In [2]:
```
# STEP 1: ACCEPT AUDIO INPUT
import os
import IPython.display as ipd

muffled_file = 'muffled-talking-6161.mp3' # source url: https://pixabay.com/so
sample_rate = 44100 # default sample rate for audio processing is 44.1 Hz

ipd.Audio(muffled_file, rate=sample_rate)
```

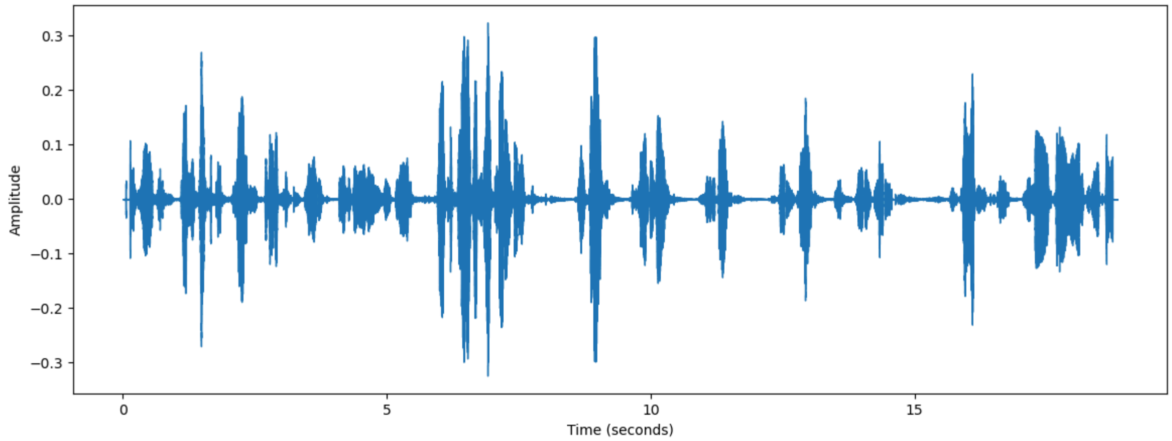
Out[2]:

0:03 / 0:18

In [3]:
```
%pip install librosa
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Requirement already satisfied: librosa in c:\users\vaishnavi pachava\appdata
\roaming\python\python311\site-packages (0.10.1)
Requirement already satisfied: audioread>=2.1.9 in c:\users\vaishnavi pachav
a\appdata\roaming\python\python311\site-packages (from librosa) (3.0.1)
Requirement already satisfied: numpy!=1.22.0,!=1.22.1,!=1.22.2,>=1.20.3 in
c:\programdata\anaconda3\lib\site-packages (from librosa) (1.24.3)
Requirement already satisfied: scipy>=1.2.0 in c:\programdata\anaconda3\lib
\site-packages (from librosa) (1.11.1)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\programdata\anacon
da3\lib\site-packages (from librosa) (1.3.0)
Requirement already satisfied: joblib>=0.14 in c:\programdata\anaconda3\lib
\site-packages (from librosa) (1.2.0)
Requirement already satisfied: decorator>=4.3.0 in c:\programdata\anaconda3
\lib\site-packages (from librosa) (5.1.1)
Requirement already satisfied: numba>=0.51.0 in c:\programdata\anaconda3\lib
\site-packages (from librosa) (0.57.1)
Requirement already satisfied: soundfile>=0.12.1 in c:\users\vaishnavi pacha
va\appdata\roaming\python\python311\site-packages (from librosa) (0.12.1)
Requirement already satisfied: pooch>=1.0 in c:\users\vaishnavi pachava\appd
ata\roaming\python\python311\site-packages (from librosa) (1.8.1)
Requirement already satisfied: soxr>=0.3.2 in c:\users\vaishnavi pachava\app
data\roaming\python\python311\site-packages (from librosa) (0.3.7)
Requirement already satisfied: typing-extensions>=4.1.1 in c:\programdata\an
aconda3\lib\site-packages (from librosa) (4.7.1)
Requirement already satisfied: lazy-loader>=0.1 in c:\programdata\anaconda3
\lib\site-packages (from librosa) (0.2)
Requirement already satisfied: msgpack>=1.0 in c:\programdata\anaconda3\lib
\site-packages (from librosa) (1.0.3)
Requirement already satisfied: llvmlite<0.41,>=0.40.0dev0 in c:\programdata
\anaconda3\lib\site-packages (from numba>=0.51.0->librosa) (0.40.0)
Requirement already satisfied: platformdirs>=2.5.0 in c:\programdata\anacond
a3\lib\site-packages (from pooch>=1.0->librosa) (3.10.0)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\l
ib\site-packages (from pooch>=1.0->librosa) (23.1)
Requirement already satisfied: requests>=2.19.0 in c:\programdata\anaconda3
\lib\site-packages (from pooch>=1.0->librosa) (2.31.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anacon
da3\lib\site-packages (from scikit-learn>=0.20.0->librosa) (2.2.0)
Requirement already satisfied: cffi>=1.0 in c:\programdata\anaconda3\lib\sit
e-packages (from soundfile>=0.12.1->librosa) (1.15.1)
Requirement already satisfied: pycparser in c:\programdata\anaconda3\lib\sit
e-packages (from cffi>=1.0->soundfile>=0.12.1->librosa) (2.21)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\programdata\an
aconda3\lib\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2.0.
4)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib
\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\programdata\anaconda
3\lib\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda
3\lib\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2023.7.22)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
# STEP 2: DISPLAY AUDIO AS A SIGNAL
import librosa # package for music & audio analysis
import numpy
from pylab import plot, figure, show, xlabel, ylabel

# PART A: WORKING WITH INPUT AUDIO
muffled_c4, m_sr = librosa.load(muffled_file)

# plot the signal's waveform
figure(figsize=(14, 5))
librosa.display.waveshow(muffled_c4, sr=m_sr)
xlabel('Time (seconds)')
ylabel('Amplitude')
show()
```




