

```
import matplotlib.pyplot as plt
import numpy as np
import librosa
import librosa.display
import IPython.display as ipd
```

## ✓ Loading Audio Files

```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

```
debussy_file = "/content/drive/My Drive/Multimedia/debussy.wav"
redhot_file = "/content/drive/My Drive/Multimedia/redhot.wav"
duke_file = "/content/drive/My Drive/Multimedia/duke.wav"
```


```
ipd.Audio(debussy_file)
```

 0:00 / 0:30

```
ipd.Audio(redhot_file)
```

 0:00 / 0:30

```
ipd.Audio(duke_file)
```

 0:00 / 0:30

```
# load audio files with librosa
debussy, sr = librosa.load(debussy_file)
redhot, _ = librosa.load(redhot_file)
duke, _ = librosa.load(duke_file)
```

## ✓ Root-mean-squared energy with Librosa

```
FRAME_SIZE = 1024
HOP_LENGTH = 512
```

```
rms_debussy = librosa.feature.rms(y=debussy, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
rms_redhot = librosa.feature.rms(y=redhot, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
rms_duke = librosa.feature.rms(y=duke, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
```

```
len(rms_debussy)
```

 1292

## ✓ Visualise RMSE + waveform

```
frames = range(len(rms_debussy))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)
```

```
# rms energy is graphed in red
```

```
plt.figure(figsize=(15, 17))
```

```
ax = plt.subplot(3, 1, 1)
```

```

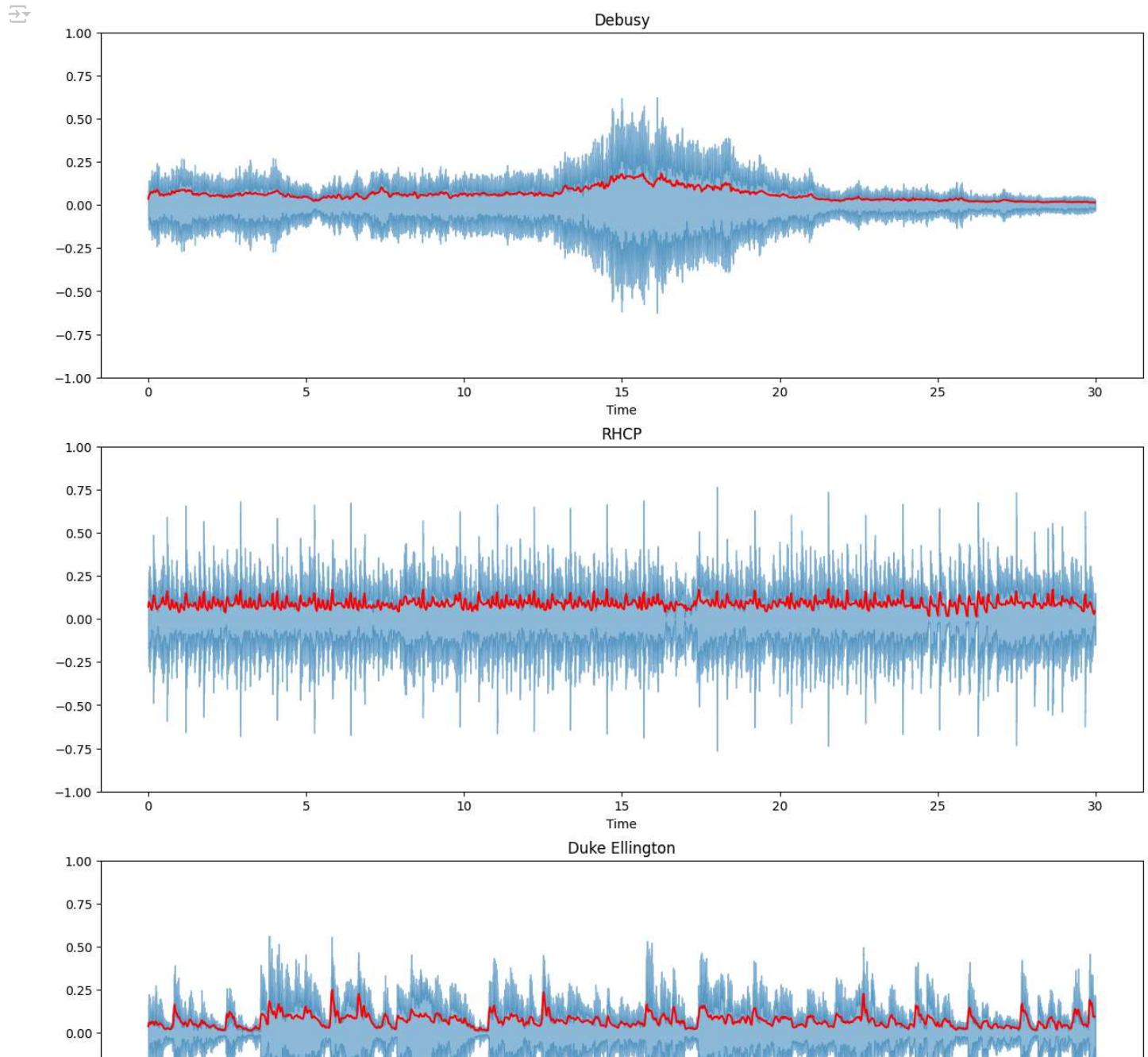
librosa.display.waveshow(debussy, alpha=0.5)
plt.plot(t, rms_debussy, color="r")
plt.ylim((-1, 1))
plt.title("Debussy")

plt.subplot(3, 1, 2)
librosa.display.waveshow(redhot, alpha=0.5)
plt.plot(t, rms_redhot, color="r")
plt.ylim((-1, 1))
plt.title("RHCP")

plt.subplot(3, 1, 3)
librosa.display.waveshow(duke, alpha=0.5)
plt.plot(t, rms_duke, color="r")
plt.ylim((-1, 1))
plt.title("Duke Ellington")

plt.show()

```



✓ RMSE from scratch

Root Mean Square (RMS) is a statistical measure of the magnitude of a signal, often used in audio processing to quantify the energy or loudness of a sound over time. In the context of audio, it represents the square root of the average of the squared values of the signal's amplitudes. It is commonly used to measure the intensity or power of an audio signal.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2}$$

Where: N is the number of data points,

$x$  is the observed value (actual value),

$x(i)$  is the predicted value.

