Operations On Basic Wav Signals

Dataset Description: (LJ001-0137.wav)

- Number of Channels: 1(Mono)
- Sample Rate: 22050 Hz
- Sample Width: 2 Bytes (16-Bit audio)
- Total Frames: 103837
- Duration: 4.71 seconds

Objective:

The objective of this experiment is to analyse speech signals and to extract information such as sample rate, number of channels, duration and frame count. Further to perform slicing, normalization, amplification and resampling and visualize their amplitude variations.

Code:

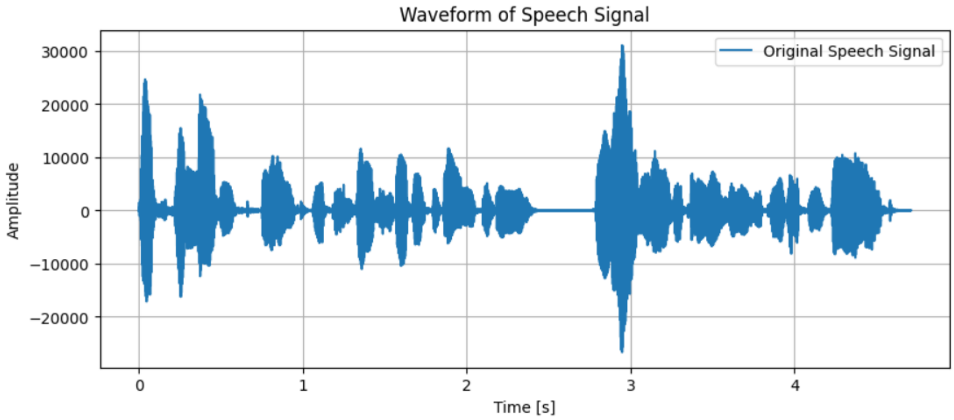
```
import wave
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import resample


def plot_speech_waveform(file_path):
    with wave.open(file_path, 'r') as wav_file:


        sample_rate = wav_file.getframerate()
        num_samples = wav_file.getnframes()
        duration = num_samples / sample_rate


        print(f"Sample Rate: {sample_rate} Hz")
        print(f"Number of Samples: {num_samples}")
        print(f"Duration: {duration:.2f} seconds")



        signal = wav_file.readframes(num_samples)
```

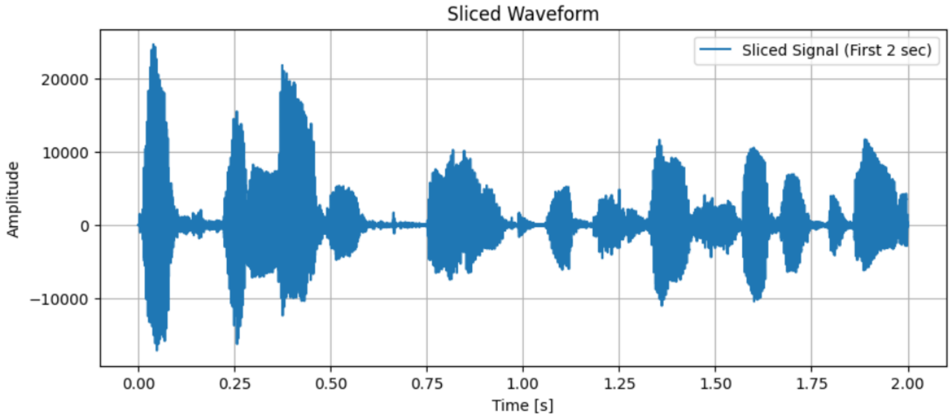
```
signal = np.frombuffer(signal, dtype=np.int16)

time_axis = np.linspace(0, duration, num=num_samples)

plt.figure(figsize=(10, 4))
plt.plot(time_axis, signal, label='Original Speech Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Waveform of Speech Signal')
plt.legend()
plt.grid()
plt.show()

# Slicing (First 2 seconds)
slice_samples = int(2 * sample_rate)
sliced_signal = signal[:slice_samples]
time_sliced = np.linspace(0, 2, num=slice_samples)

plt.figure(figsize=(10, 4))
plt.plot(time_sliced, sliced_signal, label='Sliced Signal (First 2 sec)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Sliced Waveform')
plt.legend()
plt.grid()
plt.show()


# Amplification and De-Amplification
amplified_signal = sliced_signal * 2
```

