# AASD 4004
# Machine Learning - II

Applied AI Solutions Developer Program

# Module 14
# Introduction to Audio Processing

Vejey Gandyer

# Agenda

- Audio Processing
- Why
- Basic Concepts
- Tools for Audio Processing
- Pyaudio
- Librosa
- Pydub
- Applications

# Audio Processing

What is it?

# Audio Processing

Audio Processing means changing the characteristics of an audio signal in some way

# Why

# Why to process Audio?

To enhance audio

To separate channels

To create new sounds

To store sounds

# Basic Concepts

# Basic Concepts

Frequency

Sampling Rate

Sine wave

Amplitude

Discrete Fourier Transform - Calculates which frequencies are present, Converts time domain signal to a frequency domain

Noise Filtering

# Sound

Sound is a travelling vibration

2 basic attributes

Amplitude (loudness)

Frequency (measure of wave's vibration per unit time)

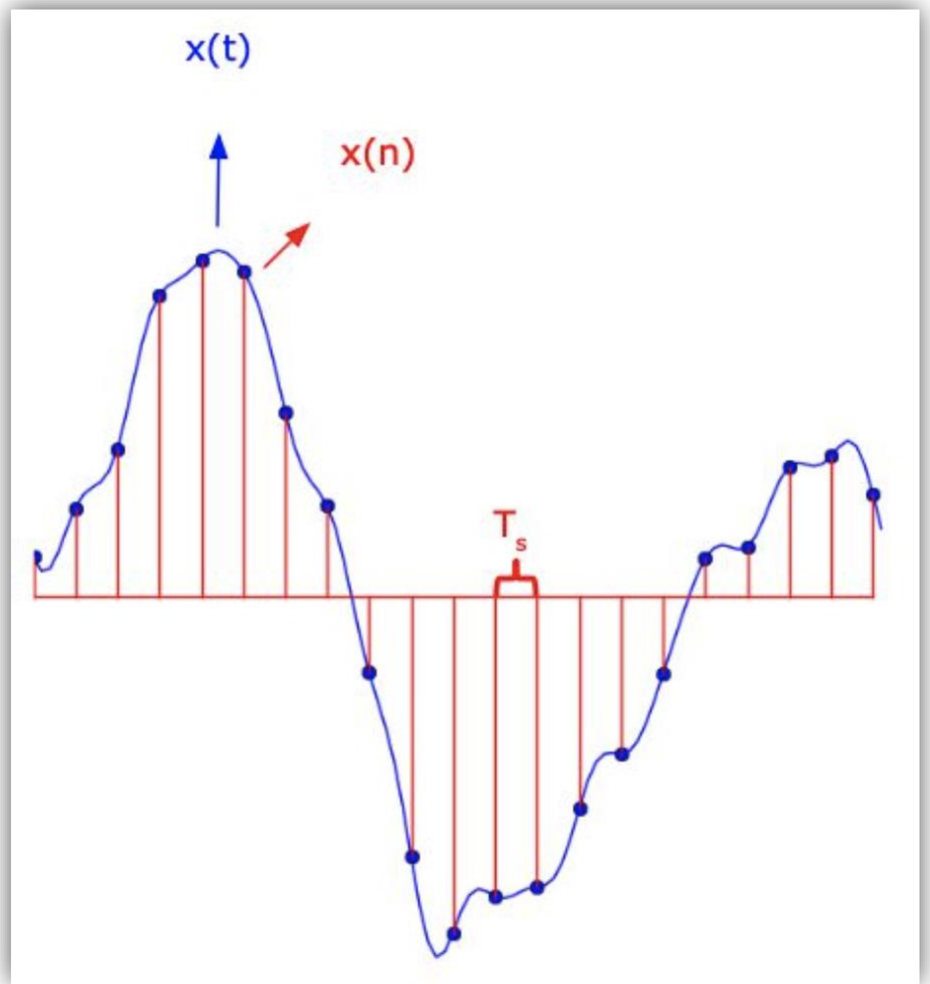
# Analog to Digital Conversion

Sampling - Convert the time-varying continuous signal x(t) to a discrete sequence x(n) of real numbers

Sampling period $T_s$ - Interval between successive discrete samples

Sampling frequency - $f_s = 1 / T_s$

Quantization - Replacing each real number of the sequence of samples with an approximation from a finite set of discrete values



# Sine wave

## Sine wave formula

```
y(t) = A * sin(2 * pi * f * t)
```

# Frequency

Number of times a sine wave repeats a second

# Sampling rate

Real world signals are analog signals

In digitizing it, converting analog to digital, takes a sample at defined intervals