```
def 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_size)
        rmse.append(rmse_current_frame)
    return np.array(rmse)
```

```
rms_debussy1 = rmse(debussy, FRAME_SIZE, HOP_LENGTH)
rms_redhot1 = rmse(redhot, FRAME_SIZE, HOP_LENGTH)
rms_duke1 = rmse(duke, FRAME_SIZE, HOP_LENGTH)
```

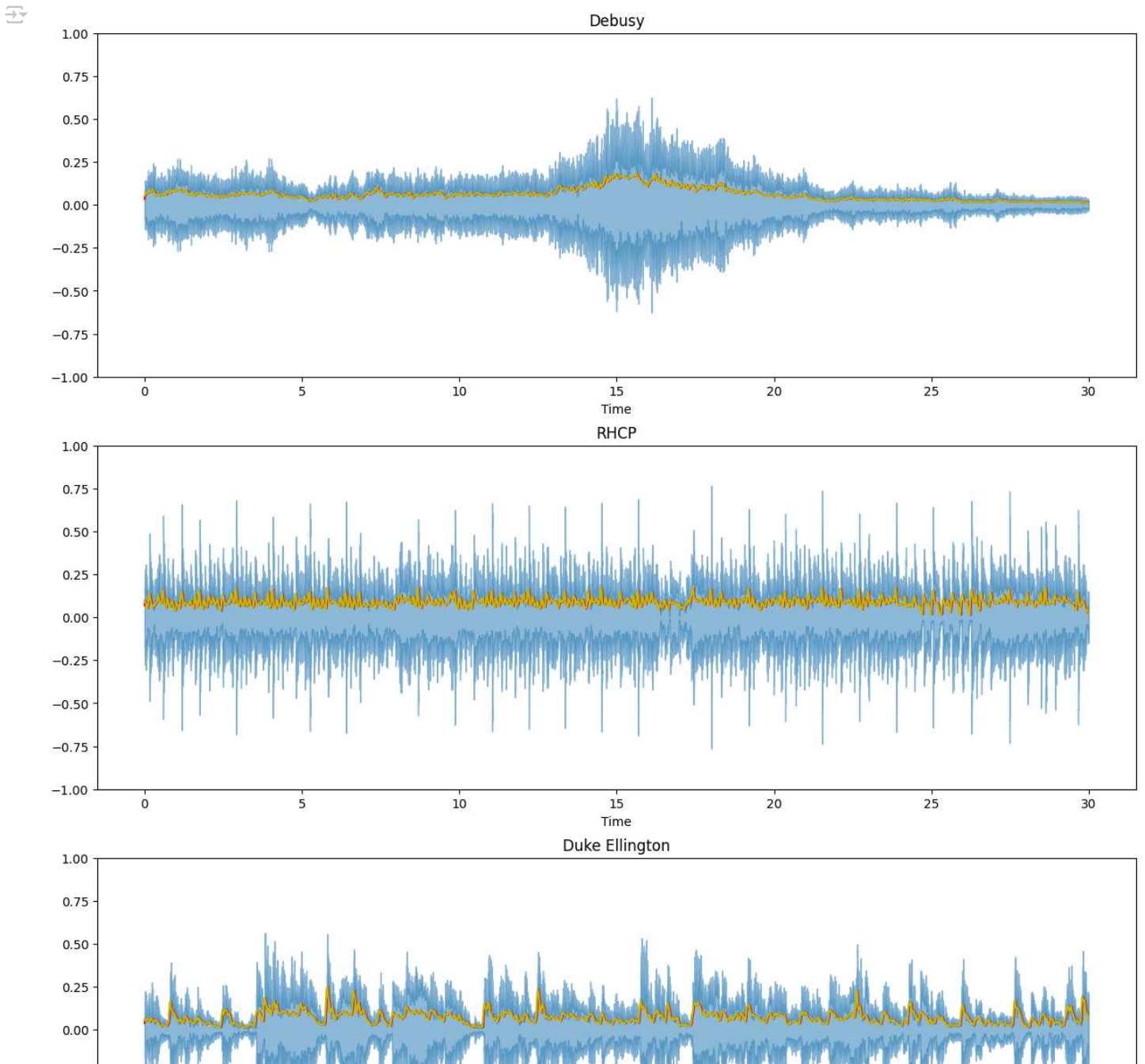
```
plt.figure(figsize=(15, 17))
```

```
ax = plt.subplot(3, 1, 1)
librosa.display.waveshow(debussy, alpha=0.5)
plt.plot(t, rms_debussy, color="r")
plt.plot(t, rms_debussy1, color="y")
plt.ylim((-1, 1))
plt.title("Debussy")
```

```
plt.subplot(3, 1, 2)
librosa.display.waveshow(redhot, alpha=0.5)
plt.plot(t, rms_redhot, color="r")
plt.plot(t, rms_redhot1, color="y")
plt.ylim((-1, 1))
plt.title("RHCP")
```

```
plt.subplot(3, 1, 3)
librosa.display.waveshow(duke, alpha=0.5)
plt.plot(t, rms_duke, color="r")
plt.plot(t, rms_duke1, color="y")
plt.ylim((-1, 1))
plt.title("Duke Ellington")
```

```
plt.show()
```



## Zero-crossing rate with Librosa

Zero Crossing Rate (ZCR) is another important feature used in audio and signal processing. It refers to the rate at which a signal changes its sign, i.e., how often the signal crosses zero. In the context of an audio signal, ZCR can be used to measure the frequency of oscillations, which is useful in distinguishing between different types of sounds, such as speech, music, and noise.

$$ZCR = \frac{1}{N-1} \sum_{i=1}^{N-1} |\text{sign}(x[i]) - \text{sign}(x[i-1])|$$

Where: N is the number of samples in the signal.

$x[i]$  is the value of the signal at sample

$\text{sign}(x)$  is the sign function, which returns:

1 for positive values, -1 for negative values, and 0 for zero values.