In [5]:
```
# PART B: TRYING SIMPLE MODIFICATIONS -> THEY AREN'T GOOD ENOUGH

# amplitude multiplier
amp_factor = 100
amp_signal = muffled_c4 * amp_factor
ipd.Audio(amp_signal, rate=m_sr)
```

Out[5]:

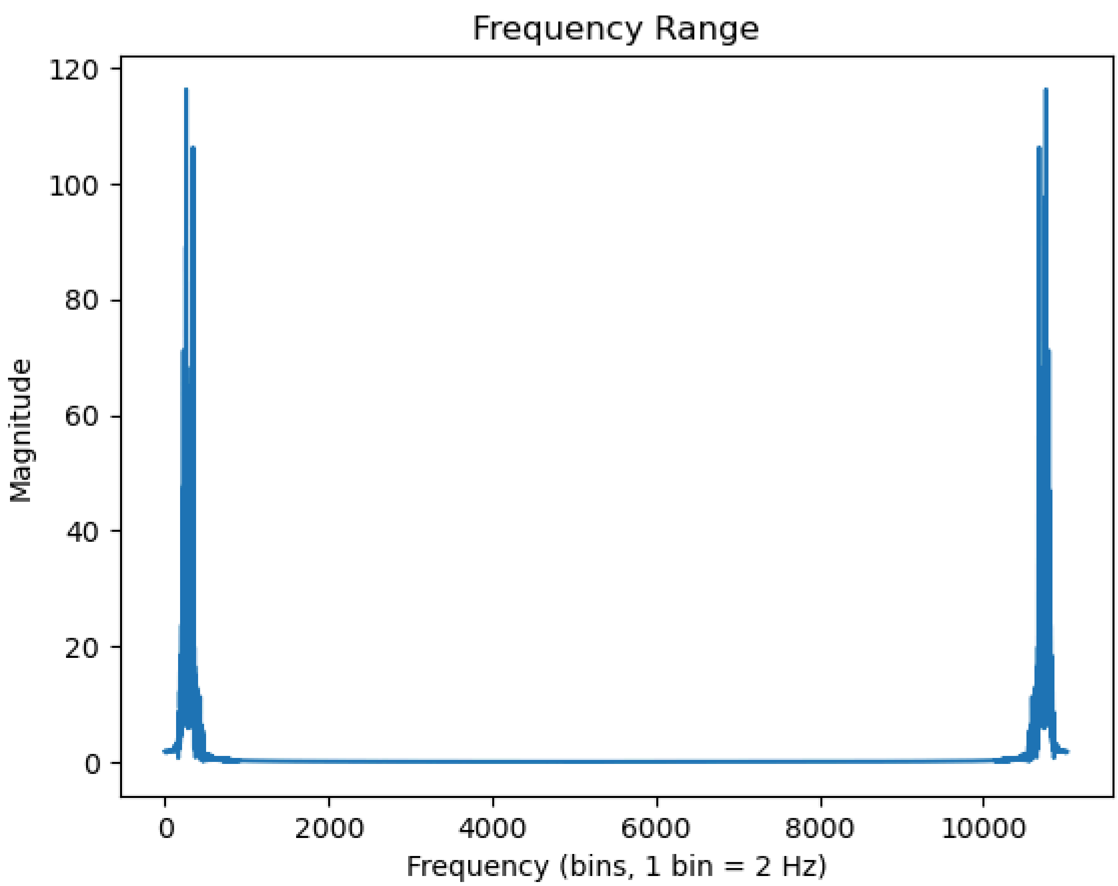
0:16 / 0:18

In [6]:
```
# slowing down audio
stretch_signal = librosa.effects.time_stretch(muffled_c4, rate=0.85)
resampled_stretch_signal = librosa.resample(stretch_signal, orig_sr=len(stretc
ipd.Audio(stretch_signal, rate=m_sr)
```