Eg: 48000

# Amplitude

Height of the sine wave

```
y(t) = A * sin(2 * pi * f * t)
```

# Let's create a sine wave

Create a sine wave

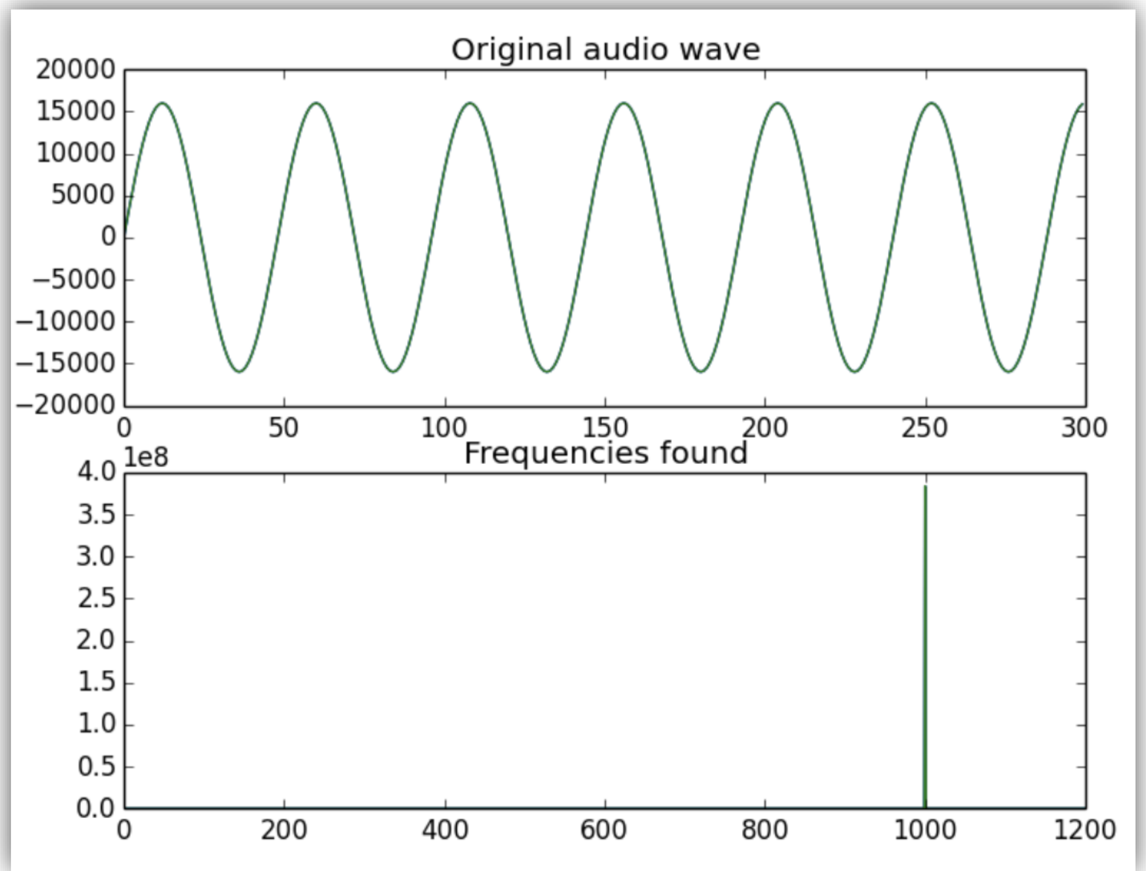
Write the sine wave in an audio file (.wav)

```
sine_wave = [np.sin(2 * np.pi * frequency * x/sampling_rate) for x in range(num_samples)]
wav_file=wave.open(file, 'w')
wav_file.setparams((nchannels, sampwidth, int(sampling_rate), nframes, comptype, compname))
for s in sine_wave:
    wav_file.writeframes(struct.pack('h', int(s*amplitude)))
```

# Discrete Fourier Transform (DFT)

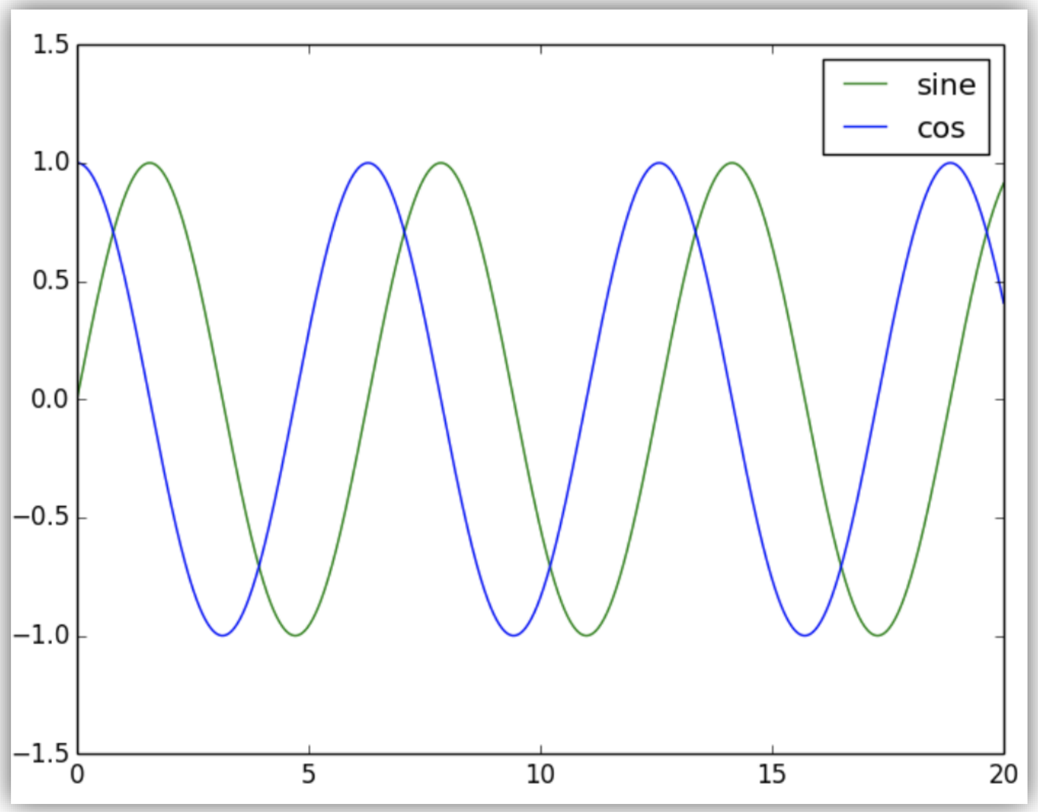
Identifies what are the frequencies present in the signal