```
zcr_debussy = librosa.feature.zero_crossing_rate(debussy, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
zcr_redhot = librosa.feature.zero_crossing_rate(redhot, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
zcr_duke = librosa.feature.zero_crossing_rate(duke, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
```

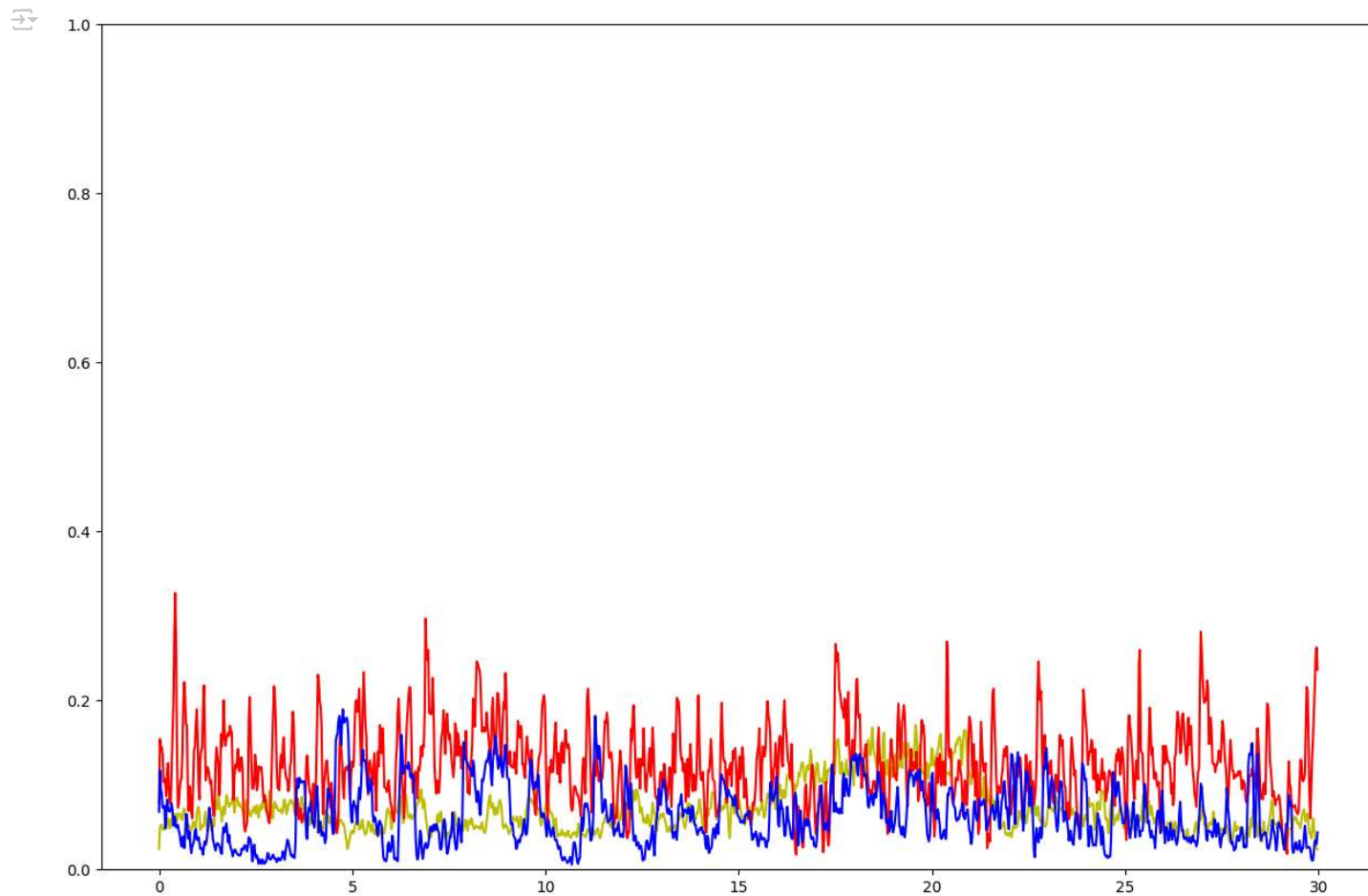
```
zcr_debussy.size
```

```
↔ 1292
```

## Visualise zero-crossing rate with Librosa

```
plt.figure(figsize=(15, 10))

plt.plot(t, zcr_debussy, color="y")
plt.plot(t, zcr_redhot, color="r")
plt.plot(t, zcr_duke, color="b")
plt.ylim(0, 1)
plt.show()
```



## ZCR: Voice vs Noise

```
voice_file = "/content/drive/My Drive/Multimedia/voice.wav"
noise_file = "/content/drive/My Drive/Multimedia/noise.wav"
```