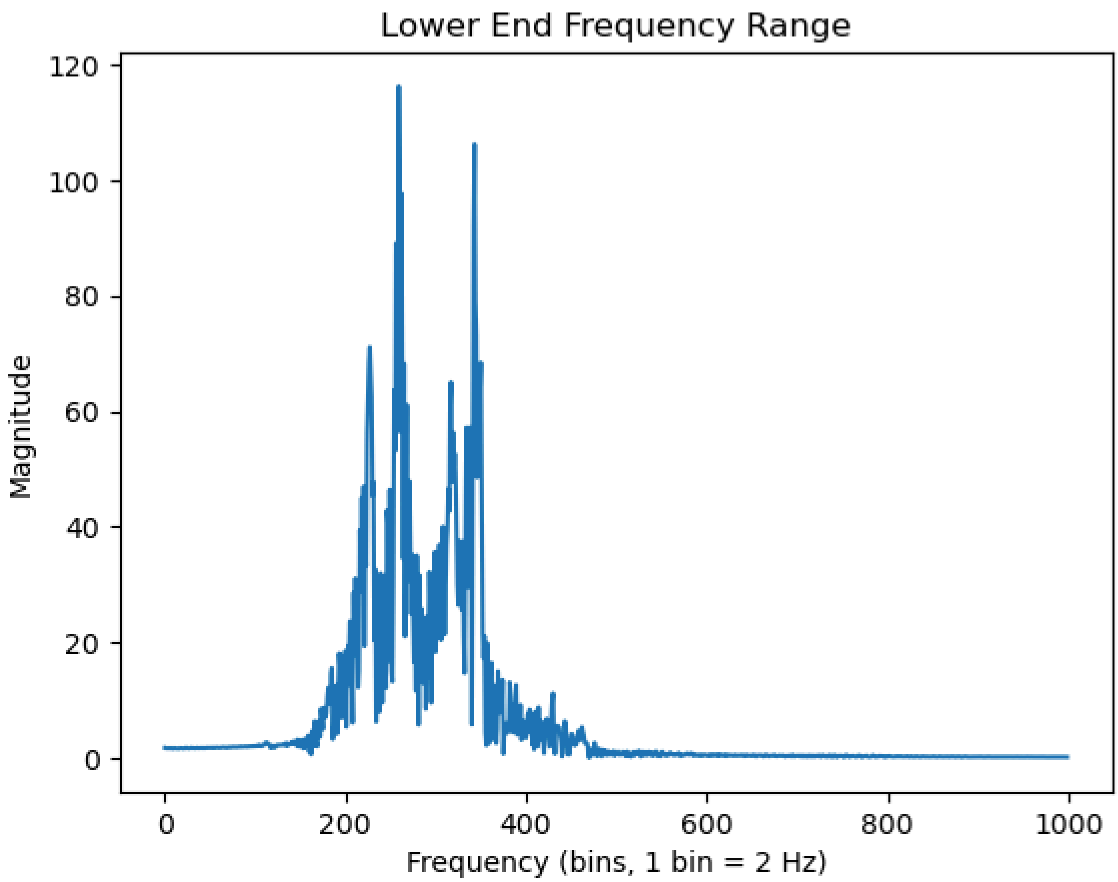
Out[6]:

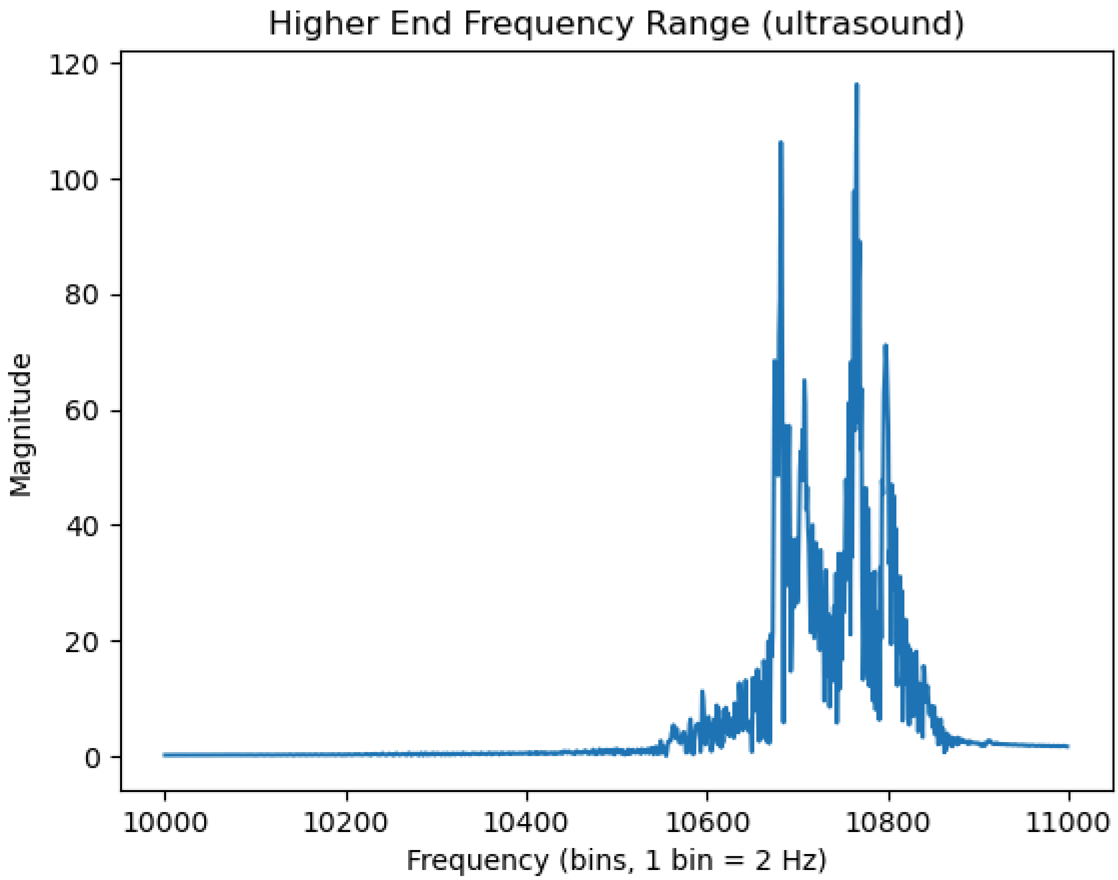
0:15 / 0:22

In [7]:
```
# STEP 3: EXTRACT FREQUENCIES VIA FOURIER TRANSFORM
from numpy import zeros
from cmath import exp, pi
from pylab import title

# define discrete fourier transform
def dft(y):
    N = len(y)
    c = zeros(N, complex)
    for k in range(N):
        for n in range(N):
            c[k] += y[n]*exp(-2j*pi*k*n/N)
    return c

# decide a subset for the transform -> where there is the most audio modulatio
startIdx = 6 * 22050 # find the sample # at 6s mark
endIdx = 11025 + startIdx # find the sample # at 6.5s mark

# plot transform of signal
transform = dft(muffled_c4[startIdx:endIdx])
plot(abs(transform))
xlabel('Frequency (bins, 1 bin = 2 Hz)')
ylabel('Magnitude')
title('Frequency Range')
show()
```



In [8]:
```
# zoom into interesting sections of the plot i.e. sections with the most chang

# lower end frequency range
lowerend_transform = abs(transform[0:1000])
plot(lowerend_transform)
xlabel('Frequency (bins, 1 bin = 2 Hz)')
ylabel('Magnitude')
title('Lower End Frequency Range')
show()

# higher end frequency range
from numpy import arange

higherend_transform = abs(transform[10000:11000])
plot(10000 + arange(1000), higherend_transform)
xlabel('Frequency (bins, 1 bin = 2 Hz)')
ylabel('Magnitude')
title('Higher End Frequency Range (ultrasound)')
show()
```




Higher End Frequency Range (ultrasound)
Magnitude
0
20
40
60
80
100
120
10000
10200
10400
10600
10800
11000
Frequency (bins, 1 bin = 2 Hz)

In [9]:
```
# STEP 4: DEVISE METHODS TO SELECTIVELY AMPLIFY COMPLEX SOUNDS
from numpy import sqrt

# PART A: FREQUENCY ANALYSIS VIA GOLDEN RATIO SEARCH FOR LOCAL & GLOBAL MAXIMA

# local maxima in an audio signal are all harmonic frequencies
def find_harmonics(signal, tolerance):
    golden_ratio = (sqrt(5) - 1) / 2

    a = 0
    b = len(signal) - 1

    freq_maxima = []

    while abs(b - a) > tolerance:
        c = b - (b - a) * golden_ratio
        d = a + (b - a) * golden_ratio

        fc = signal[int(c)]
        fd = signal[int(d)]

        if fc < fd:
            a = c
            freq_maxima.append(int(c))
        else:
            b = d
            freq_maxima.append(int(d))

    return freq_maxima

# the global maximum is the most common frequency, which is the fundamental fr
def find_fundamental(signal, local_maxima):
    max_val = 0
    max_freq = 0

    for m in local_maxima:
        if signal[m] > max_val:
            max_val = signal[m]
            max_freq = m

    return max_freq

# test on lower end frequency range
lowerend_harmonics = find_harmonics(lowerend_transform, 1e-05)
lowerend_fundamental = find_fundamental(lowerend_transform, lowerend_harmonics

print('Fundamental Frequency: ', lowerend_fundamental, ' Hz')
print(lowerend_transform[259])
print(lowerend_transform[260])
print(lowerend_transform[258])

print('Harmonics: ')
for h in lowerend_harmonics:
```

```
    print(h * 2, ' Hz')
```

```
Fundamental Frequency:  518  Hz
116.22455061032198
97.98088651687793
57.85533431339062
Harmonics:
1234  Hz
762  Hz
290  Hz
582  Hz
402  Hz
470  Hz
540  Hz
496  Hz
514  Hz
530  Hz
524  Hz
520  Hz
516  Hz
518  Hz
516  Hz
518  Hz
516  Hz
518  Hz
516  Hz
518  Hz
516  Hz
518  Hz
518  Hz
518  Hz
518  Hz
518  Hz
518  Hz
516  Hz
518  Hz
516  Hz
518  Hz
518  Hz
518  Hz
516  Hz
518  Hz
516  Hz
518  Hz
518  Hz
518  Hz
```