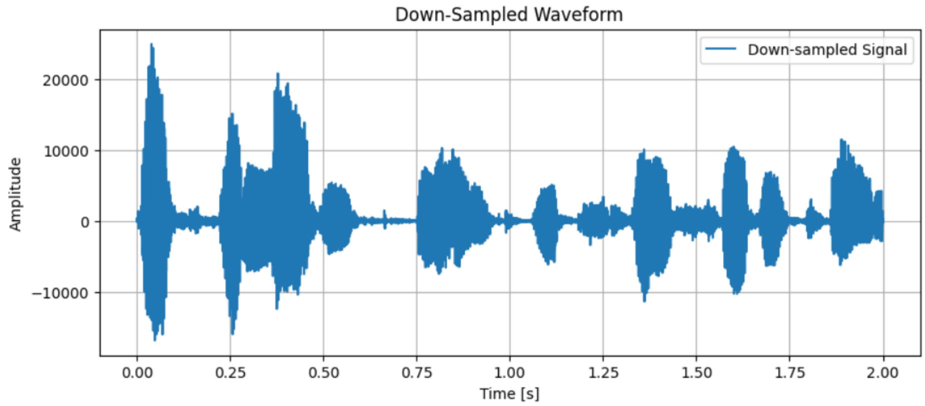
```
deamplified_signal = sliced_signal * 0.5

plt.figure(figsize=(10, 4))
plt.plot(time_sliced, amplified_signal, label='Amplified Signal')
plt.plot(time_sliced, deamplified_signal, label='De-Amplified Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Amplified & De-Amplified Signals')
plt.legend()
plt.grid()
plt.show()

# Up-sampling (Doubling Sample Rate)
upsampled_signal = resample(sliced_signal, len(sliced_signal) * 2)
time_upsampled = np.linspace(0, 2, num=len(upsampled_signal))

plt.figure(figsize=(10, 4))
plt.plot(time_upsampled, upsampled_signal, label='Up-sampled Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Up-Sampled Waveform')
plt.legend()
plt.grid()
plt.show()

# Down-sampling
downsampled_signal = resample(sliced_signal, len(sliced_signal) // 2)
time_downsampled = np.linspace(0, 2, num=len(downsampled_signal))

plt.figure(figsize=(10, 4))
```

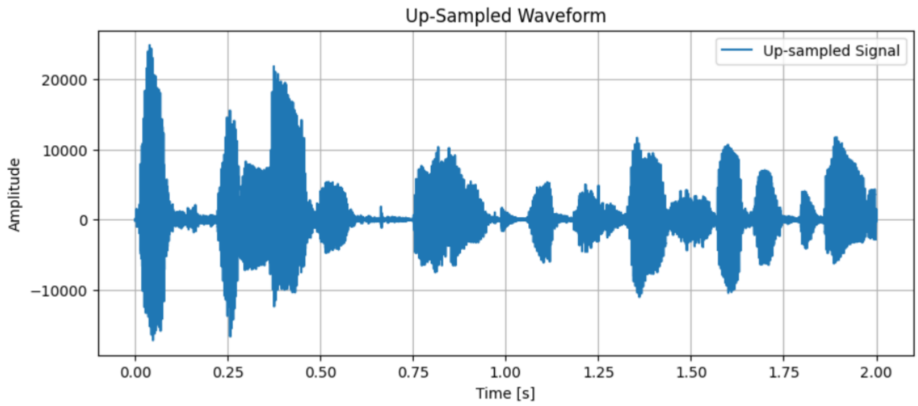
```
    plt.plot(time_downsampled, downsampled_signal, label='Down-sampled Signal')
    plt.xlabel('Time [s]')
    plt.ylabel('Amplitude')
    plt.title('Down-Sampled Waveform')
    plt.legend()
    plt.grid()
    plt.show()
plot_speech_waveform("/content/LJ001-0137.wav")
```

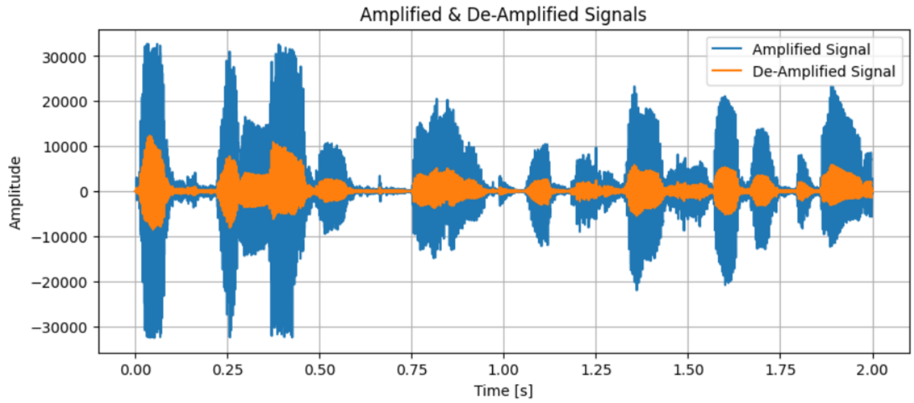
Output:











Conclusion:

The plots illustrate how different signal processing techniques impact the waveform. Normalization ensures consistency, amplification controls intensity, and resampling adjusts resolution. These transformations are crucial for optimizing speech signals for applications like speech recognition, audio compression, and real-time voice communication.