```
ipd.Audio(voice_file)
```

```
↔
```

```
ipd.Audio(noise_file)
```



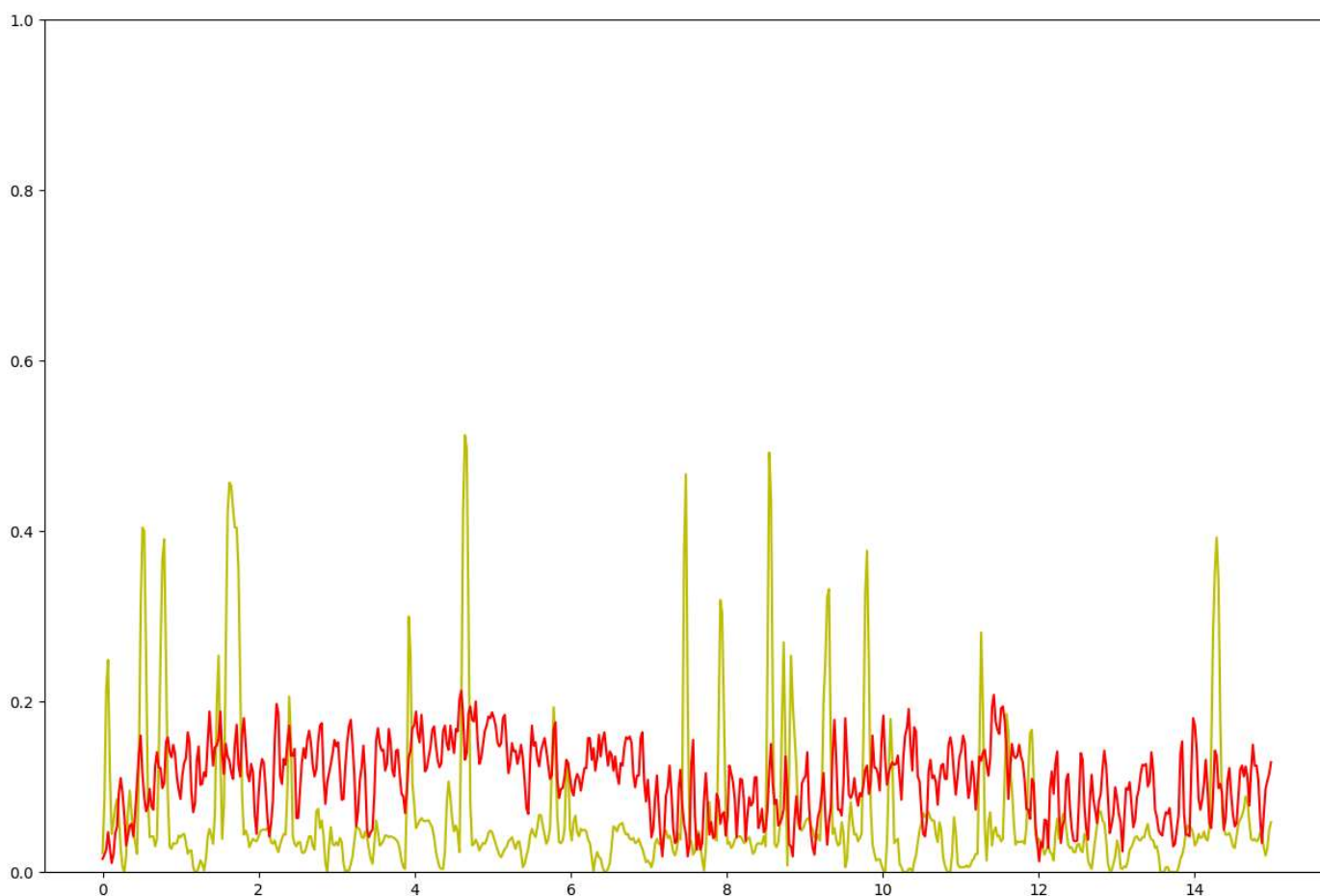
```
# load audio files
voice, _ = librosa.load(voice_file, duration=15)
noise, _ = librosa.load(noise_file, duration=15)

# get ZCR
zcr_voice = librosa.feature.zero_crossing_rate(voice, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
zcr_noise = librosa.feature.zero_crossing_rate(noise, frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]

frames = range(len(zcr_voice))
t = librosa.frames_to_time(frames, hop_length=HOP_LENGTH)

plt.figure(figsize=(15, 10))

plt.plot(t, zcr_voice, color="y")
plt.plot(t, zcr_noise, color="r")
plt.ylim(0, 1)
plt.show()
```



✓ **Skewness** is a statistical measure that quantifies the asymmetry of a probability distribution of a real-valued dataset. It indicates whether the data points are more concentrated on one side of the mean than the other.

$$S = \sum_{i=1}^N \left( \frac{\sigma x_i - \mu}{N} \right)^3$$

where:  $x_i$  = individual data points

$\mu$  = mean of the data

$\sigma$  = standard deviation

$N$  = number of data points

```
from scipy.stats import skew

# Load the audio files
debussy, sr_debussy = librosa.load("/content/drive/My Drive/Multimedia/debussy.wav")
redhot, sr_redhot = librosa.load("/content/drive/My Drive/Multimedia/redhot.wav")
duke, sr_duke = librosa.load("/content/drive/My Drive/Multimedia/duke.wav")

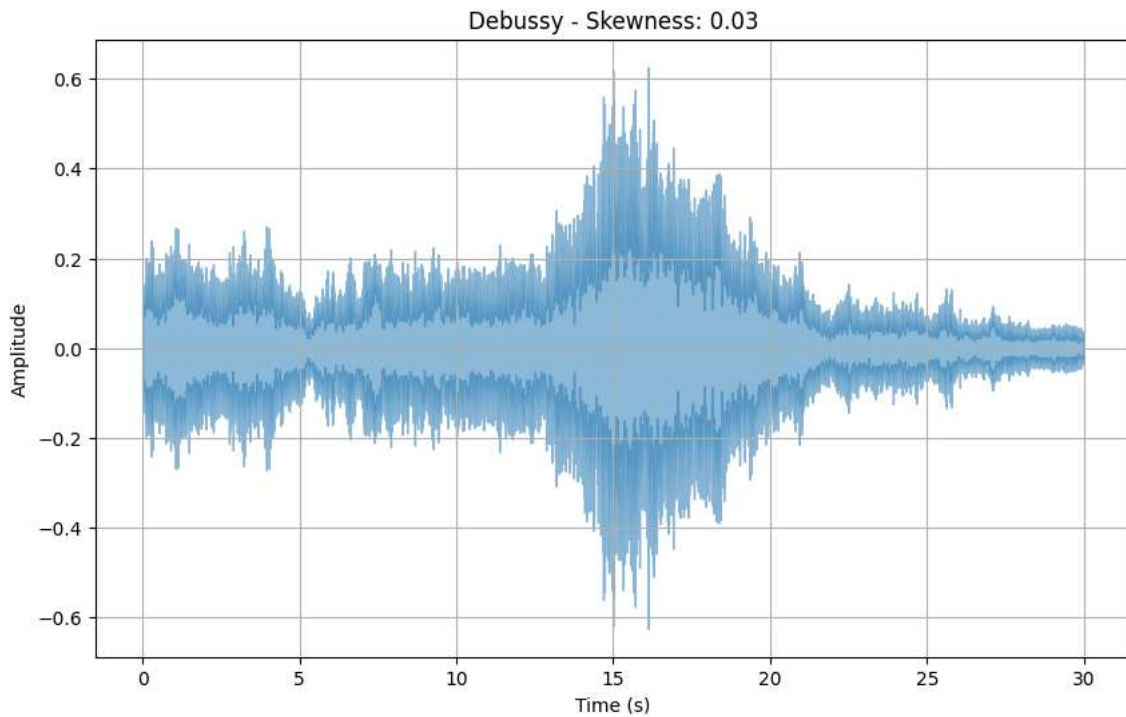
# Function to compute skewness and plot the waveform
def compute_skewness_and_plot(y, sr, title):
    # Compute the skewness of the signal
    signal_skewness = skew(y)

    # Plot the waveform of the audio signal
    plt.figure(figsize=(10, 6))
    librosa.display.waveshow(y, sr=sr, alpha=0.5)
    plt.title(f'{title} - Skewness: {signal_skewness:.2f}')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

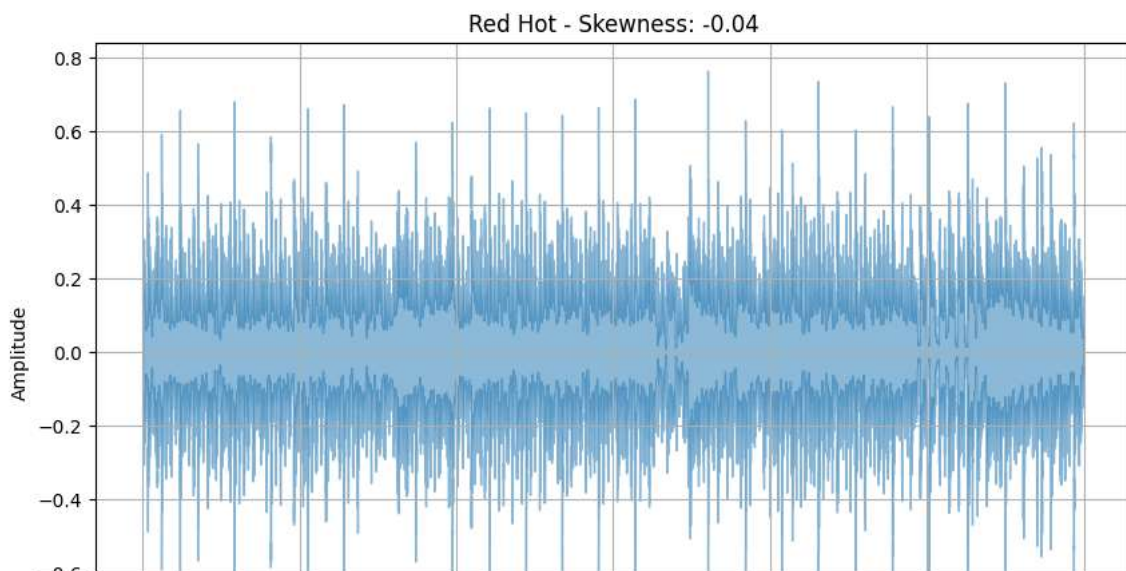
    # Print the skewness value
    print(f"Skewness of the {title} signal: {signal_skewness:.2f}")

# Compute skewness and plot for all three files
compute_skewness_and_plot(debussy, sr_debussy, "Debussy")
compute_skewness_and_plot(redhot, sr_redhot, "Red Hot")
compute_skewness_and_plot(duke, sr_duke, "Duke")
```





Skewness of the Debussy signal: 0.03



A skewness of -0.56 indicates that the distribution of the signal has a slightly longer tail on the left side. This means that there are more signal values that are smaller than the mean.

## Waveform Factor

The Waveform Factor is a time-domain feature that quantifies the "peakedness" or "shape" of a waveform. It provides insight into how much the signal deviates from a simple sinusoidal or periodic signal. Specifically, the waveform factor is defined as the ratio of the Root Mean Square (RMS) value of the signal to its Mean Absolute Value (MAV).

The waveform factor is used to describe the "roughness" or "spikiness" of a signal.

The Waveform Factor (WF) of a signal (  $x[n]$  ) is given by the formula:

$$WF = \frac{MAV(x)}{RMS(x)}$$

e

```
# Load the audio files
debussy, sr_debussy = librosa.load("/content/drive/My Drive/Multimedia/debussy.wav")
redhot, sr_redhot = librosa.load("/content/drive/My Drive/Multimedia/redhot.wav")
```



```
duke, sr_duke = librosa.load("/content/drive/My Drive/Multimedia/duke.wav")

# Function to compute RMS, MAV, and Waveform Factor
def compute_waveform_features(y, sr, title):
    # Calculate the RMS (Root Mean Square) of the signal
    rms = np.sqrt(np.mean(np.square(y)))