In [18]:
```
# PART B: FREQUENCY TRIMMING

# remove unnecessary frequencies
def freq_trimmer(signal, harmonics, limit):
    N = len(signal)
    c = zeros(N, complex)

    for n in range(N):
        if n in harmonics:
            continue
        if signal[n] < limit:
            c[n] = 0
        else:
            c[n] = signal[n]

    return c.real
```

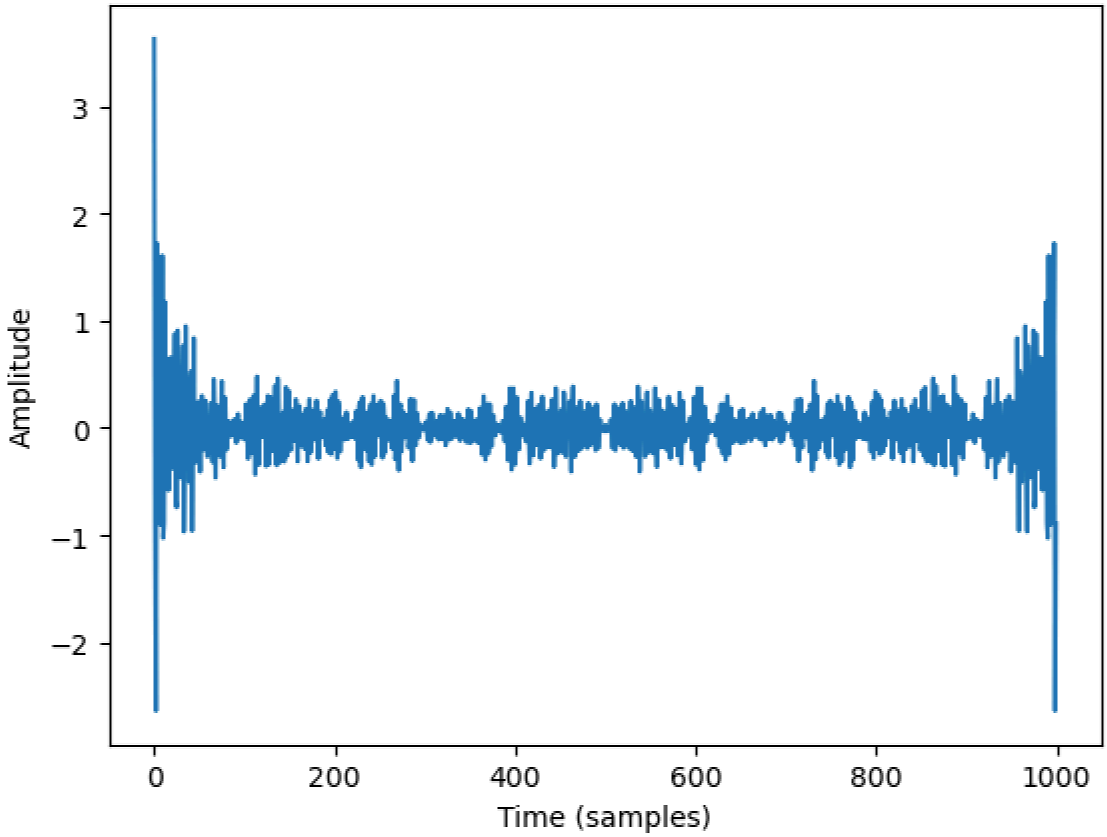
In [60]:
```
# PART C: PITCH SHIFTING VIA PHASE MANIPULATION
def pitch_push(signal, freq_shift, sr):
    # compute the phase shift based on the frequency shift
    phase_shift = 2 * pi * freq_shift / sr

    # normalize time axis that helps incorporate shift
    N = len(signal)
    t = arange(N)

    # apply phase shift to direct audio signal
    shifted_audio = zeros(N, complex)
    for n in range(N):
        shifted_audio[n] = signal[n] * exp(phase_shift*t[n]/sr)

    return shifted_audio.real
```

In [61]:
```
# PART D: NOISE REDUCTION BY SPECTRAL SUBTRACTION
from numpy import mean, where, max

def noise_reduce(signal, threshold):
    # calculate power spectrum from fourier transform
    power_spectrum = abs(signal) ** 2

    # apply noise condition to find general noise spectrum from power spectrum
    noise_spectrum = mean(power_spectrum[where(power_spectrum < threshold * ma

    # perform spectral subtraction
    clean_spectrum = power_spectrum - noise_spectrum
    plot(clean_spectrum)
    return clean_spectrum
```

In [12]:

```python
# STEP 5: PLAYBACK AMPLIFIED AUDIO & COMPARE WITH ORIGINAL

# define inverse discrete fourier transform to convert fourier coefficients ba
def inverse_dft(c):
    N = len(c)
    y = zeros(N, complex)
    for k in range(N):
        for n in range(N):
            y[k] += c[n]*exp(2j*pi*k*n/N)/N
    return y.real

# plot the modified audio signal
def playback_plot(signal):
    # plot the signal's waveform
    plot(signal)
    xlabel('Time (samples)')
    ylabel('Amplitude')
    show()
```