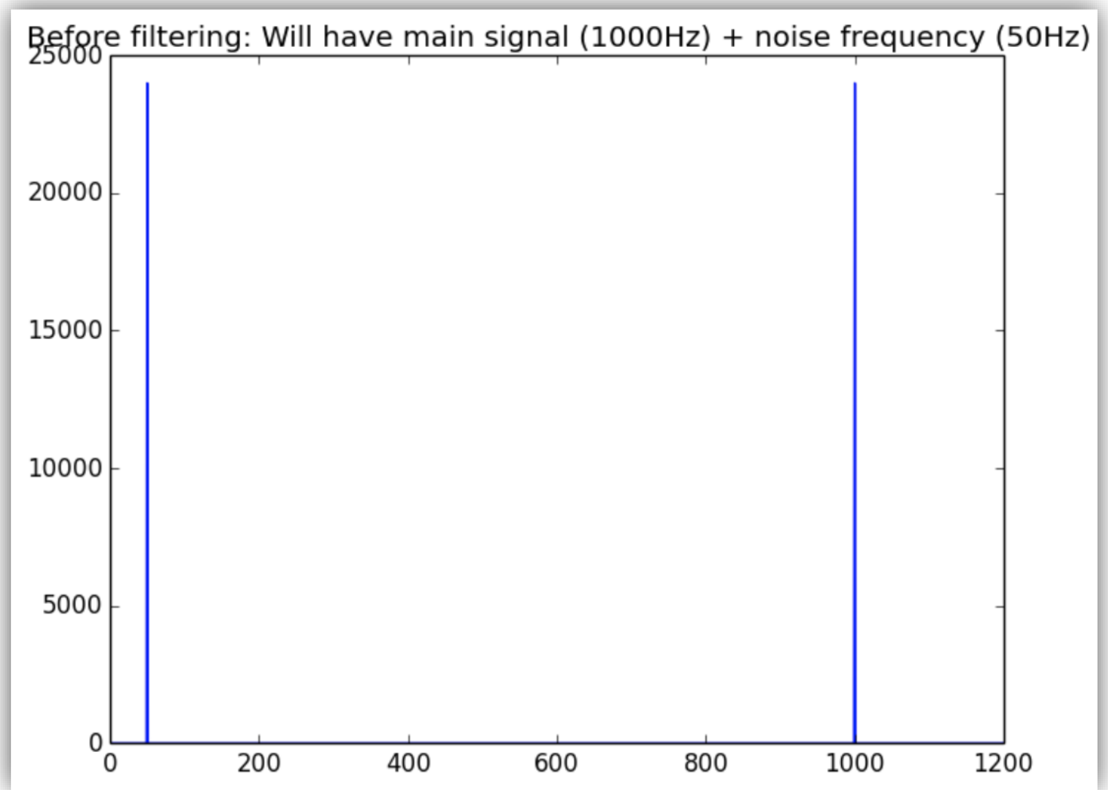
```
data_fft = np.fft.fft(data)
```



# Discrete Fourier Transform (DFT)

Identifies what are the frequencies present in the signal



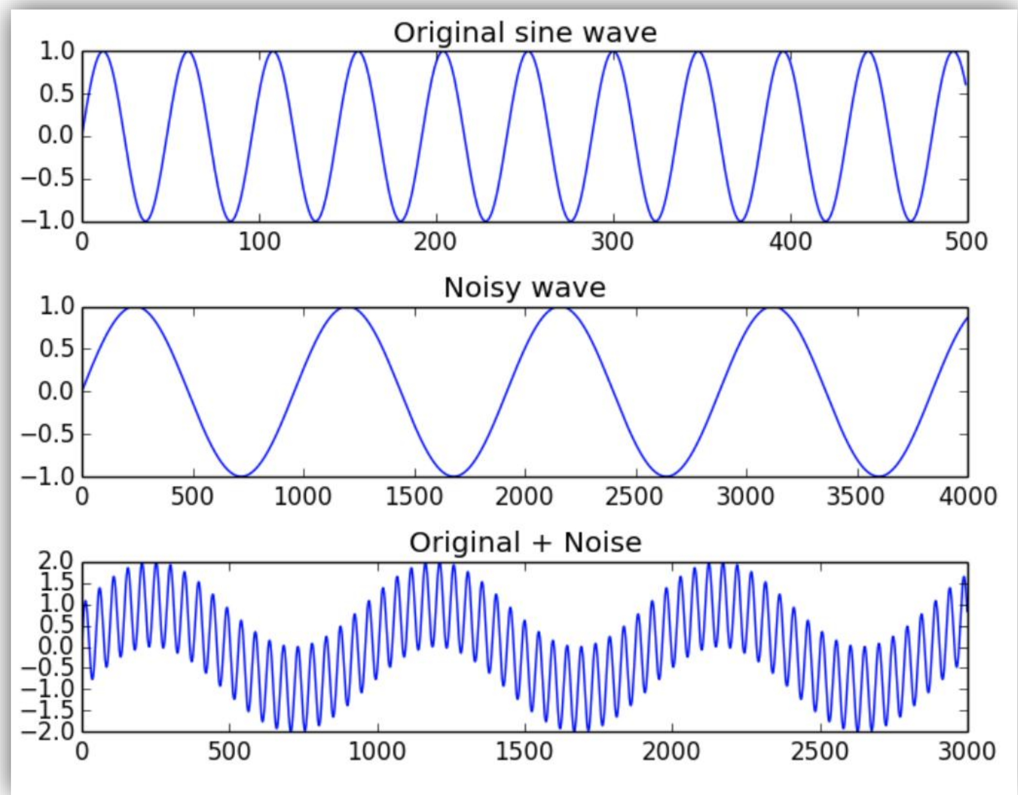



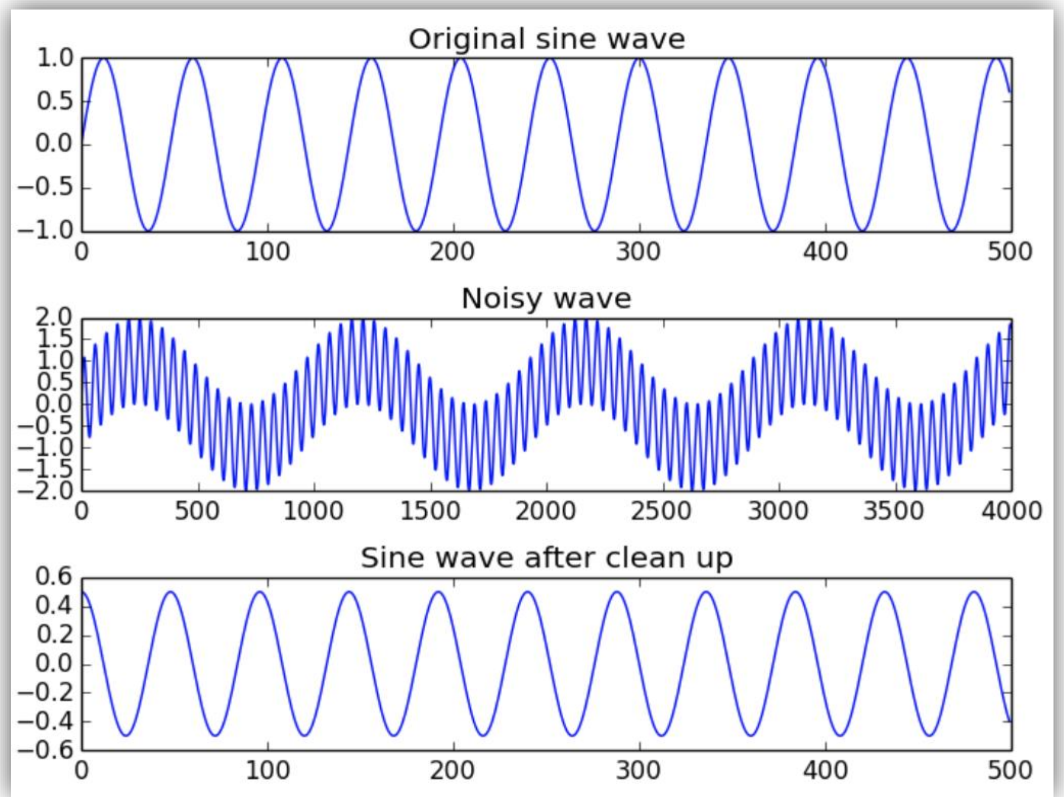
# Cleaning a noisy signal (sine wave)

Generate a sine wave (signal)

Add noise

Filter the noise



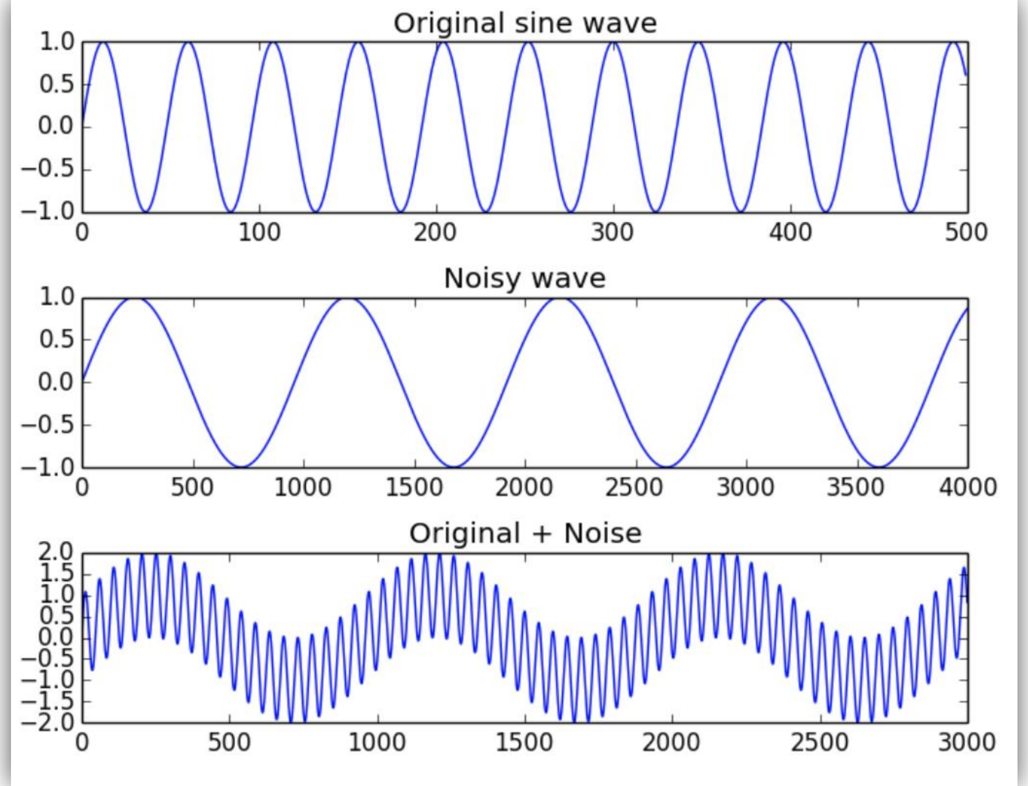
# Cleaning a noisy signal (sine wave)

Generate a sine wave (signal)

Add noise

Filter the noise

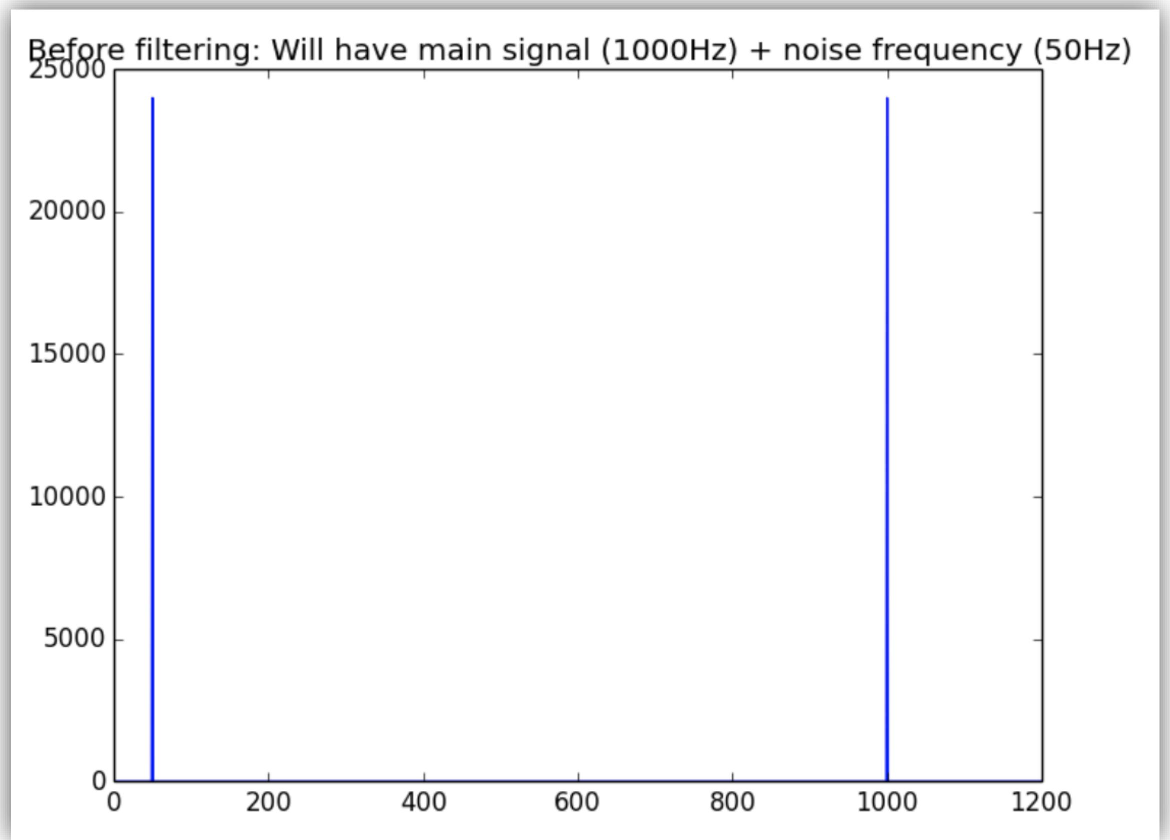
```
frequency = 1000
noisy_freq = 50
num_samples = 48000
sampling_rate = 48000.0
```



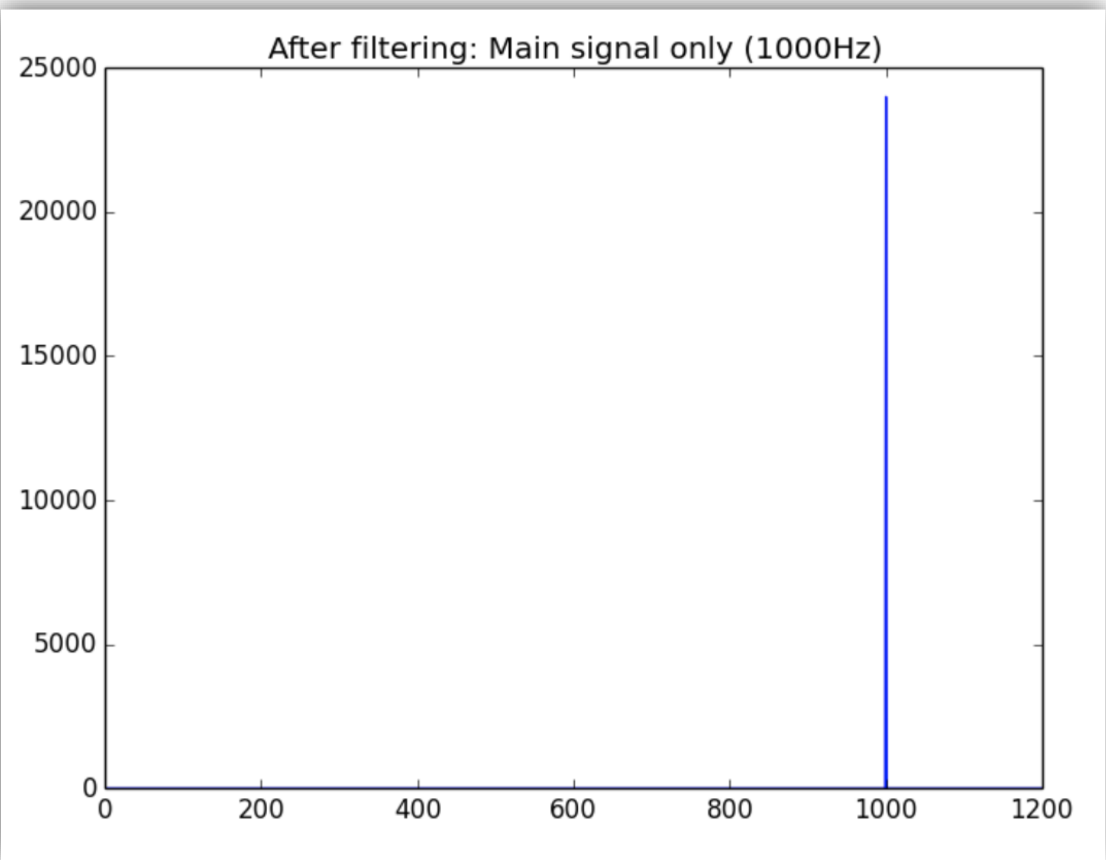

```
sine_wave = [np.sin(2 * np.pi * frequency * x1 / sampling_rate) for x1 in range(num_samples)]
sine_noise = [np.sin(2 * np.pi * noisy_freq * x1/  sampling_rate) for x1 in range(num_samples)]
```

# Cleaning a noisy signal (sine wave)

Noisy signal




Clean signal



# Tools for Audio Processing

# Tools

**ffmpeg/libav** - Handling multimedia files and streams

**sox** - Swiss Army knife of sound processing programs

**audacity** - Editing and playback software

**pyAudioAnalysis** - IO, advanced feature extraction and signal analysis

**librosa** - Nice library for audio analysis

**pydub** - Library for audio processing

# Pydub

https://github.com/jiaaro/pydub

# Loading a wav file

## pydub

```
from pydub import AudioSegment
import numpy as np
audiofile = AudioSegment.from_file("data/music_8k.mp3")
data_mp3 = np.array(audiofile.get_array_of_samples())
```

## scipy

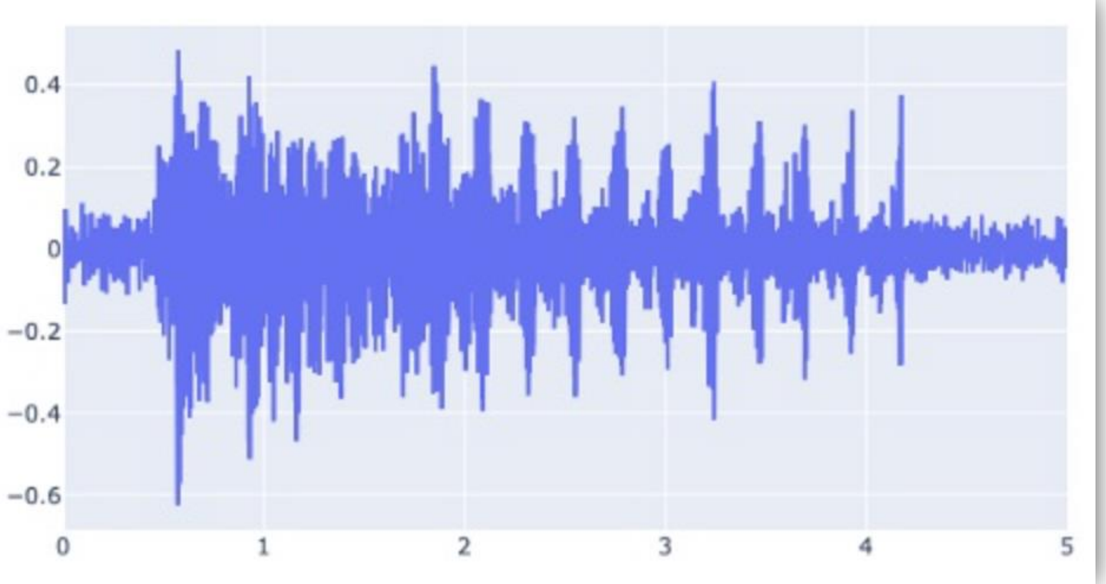
```
from scipy.io import wavfile
fs_wav, data_wav = wavfile.read("data/music_8k.wav")
```

# Normalization

```
fs_wav, data_wav = wavfile.read("data/lost_highway_small.wav")
data_wav_norm = data_wav / (2**15)
time_wav = np.arange(0, len(data_wav)) / fs_wav
plotly.offline.iplot({ "data": [go.Scatter(x=time_wav,
                                           y=data_wav_norm,
                                           name='normalized audio signal')]})
```

Makes the signal values independent to the sample resolution

Squish the values of an audio signal in the (-1, 1) by dividing by $2^{15}$

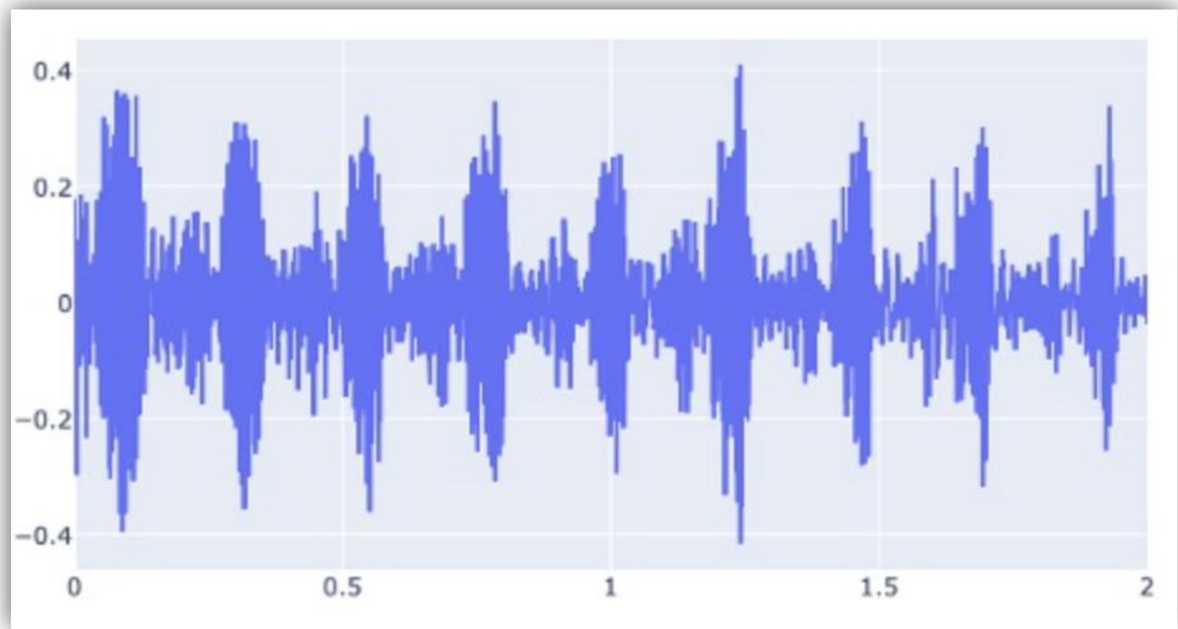


# Trim/Clip/Segment audio clips

```
data_wav_norm_crop = data_wav_norm[2 * fs_wav: 4 * fs_wav]
time_wav_crop = np.arange(0, len(data_wav)) / fs_wav
plotly.offline.iplot({ "data": [go.Scatter(x=time_wav_crop,
                                           y=data_wav_norm_crop,
                                           name='cropped audio signal')]})
```

Clips the audio file portions by referring to the respective indices in the numpy array

Seconds need to be multiplied by the sampling frequency



# Split audio into fixed-size segments

```
fs, signal = wavfile.read("data/obama.wav")
signal = signal / (2**15)
signal_len = len(signal)
segment_size_t = 1 # segment size in seconds
segment_size = segment_size_t * fs  # segment size in samples
# Break signal into list of segments in a single-line Python code
segments = np.array([signal[x:x + segment_size] for x in
                     np.arange(0, signal_len, segment_size)])
# Save each segment in a seperate filename
for iS, s in enumerate(segments):
    wavfile.write("data/obama_segment_{0:d}_{1:d}.wav".format(segment_size_t * iS,
                                                              segment_size_t * (iS + 1)), fs, (s))
```

# Remove silent segments from audio

Computes energy as the sum of squares of the samples

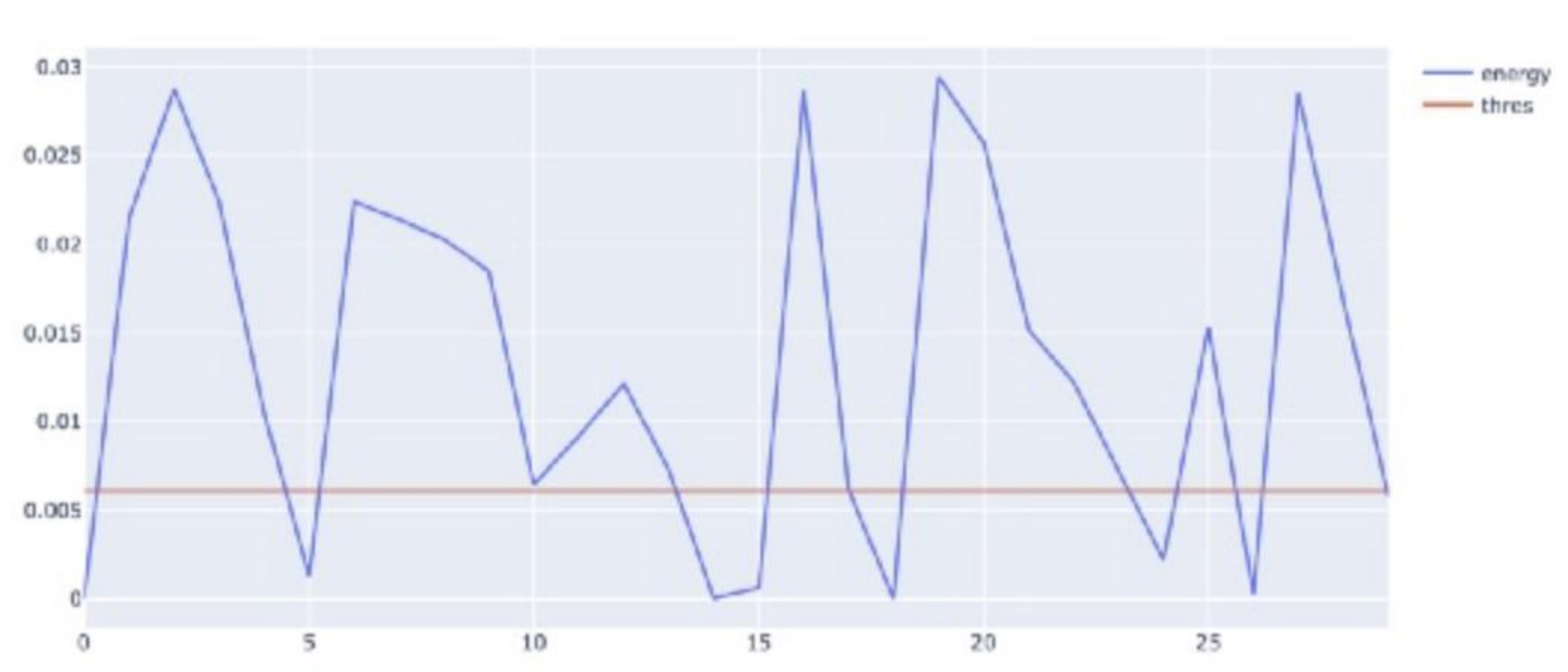
Calculates a threshold as 50% of the median energy value

Keeps the segments whose energy are above the set threshold

```
energies = [(s**2).sum() / len(s) for s in segments]
thres = 0.5 * np.median(energies)
index_of_segments_to_keep = (np.where(energies > thres)[0])
segments2 = segments[index_of_segments_to_keep]
new_signal = np.concatenate(segments2)
wavfile.write("data/obama_processed.wav", fs, new_signal)
```

# Remove silent segments from audio

Keeps the segments whose energy are above the set threshold



# Remove silent segments from audio

Original

Processed

# PyAudio

# PyAudio - Installation

Brew

```
brew install portaudio
```

PyPI

```
pip install pyaudio
```

Conda

```
conda install -c conda-forge pyaudio
```

# Recording & Visualizing Frequencies

Goals:

Capture sound by a microphone in real-time

Visualize the frequencies present in real-time

# Recording & Visualizing Frequencies

1. Capture the sound in fixed-size segments (200 ms)
2. For each segment, plot the frequency distribution in real-time
    1. Compute magnitude X of FFT of the segment and frequency values (in Hz) in a separate array freqs
    2. Downsample X and freqs, so that we keep very few frequency coefficients to visualize
    3. Compute total segment's energy (to normalize against the maximum width of the frequency visualization)
    4. Bar Plot downsampled frequency energies X for all frequencies

# Recording & Visualizing Frequencies

Capture the sound in fixed-size segments (200 ms)

```
pa = pyaudio.PyAudio()
stream = pa.open(format=pyaudio.paInt16, channels=1, rate=fs,
                 input=True, frames_per_buffer=int(fs * buff_size))
```

# Recording & Visualizing Frequencies

Compute magnitude X of FFT of the segment and frequency values (in Hz) in a separate array freqs

```
X = np.abs(scp.fft(x))[0:int(seg_len/2)]
freqs = (np.arange(0, 1 + 1.0/len(X), 1.0 / len(X)) * fs / 2)
```

# Recording & Visualizing Frequencies

Downsample X and freqs, so that we keep very few frequency  coefficients to visualize

```
wanted_step = (int(freqs.shape[0] / wanted_num_of_bins))
freqs2 = freqs[0::wanted_step].astype('int')
X2 = np.mean(X.reshape(-1, wanted_step), axis=1)
```

# Recording & Visualizing Frequencies

Bar Plot downsampled frequency energies X for all frequencies

```
fig = tpl.figure()
fig.barh(X2, labels=[str(int(f)) + " Hz" for f in freqs2[0:-1]],
            show_vals=False, max_width=max_width_from_energy)
fig.show()
```

# Recording & Visualizing Frequencies

Demo

# Librosa

https://librosa.org/doc/latest/index.html

# Librosa - Installation

PyPI

```
pip install librosa
```

Conda

```
conda install –c conda-forge librosa
```

Source

```
tar xzf librosa-VERSION.tar.gz
cd librosa-VERSION/
python setup.py install
```

# Librosa - Submodules

**beat** - Estimating tempo and detecting beat events

**Core** - Load audio, compute spectrograms and other analysis

**Decompose** - Harmonic-percussive source separation (HPSS)

**Display** - display and visualization routines

**Effects** - Pitch Shifting, Time-stretching

**Feature** - Chromagrams, Mel Spectrogram, MFCC

**Filters** - Filter-bank generation (Chroma, pseudo-CQT, CQT)

**Onset** - Onset detection and Onset strength computation

# Librosa  - Quick Example (Beats)

Importing library

```
import librosa
```

Get the audio file

```
filename = librosa.example('nutcracker')
```

Load the audio file

```
y, sr = librosa.load(filename)
```

Beat tracker

```
tempo, beat_frames = librosa.beat.beat_track(y=y, sr=sr)
```

Tempo

```
print('Estimated tempo: {:.2f} beats per minute'.format(tempo))
```

Frame numbers into timestamps

```
beat_times = librosa.frames_to_time(beat_frames, sr=sr)
```

# Beats Generator

# Beats Generator - Tempo tracking

Task of automatically estimating a song's temp (in beats per minute) directly from the signal using librosa library

Input: Mono audio file

Output: Stereo file

Left channel: original song

Right channel: "beep" sound mimicing the tempo of the song

# Beats Generator - Tempo tracking

```
[Fs, s] = wavfile.read('data/music_44100.wav')
tempo, beats = librosa.beat.beat_track(y=s.astype('float'), sr=Fs, units="time")
beats -= 0.05
s = s.reshape(-1, 1)
s = np.array(np.concatenate((s, np.zeros(s.shape)), axis=1))
for ib, b in enumerate(beats):
    t = np.arange(0, 0.2, 1.0 / Fs)
    amp_mod = 0.2 / (np.sqrt(t)+0.2) - 0.2
    amp_mod[amp_mod < 0] = 0
    x = s.max() * np.cos(2 * np.pi * t * 220) * amp_mod
    s[int(Fs * b):
      int(Fs * b) + int(x.shape[0]), 1] = x.astype('int16')
wavfile.write("data/music_44100_with_tempo.wav", Fs, np.int16(s))
```

# Beats Generator

Sample 1

Sample 2

# PyAudioAnalysis

https://github.com/tyiannak/pyAudioAnalysis

# PyAudioAnalysis - Installation

PyPI

```
pip install PyAudioAnalysis
```

Conda

```
conda install -c conda-forge PyAudioAnalysis
```

Source

```
git clone https://github.com/tyiannak/pyAudioAnalysis.git
cd pyAudioAnalysis
pip install -r ./requirements.txt
```

# Further Reading

Basic Concepts in Audio Processing

https://www.pythonforengineers.com/audio-and-digital-signal-processingdsp-in-python/

https://www.ee.iitb.ac.in/student/~daplab/publications/chapter9-prao.pdf

Librosa

https://librosa.org/doc/latest/index.html

PyAudioAnalysis

https://github.com/tyiannak/pyAudioAnalysis

Pydub

https://github.com/jiaaro/pydub