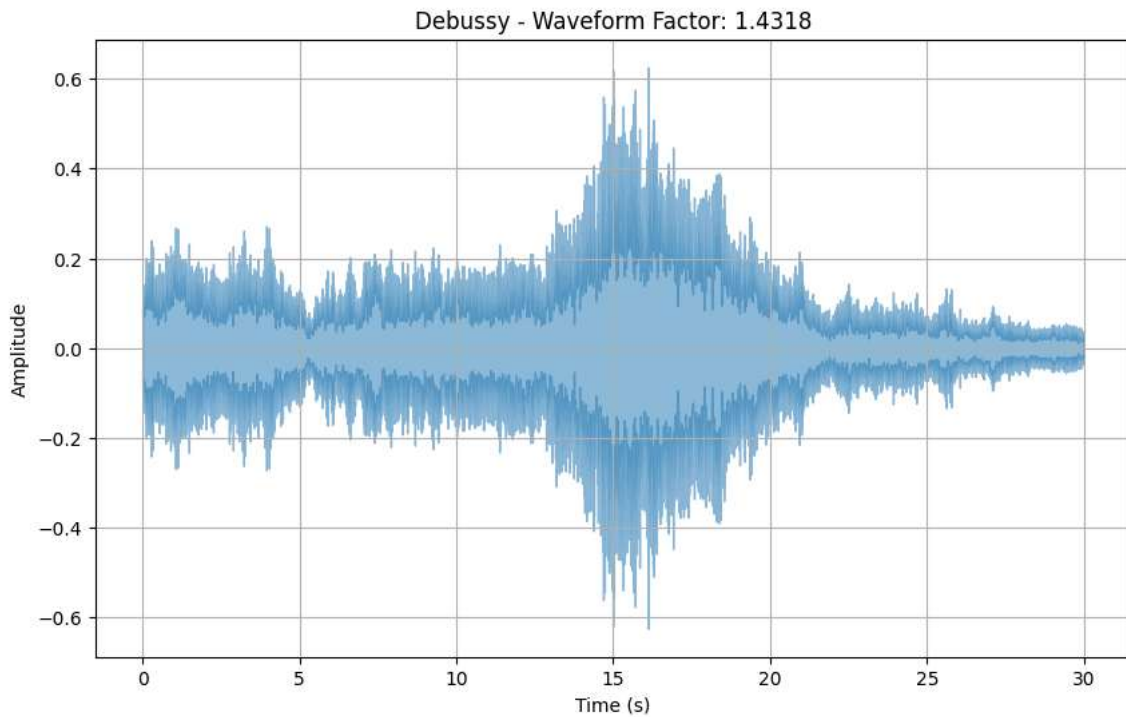
    # Calculate the MAV (Mean Absolute Value) of the signal
    mav = np.mean(np.abs(y))

    # Compute the Waveform Factor
    waveform_factor = rms / mav

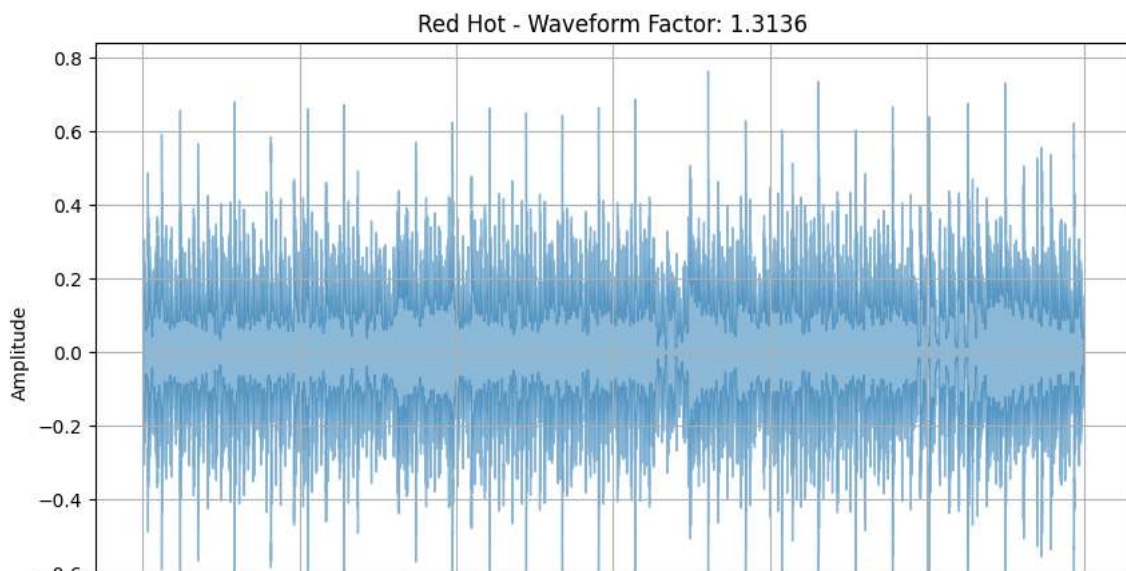
    # Plot the waveform of the audio signal
    plt.figure(figsize=(10, 6))
    librosa.display.waveshow(y, sr=sr, alpha=0.5)
    plt.title(f'{title} - Waveform Factor: {waveform_factor:.4f}')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

    # Print the waveform factor value
    print(f"Waveform Factor of the {title} signal: {waveform_factor:.4f}")

# Compute and plot for all three files
compute_waveform_features(debussy, sr_debussy, "Debussy")
compute_waveform_features(redhot, sr_redhot, "Red Hot")
compute_waveform_features(duke, sr_duke, "Duke")
```



Waveform Factor of the Debussy signal: 1.4318



A waveform factor of 1.5885 indicates that the signal is spikier or has more irregularities compared to a perfectly smooth signal (such as a sine wave).

## ✓ AutoCorrelation:

Autocorrelation-based feature extraction is commonly used in time series analysis, signal processing, and pattern recognition to identify repetitive patterns or dependencies within a dataset. Autocorrelation measures how a signal correlates with a delayed version of itself. Given a time series  $x(t)$ , the autocorrelation function (ACF) at lag  $k$  is defined as:

$$R_k = \frac{1}{N} \sum_{t=1}^{N-k} x(t)x(t+k)$$

where:  $N$  is the total number of samples,

$x(t)$  is the value at time

$x(t+k)$  is the value at time

$t+k$  (lagged version of  $x$ )

```
# Load audio files
debussy, sr_debussy = librosa.load("/content/drive/My Drive/Multimedia/debussy.wav")
redhot, sr_redhot = librosa.load("/content/drive/My Drive/Multimedia/redhot.wav")
duke, sr_duke = librosa.load("/content/drive/My Drive/Multimedia/duke.wav")

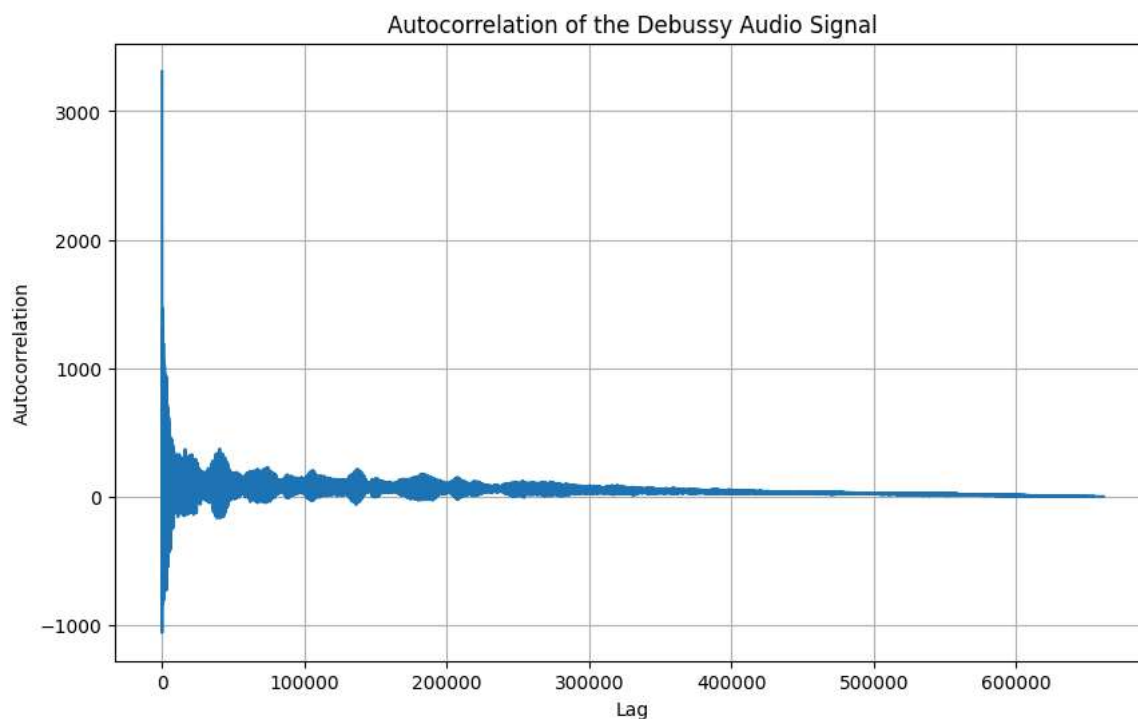
# Define autocorrelation function
def autocorrelation(x):
    result = np.correlate(x, x, mode='full')
    return result[result.size // 2:]

# Function to calculate and plot autocorrelation
def plot_autocorrelation(y, sr, title):
    auto_corr = autocorrelation(y)

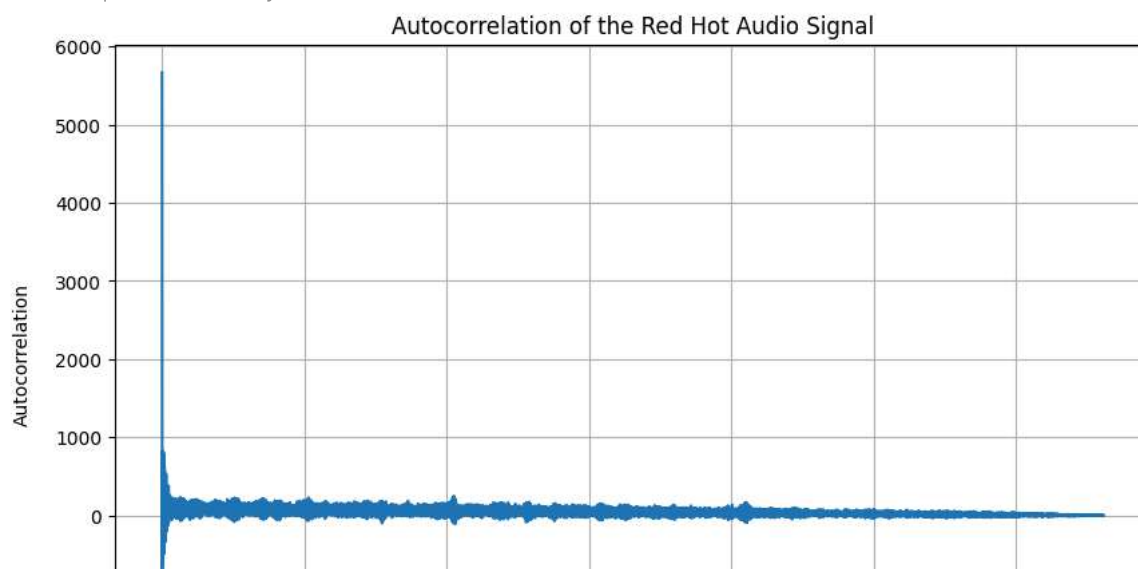
    # Plot the autocorrelation function
    plt.figure(figsize=(10, 6))
    plt.plot(auto_corr)
    plt.title(f'Autocorrelation of the {title} Audio Signal')
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.grid(True)
    plt.show()

    # Optionally, find the lag corresponding to the first peak (pitch detection)
    peaks = np.where(auto_corr > 0.5)[0]
    if len(peaks) > 1:
        pitch_lag = peaks[1] # First significant peak
        pitch = sr / pitch_lag # Pitch in Hz (fundamental frequency)
        print(f"Estimated pitch for {title}: {pitch} Hz")
    else:
        print(f"No significant pitch detected for {title}")

# Plot and detect pitch for all three files
plot_autocorrelation(debussy, sr_debussy, "Debussy")
plot_autocorrelation(redhot, sr_redhot, "Red Hot")
plot_autocorrelation(duke, sr_duke, "Duke")
```



Estimated pitch for Debussy: 22050.0 Hz



## ✓ Standard deviation

Standard deviation (SD) is a statistical measure that quantifies the amount of variation or dispersion in a dataset. It tells us how much individual data points deviate from the mean. A low standard deviation means the data points are close to the mean, while a high standard deviation indicates more spread.

For an entire population, standard deviation is calculated as:

$$\sigma = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

where:  $x(i)$  = individual data points

$\mu$  = mean of the population

$N$  = total number of data points

fit

```
# Load audio files
debussy, sr_debussy = librosa.load("/content/drive/My Drive/Multimedia/debussy.wav")
redhot, sr_redhot = librosa.load("/content/drive/My Drive/Multimedia/redhot.wav")
```

```
duke, sr_duke = librosa.load("/content/drive/My Drive/Multimedia/duke.wav")

# Define autocorrelation function
def autocorrelation(x):
    result = np.correlate(x, x, mode='full')
    return result[result.size // 2:]

# Function to calculate and plot autocorrelation with standard deviation
def plot_autocorrelation_and_std(y, sr, title):
    auto_corr = autocorrelation(y)

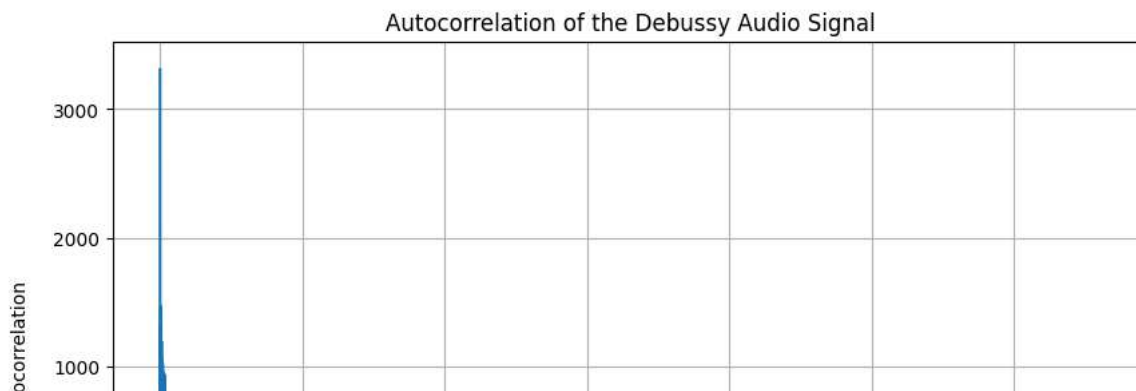
    # Calculate standard deviation
    std_dev = np.std(y)

    # Plot the autocorrelation function
    plt.figure(figsize=(10, 6))
    plt.plot(auto_corr)
    plt.title(f'Autocorrelation of the {title} Audio Signal')
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.grid(True)
    plt.show()

    # Optionally, find the lag corresponding to the first peak (pitch detection)
    peaks = np.where(auto_corr > 0.5)[0]
    if len(peaks) > 1:
        pitch_lag = peaks[1] # First significant peak
        pitch = sr / pitch_lag # Pitch in Hz (fundamental frequency)
        print(f"Estimated pitch for {title}: {pitch} Hz")
    else:
        print(f"No significant pitch detected for {title}")

    # Output the standard deviation
    print(f"Standard Deviation of {title} signal: {std_dev:.4f}")

# Plot and detect pitch for all three files
plot_autocorrelation_and_std(debussy, sr_debussy, "Debussy")
plot_autocorrelation_and_std(redhot, sr_redhot, "Red Hot")
plot_autocorrelation_and_std(duke, sr_duke, "Duke")
```



## ✓ Crest Factor

The Crest Factor (CF) is a measure of how extreme the peaks in a signal are compared to its overall energy. It is defined as the ratio of the peak value (

$$C_F = \frac{x_{\max}}{\text{RMS}}$$

where: x max = Maximum (peak) absolute value of the signal.

RMS = Root Mean Square value, which represents the signal's effective power.

Crest Factor is unitless and is always greater than or equal to 1.

Standard Deviation of Debussy signal: 0.0007

```
# Load audio files
debussy, sr_debussy = librosa.load("/content/drive/My Drive/Multimedia/debussy.wav")
redhot, sr_redhot = librosa.load("/content/drive/My Drive/Multimedia/redhot.wav")
duke, sr_duke = librosa.load("/content/drive/My Drive/Multimedia/duke.wav")
```

```
# Function to calculate Crest Factor
```

```
def crest_factor(x):
    x_max = np.max(np.abs(x)) # Maximum absolute value
    rms = np.sqrt(np.mean(x**2)) # Root Mean Square
    return x_max / rms
```

```
# Function to calculate and output standard deviation and Crest Factor
```

```
def CF(y, title):
    # Calculate standard deviation
    std_dev = np.std(y)

    # Calculate Crest Factor
    cf = crest_factor(y)

    # Output the standard deviation and Crest Factor
    print(f"Standard Deviation of {title} signal: {std_dev:.4f}")
    print(f"Crest Factor of {title} signal: {cf:.4f}")
```