In [22]:
```
# PART A: TEST FREQUENCY TRIMMING
trimmed_signal = freq_trimmer(lowerend_transform, lowerend_harmonics, 30)

inv = inverse_dft(trimmed_signal)
playback_plot(inv)

ipd.Audio(inv, rate=m_sr)
```




Out[22]:

0:00 / 0:00

In [13]:
```
# PART B: TEST PITCH SHIFTING
pitchpushed_signal = pitch_push(muffled_c4, 400, m_sr)

ipd.Audio(pitchpushed_signal, rate=m_sr)
```

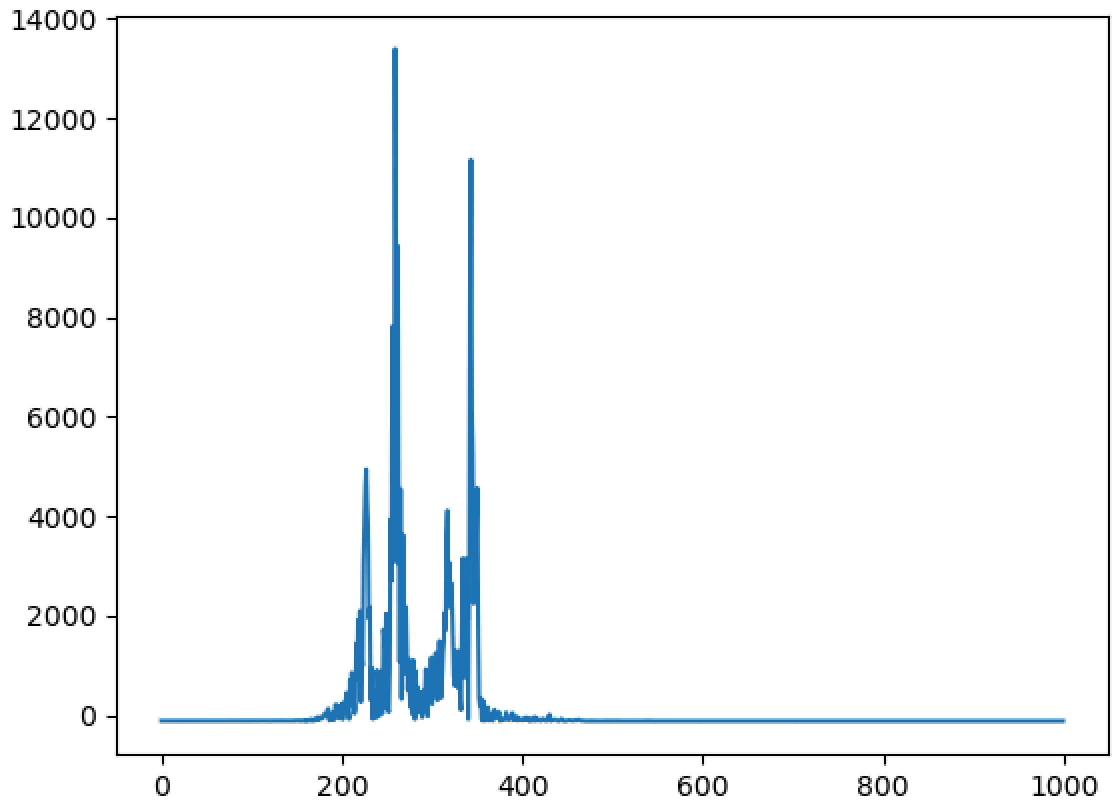
Out[13]:

0:15 / 0:18

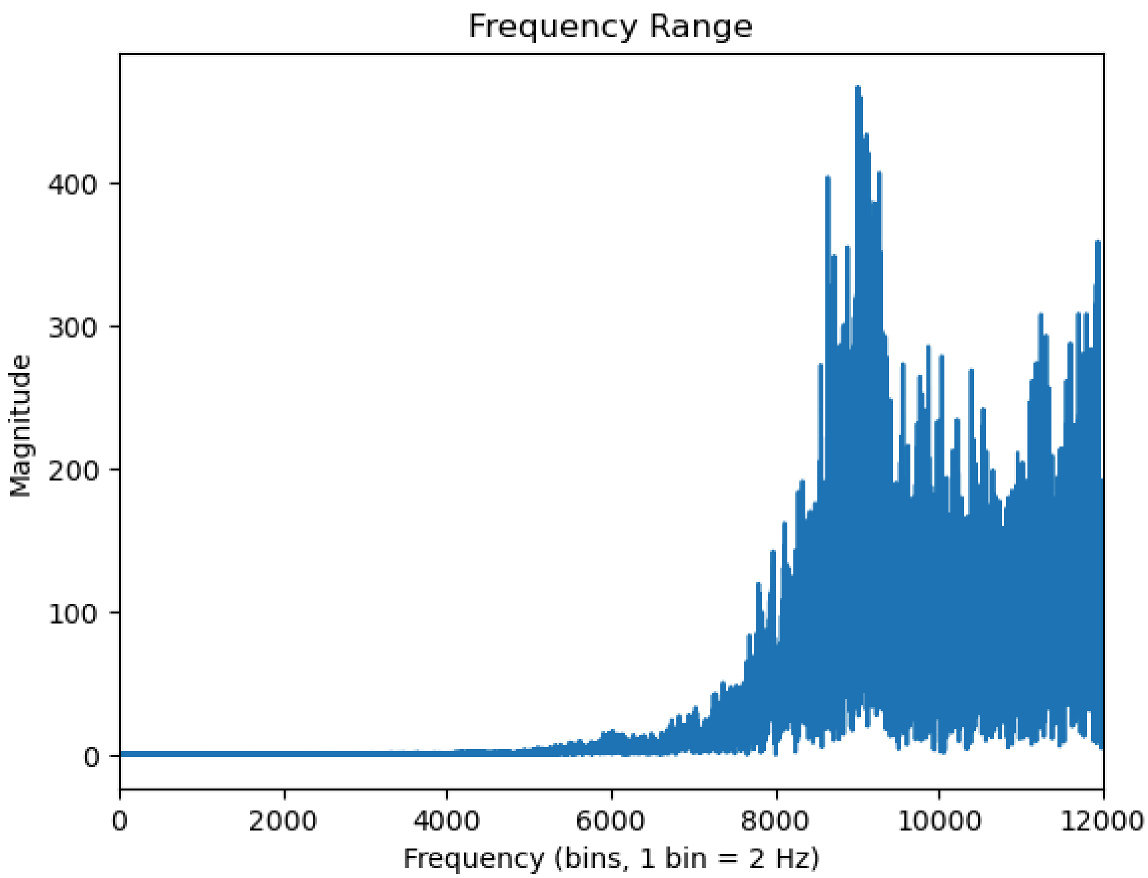
In [49]:
```
# PART C: TEST NOISE REDUCTION
reduced_signal = noise_reduce(lowerend_transform, 0.18)

inv_r = inverse_dft(reduced_signal)

ipd.Audio(inv_r, rate=m_sr)
```

Out[49]:

0:00 / 0:00



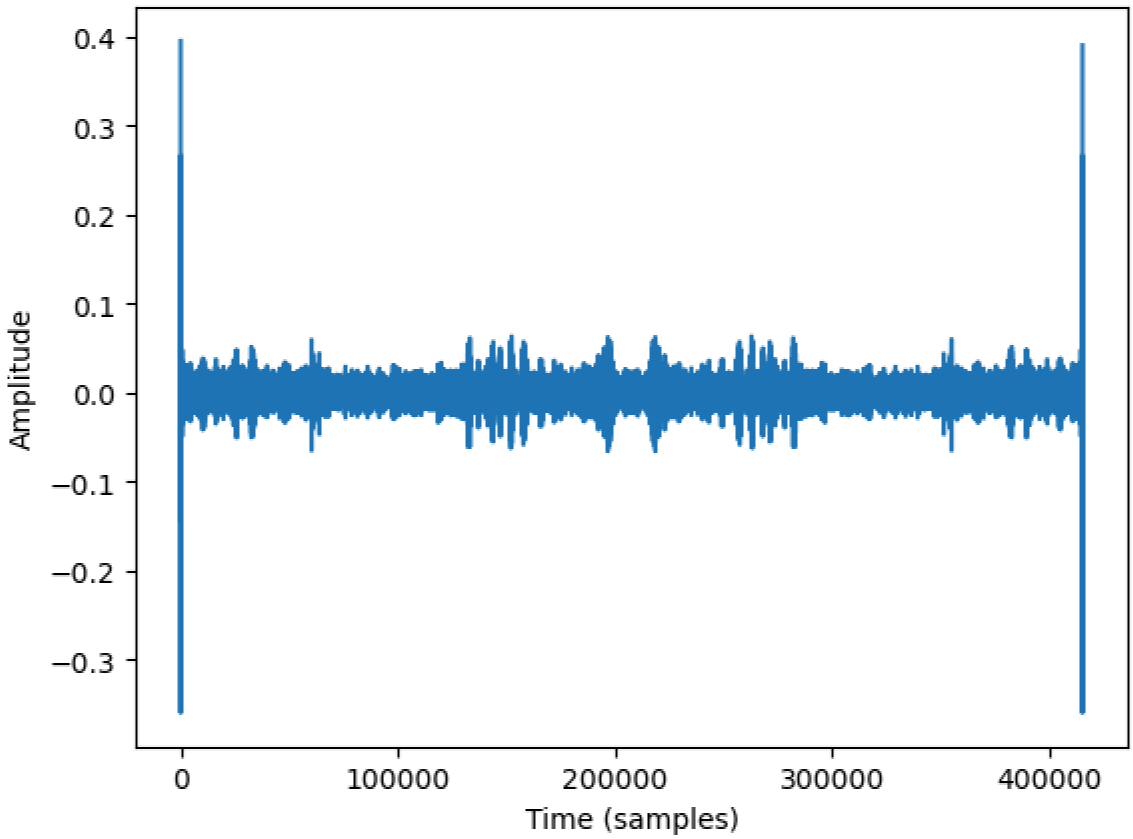
In [93]:
```
# PART D: COMBINE IT ALL -> LIMITED USE ON LARGE NUMBER OF SAMPLES
from numpy.fft import fft as fast_ft # use in-built fast fourier transform to
from numpy.fft import ifft as fast_ift # use in-built fast inverse fourier tr
from pylab import xlim

fulltr = fast_ft(muffled_c4)
plot(abs(fulltr))
xlim(0, 12000)
xlabel('Frequency (bins, 1 bin = 2 Hz)')
ylabel('Magnitude')
title('Frequency Range')
show()
```




