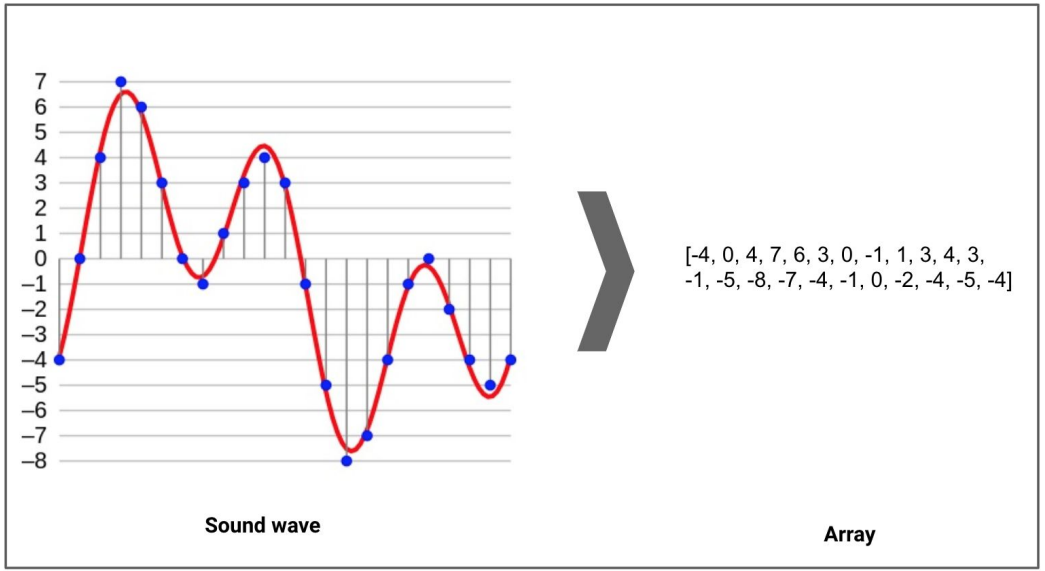
Audio file overview

- The sound snippet are digital audio files in **.wav format** (Waveform Audio File).
- Sound waves are digitized by sampling them at discrete intervals known as the sampling rate
- Each sample is the amplitude of the wave at a particular time interval

What is Sampling and Sampling frequency?

In signal processing, sampling is the reduction of a continuous signal into a series of discrete values.

The sampling frequency or rate is the number of samples taken over some fixed amount of time.



Applications of Audio Processing

- Indexing music collections according to their audio features.
- Recommending music for radio channels
- Similarity search for audio files (aka Shazam)
- Speech processing and synthesis — generating artificial voice for conversational agents

Audio Data Handling using Python

- Sound is represented in the form of an audio signal having parameters such as **frequency, bandwidth, decibel**, etc.
- A typical audio signal can be expressed as a f**unction of Amplitude and Time**.
- There are devices built that help you catch these sounds and represent it in a computer-readable format. Examples of these formats are
  - wav (Waveform Audio File) format
  - mp3 (MPEG-1 Audio Layer 3) format
  - WMA (Windows Media Audio) format

Librosa

Loading an audio file

```
import librosa
audio_data = '/../../gruesome.wav'
x , sr = librosa.load(audio_data)
print(type(x), type(sr))#<class 'numpy.ndarray'> <class 'int'>print(x.shape, sr)#(9431
```

This returns an audio time series as a numpy array with a default sampling rate(sr) of 22KHZ mono. We can change this behavior by resampling at 44.1KHz.

```
librosa.load(audio_data, sr=44100)
```

Playing audio

Using,**`IPython.display.Audio`** you can play the audio in your jupyter notebook.

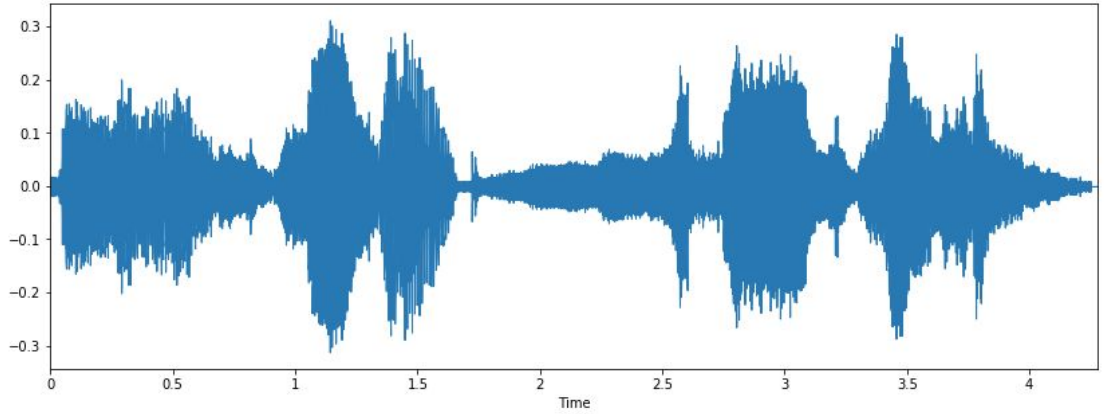
```
import IPython.display as ipd
ipd.Audio(audio_data)
```

This returns an audio widget:

Visualizing Audio

We can plot the audio array using **librosa.display.waveplot**:

```
import matplotlib.pyplot as plt
import librosa.display
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```



Spectrogram

- A spectrogram is a visual way of representing the **signal strength, or “loudness”**, of a signal over time at various frequencies present in a particular waveform.
- A spectrogram is usually depicted as a **heat map**, i.e., as an image with the **intensity shown by varying the color or brightness.**

We can display a spectrogram using. **librosa.display.specshow**.

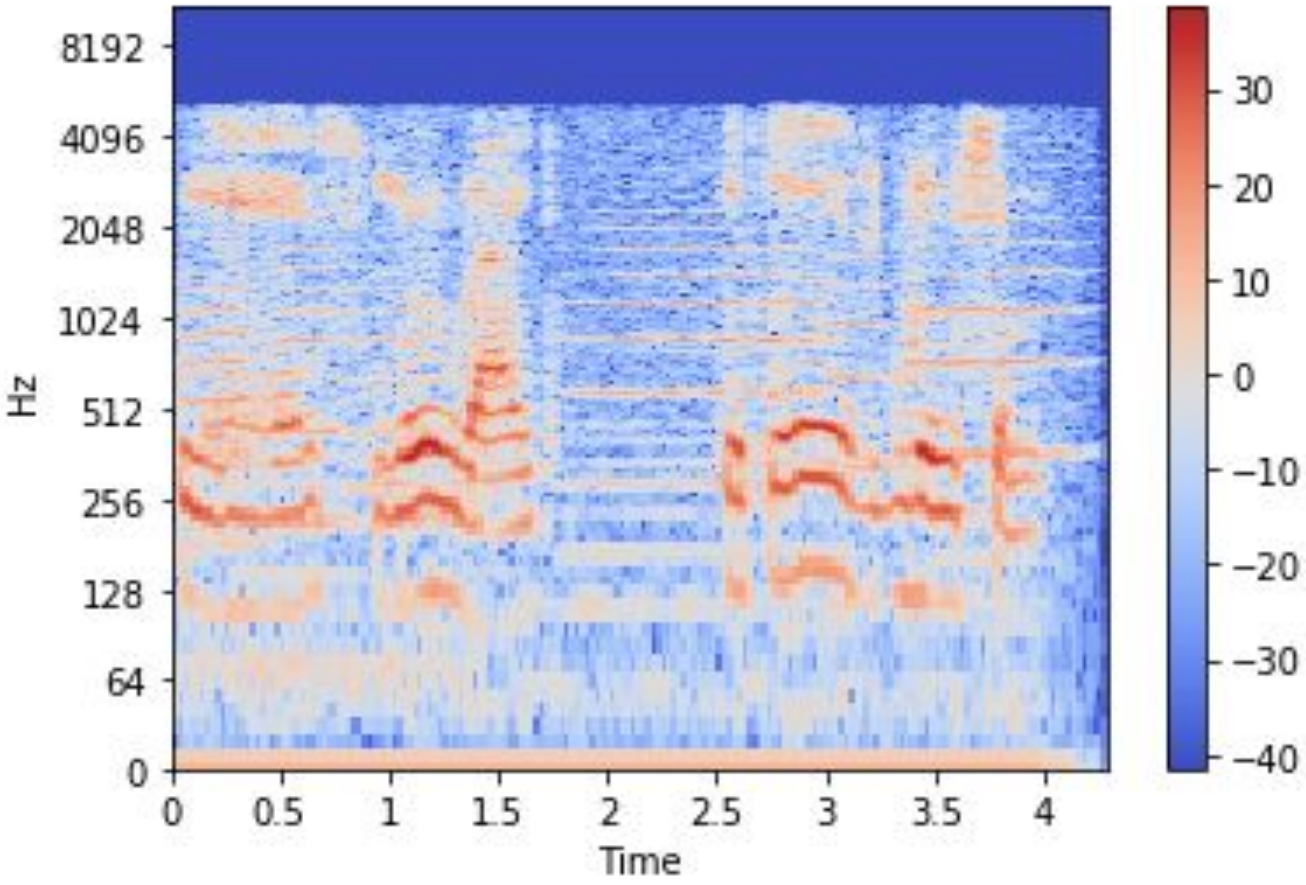
```
X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```

.stft() converts data into short term Fourier transform.

STFT converts signals such that we can know the amplitude of the given frequency at a given time.

Using STFT we can determine the amplitude of various frequencies playing at a given time of an audio signal.

.specshow is used to display a spectrogram.


8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
2
2.5
3
3.5
4
Time
30
20
10
0
−10
−20
−30
−40

Feature extraction from Audio signal

The process of extracting features to use them for analysis is called feature extraction.

Mel-Frequency Cepstral Coefficients(MFCCs)

```
mfccs = librosa.feature.mfcc(x, sr=fs)
print(mfccs.shape)
(20, 97)
#Displaying  the MFCCs:
plt.figure(figsize=(15, 7))
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```