In [63]:
```
# PART I: DO FREQUENCY ANALYSIS
full_harmonics = find_harmonics(fulltr, 1e-05)
```

In [96]:

```
# PART II: DO FREQUENCY TRIMMING
full_trim = freq_trimmer(fulltr, full_harmonics, 100)

inv_ft = fast_ift(full_trim)
playback_plot(inv_ft)

ipd.Audio(inv_ft, rate=m_sr)
```



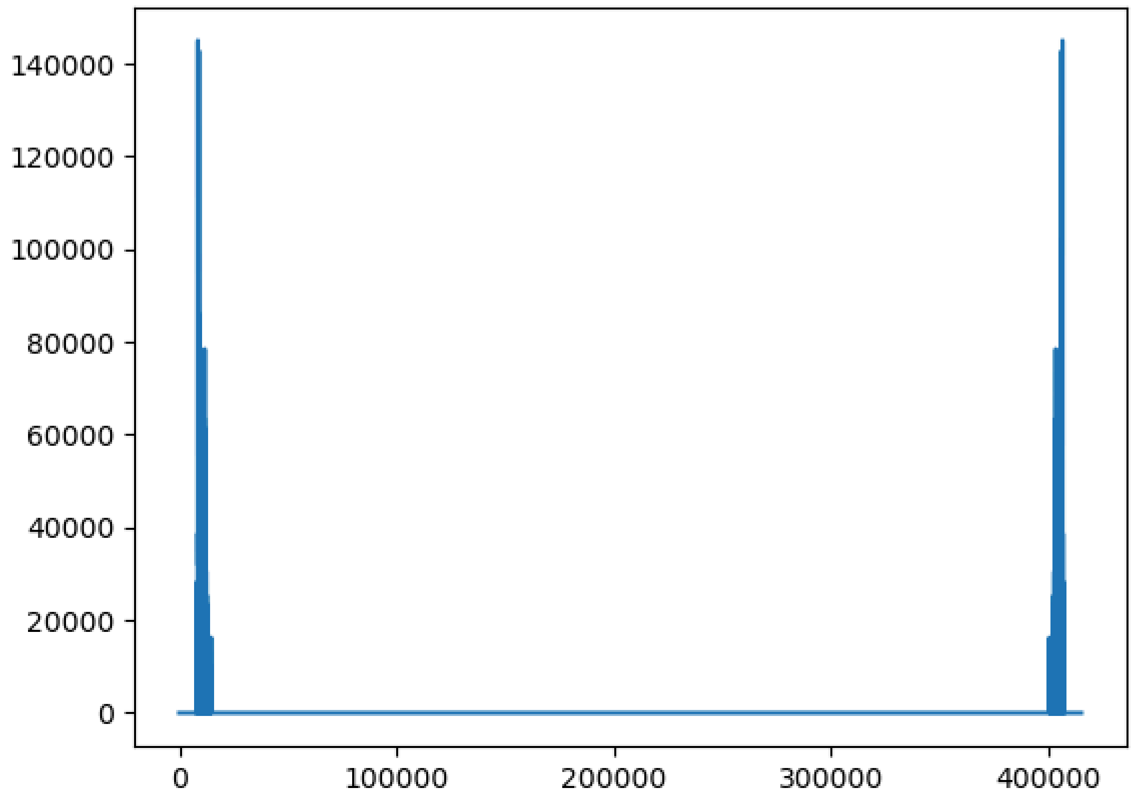

Out[96]:

0:13 / 0:18

In [115]:
```
# PART III: DO NOISE REDUCTION ON TRIMMED WAVE
full_reduce = noise_reduce(fast_ft(inv_ft), 1000)

inv_fr = fast_ift(full_reduce)

ipd.Audio(inv_fr, rate=m_sr)
```

Out[115]:

0:09 / 0:18




In [118]:
```
# PART IV: DO PITCH SHIFTING ON NOISE REDUCED WAVE
full_pitchpush_nr = pitch_push(inv_fr, 50, m_sr)

ipd.Audio(full_pitchpush_nr, rate=m_sr)
```

Out[118]:

0:16 / 0:18

In [ ]: