

Nama : Eric Arwido Damanik NIM : 122140157

Sistem Teknologi Multimedia

Soal 1. Analisis Suara Multi-Level

In [19]:

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

# Fungsi untuk Visualisasi Bentuk Gelombang dan Spektrogram
def visualize_audio(y, sr, title="Sinyal Audio"):
    plt.figure(figsize=(20, 12))

    # Bentuk Gelombang
    plt.subplot(2, 1, 1)
    librosa.display.waveform(y, sr=sr)
    plt.title(f"{title} - Bentuk Gelombang")
    plt.xlabel("Waktu (detik)")
    plt.ylabel("Amplitudo")

    # Spektrogram
    plt.subplot(2, 1, 2)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar(format='%.2f dB')
    plt.title(f"{title} - Spektrogram")
    plt.xlabel("Waktu (detik)")
    plt.ylabel("Frekuensi (Hz)")

    plt.tight_layout()
    plt.show()

# Memuat File Audio
audio_path = 'Data/rekaman_berita_multilevel.wav'
```

```
y, sr = librosa.load(audio_path, sr=None)

# Informasi tentang audio
print(f"Sample Rate Asli: {sr} Hz")

# Visualisasi Audio Asli
visualize_audio(y, sr, title="Audio Asli")

# Melakukan Resampling Audio ke 8000 Hz
sr_resampled = 8000
y_resampled = librosa.resample(y, orig_sr=sr, target_sr=sr_resampled)
print(f"Sample Rate Setelah Resampling: {sr_resampled} Hz")

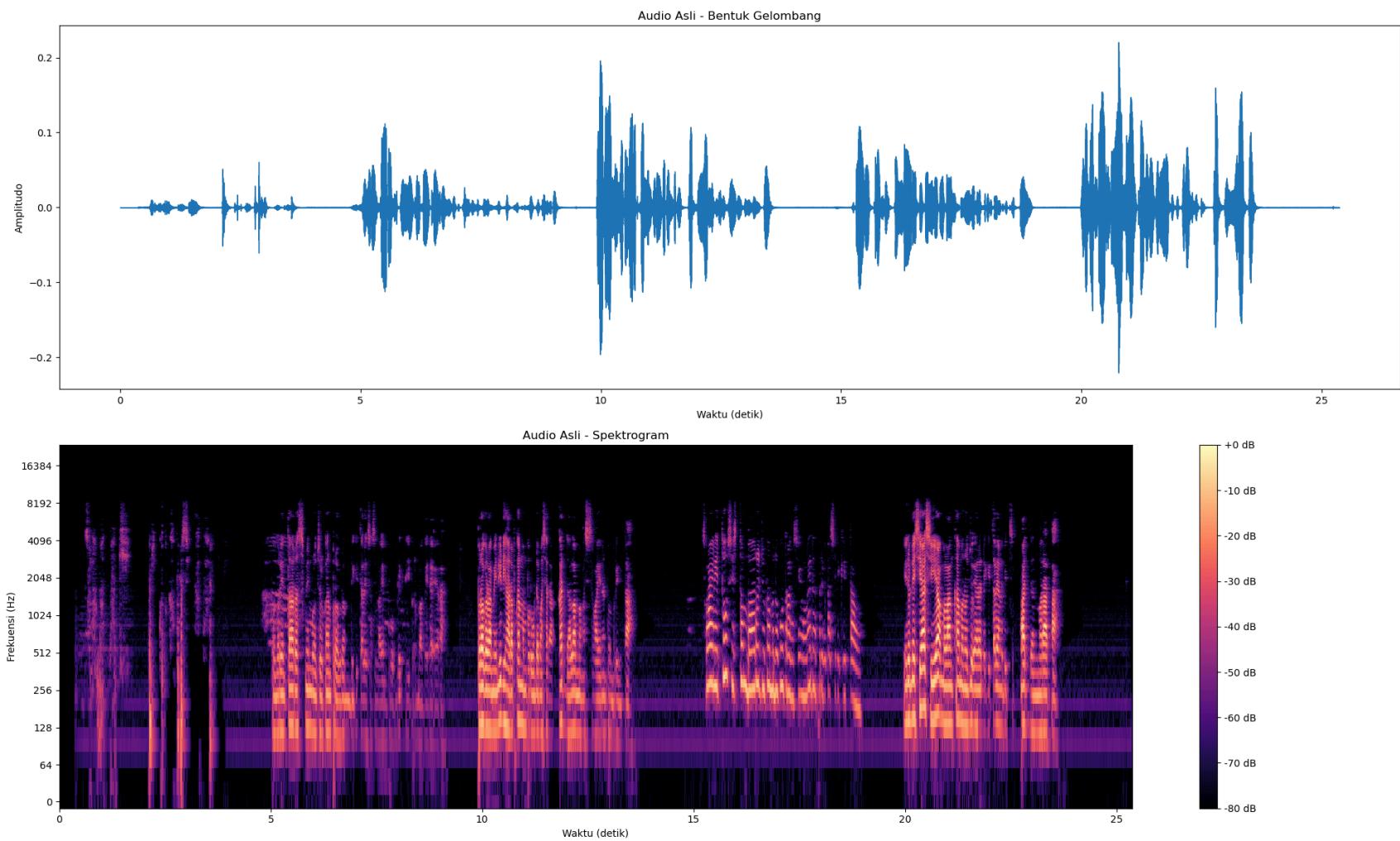
# Visualisasi Audio Setelah Resampling
visualize_audio(y_resampled, sr_resampled, title="Audio Setelah Resampling")

# Menyimpan Audio Setelah Resampling
resampled_audio_path = 'Data/rekaman_berita_multilevel_resampled.wav'
sf.write(resampled_audio_path, y_resampled, sr_resampled)

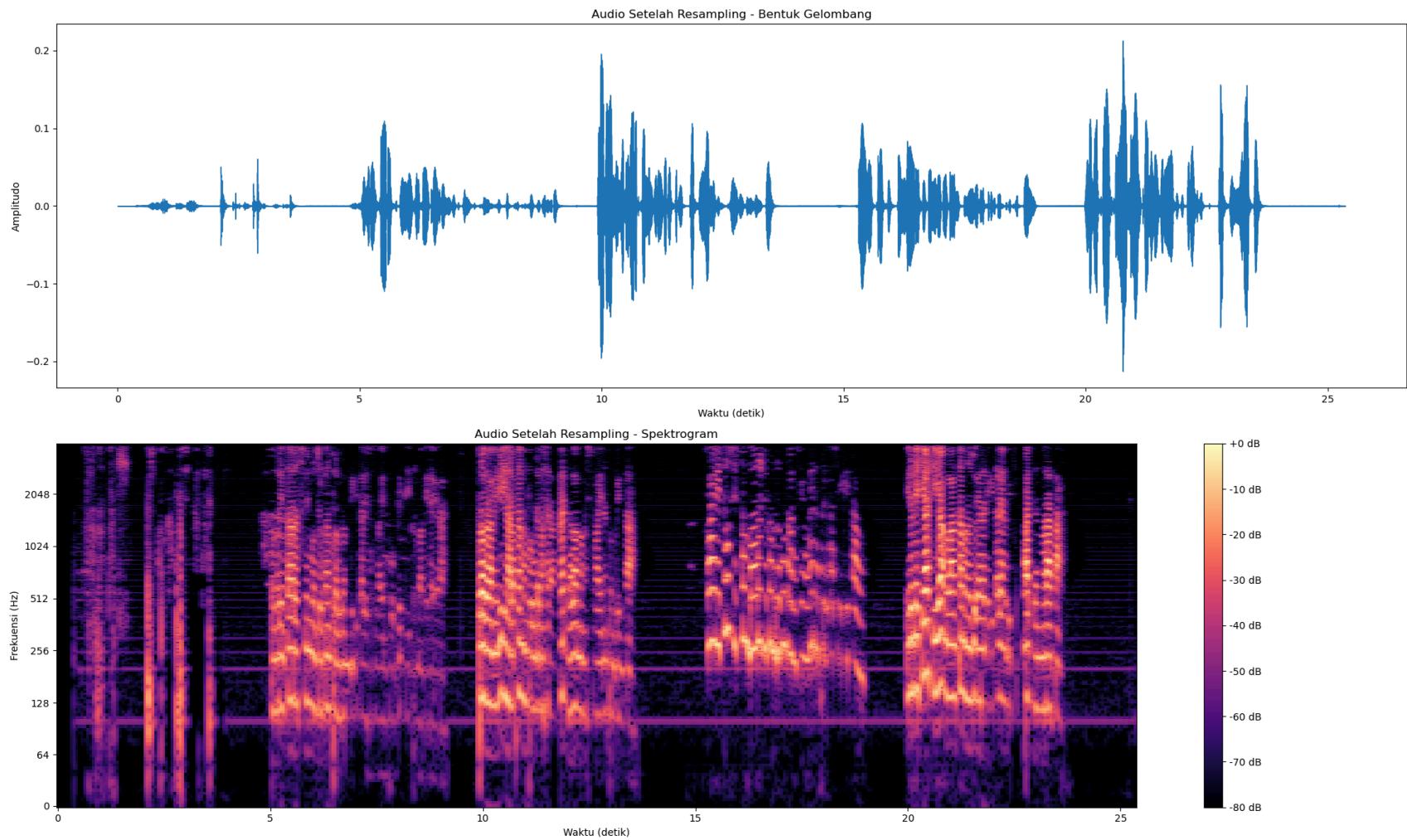
# Durasi audio
duration_original = librosa.get_duration(y=y, sr=sr)
duration_resampled = librosa.get_duration(y=y_resampled, sr=sr_resampled)
print(f"Durasi Audio Asli: {duration_original:.2f} detik")
print(f"Durasi Audio Setelah Resampling: {duration_resampled:.2f} detik")

# Memutar Audio Sebelum dan Sesudah Resampling
print('Audio Sebelum Resampling')
display(ipd.Audio(y, rate=sr))
print('Audio Setelah Resampling')
display(ipd.Audio(y_resampled, rate=sr_resampled))
```

Sample Rate Asli: 48000 Hz



Sample Rate Setelah Resampling: 8000 Hz



Durasi Audio Asli: 25.37 detik

Durasi Audio Setelah Resampling: 25.37 detik

Audio Sebelum Resampling

▶ 0:00 / 0:25 ⏸ 🔊 ⋮

Audio Setelah Resampling

▶ 0:00 / 0:25 ⏸ 🔊 ⋮

Penjelasan Hasil Visualisasi:

Waveform Pada Detik 5 Pertama, Amplitudo terlihat sangat kecil, 5 Detik Kemudian Makin Besar, 5 Detik Kemudian Semakin Besar, 5 Detik Kemudian terlihat mengecil karena suara cempreng, dan 5 Detik berikutnya terlihat lebih besar dari lainnya. Ini menunjukkan bahwa audio memiliki suara multi-level. Ini terjadi sebelum dan sesudah resampling, keliatan sama saja.

Spectrogram Warna kuning pada detik 5 pertama terlihat sedikit, 5 detik berikutnya semakin besar dan seterusnya, kecuali pada detik ke 15-20 yang terlihat sedikit pengurangan terangnya dibandingkan 5 detik sebelumnya. Terlihat bahwa audio ini memiliki suara multi-level.

Perbedaan Sebelum dan Sesudah Resampling: Durasi Audio : Durasi Audio Asli: 25.37 Durasi Audio Setelah Resampling: 25.37

Perbedaan terlihat jelas di spectrogram ketika di resampling ke 8000 Hz. Kualitas Audio Sebelum Resampling lebih bagus daripada setelah resampling. Terdengar kalo di resampling ke 8000 Hz akan sedikit menurunkan kualitas dari audio asli. Suara seperti teredam.

Soal 2. Noise Reduction dengan Filtering

```
In [20]: from scipy.signal import butter, lfilter

# Fungsi Filter (High-Pass Filter, Low-Pass Filter, Band-Pass Filter)

def high_pass_filter(data, cutoff, fs, order=5):
    # Menghitung frekuensi Nyquist (setengah dari sampling rate)
    nyquist = 0.5 * fs
    # Normalisasi frekuensi cutoff terhadap frekuensi Nyquist
    normal_cutoff = cutoff / nyquist
    # Membuat koefisien filter Butterworth tipe high-pass
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    # Menerapkan filter pada data audio
    y = lfilter(b, a, data)
    return y

def low_pass_filter(data, cutoff, fs, order=5):
    # Menghitung frekuensi Nyquist (setengah dari sampling rate)
    nyquist = 0.5 * fs
    # Normalisasi frekuensi cutoff terhadap frekuensi Nyquist
```

```
normal_cutoff = cutoff / nyquist
# Membuat koefisien filter Butterworth tipe low-pass
b, a = butter(order, normal_cutoff, btype='low', analog=False)
# Menerapkan filter pada data audio
y = lfilter(b, a, data)
return y

def band_pass_filter(data, lowcut, highcut, fs, order=5):
    # Menghitung frekuensi Nyquist (setengah dari sampling rate)
    nyquist = 0.5 * fs
    # Normalisasi frekuensi batas bawah terhadap frekuensi Nyquist
    low = lowcut / nyquist
    # Normalisasi frekuensi batas atas terhadap frekuensi Nyquist
    high = highcut / nyquist
    # Membuat koefisien filter Butterworth tipe band-pass
    b, a = butter(order, [low, high], btype='band')
    # Menerapkan filter pada data audio
    y = lfilter(b, a, data)
    return y

# Fungsi Buat Visualisasi Spektrogram
def plot_spectrogram(y, sr, title):
    # Mengkonversi STFT ke skala desibel
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
    # Membuat figure dengan ukuran 14x5 inci
    plt.figure(figsize=(14, 5))
    # Menampilkan spektrogram dengan skala logaritmik pada sumbu Y
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
    # Menambahkan colorbar dengan format desibel
    plt.colorbar(format='%+2.0f dB')
    # Menambahkan judul pada plot
    plt.title(title)
    # Menampilkan plot
    plt.show()

# Memuat File Audio dengan Noise
audio_path_noise = 'Data/rekaman_noise.wav'
y_noise, sr_noise = librosa.load(audio_path_noise, sr=None)

# Informasi Audio
print(f'Sample Rate: {sr_noise} Hz')
print(f'Durasi Audio: {librosa.get_duration(y=y_noise, sr=sr_noise)} detik')
```

```
# Spektrogram Asli
plot_spectrogram(y_noise, sr_noise, 'Spectrogram Rekaman dengan Noise')

# Menerapkan Filter untuk Mengurangi Noise
cutoff_freq = 500

# Menerapkan High-Pass Filter (menghilangkan frekuensi rendah di bawah 500 Hz)
y_high_passed = high_pass_filter(y_noise, cutoff_freq, sr_noise)
plot_spectrogram(y_high_passed, sr_noise, 'Spectrogram Setelah High-Pass Filter')
sf.write('Data/rekaman_high_passed.wav', y_high_passed, sr_noise)
print('Audio setelah High-Pass Filter disimpan ke: Data/rekaman_high_passed.wav')

# Menerapkan Low-Pass Filter (menghilangkan frekuensi tinggi di atas 500 Hz)
y_low_passed = low_pass_filter(y_noise, cutoff_freq, sr_noise)
plot_spectrogram(y_low_passed, sr_noise, 'Spectrogram Setelah Low-Pass Filter')
sf.write('Data/rekaman_low_passed.wav', y_low_passed, sr_noise)
print('Audio setelah Low-Pass Filter disimpan ke: Data/rekaman_low_passed.wav')

# Menerapkan Band-Pass Filter (hanya mempertahankan frekuensi antara 500 Hz - 2500 Hz)
lowcut = 500
highcut = 2500
y_band_passed = band_pass_filter(y_noise, lowcut, highcut, sr_noise)
plot_spectrogram(y_band_passed, sr_noise, 'Spectrogram Setelah Band-Pass Filter')
sf.write('Data/rekaman_band_passed.wav', y_band_passed, sr_noise)
print('Audio setelah Band-Pass Filter disimpan ke: Data/rekaman_band_passed.wav')

# Memutar Audio untuk Perbandingan
print('\n==== Perbandingan Audio ====')
print('Audio Asli dengan Noise:')
display(ipd.Audio(y_noise, rate=sr_noise))

print('Audio Setelah High-Pass Filter (frekuensi rendah dihilangkan):')
display(ipd.Audio(y_high_passed, rate=sr_noise))

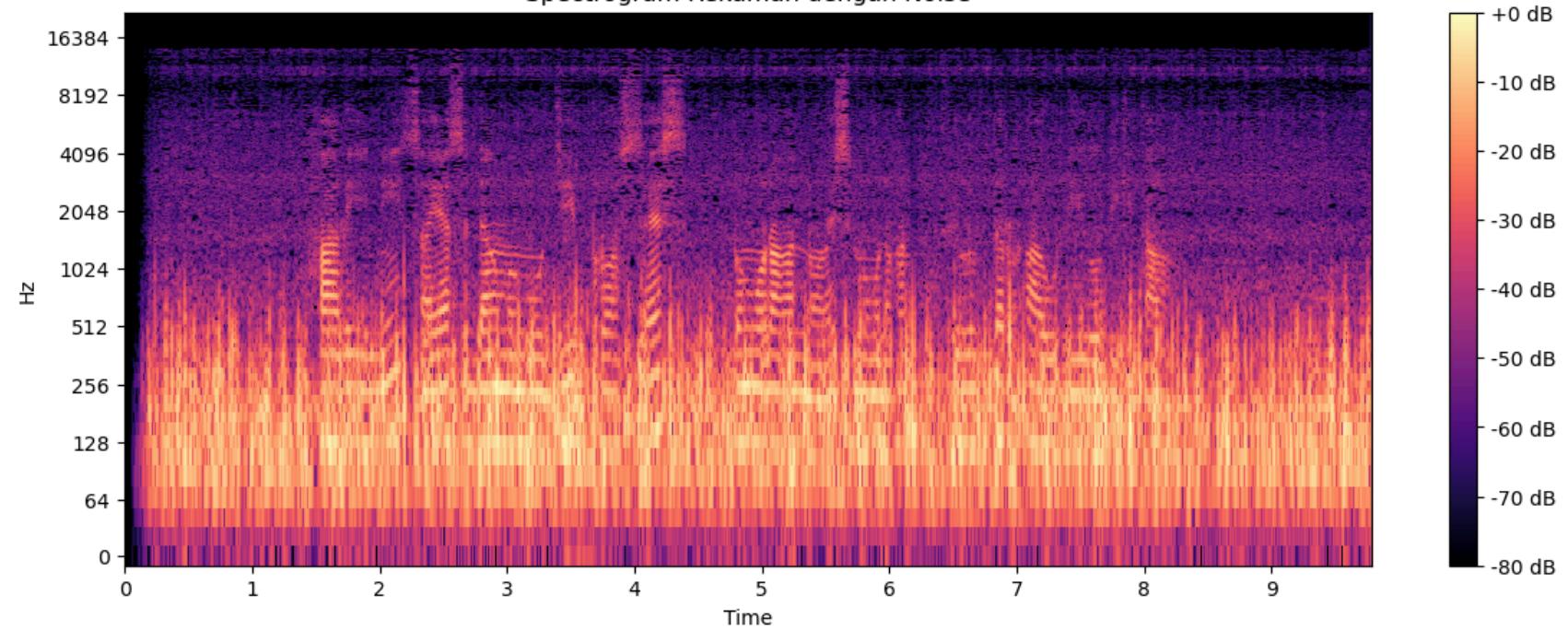
print('Audio Setelah Low-Pass Filter (frekuensi tinggi dihilangkan):')
display(ipd.Audio(y_low_passed, rate=sr_noise))

print('Audio Setelah Band-Pass Filter (hanya frekuensi 500-2500 Hz):')
display(ipd.Audio(y_band_passed, rate=sr_noise))
```

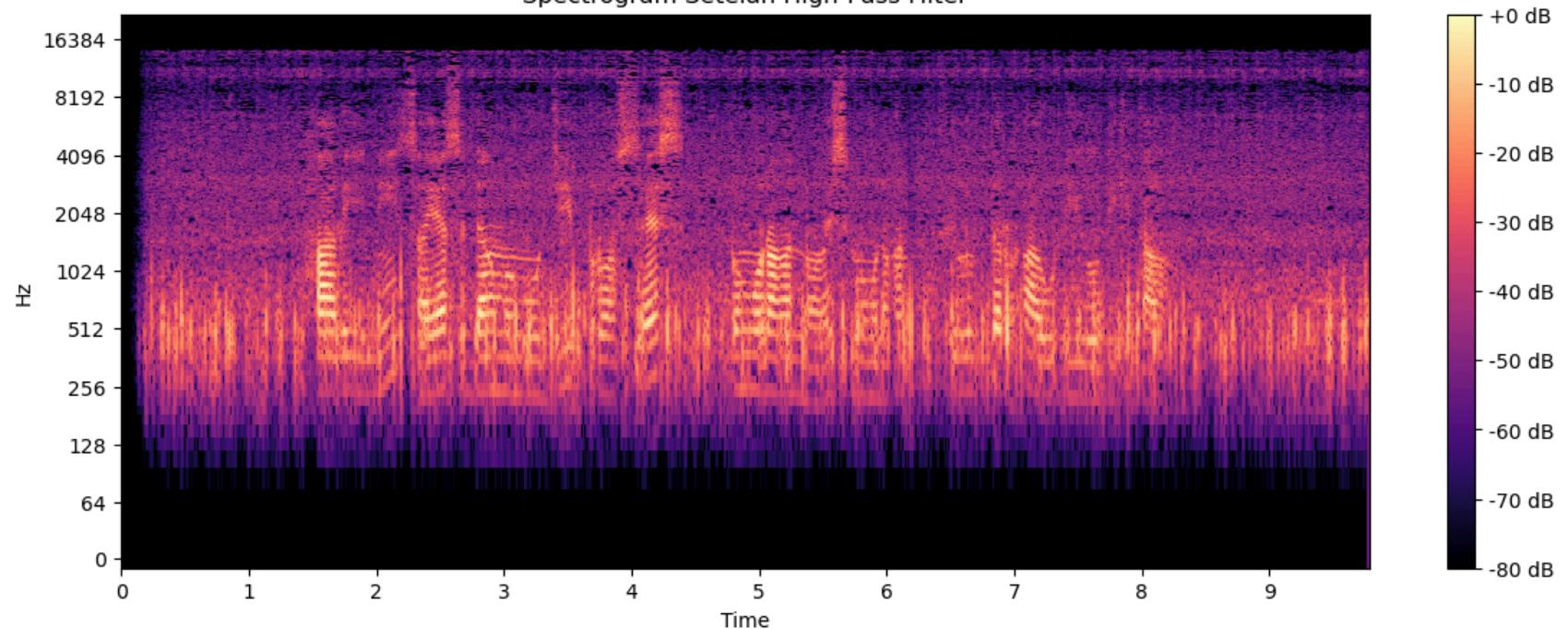
Sample Rate: 44100 Hz

Durasi Audio: 9.775600907029478 detik

Spectrogram Rekaman dengan Noise

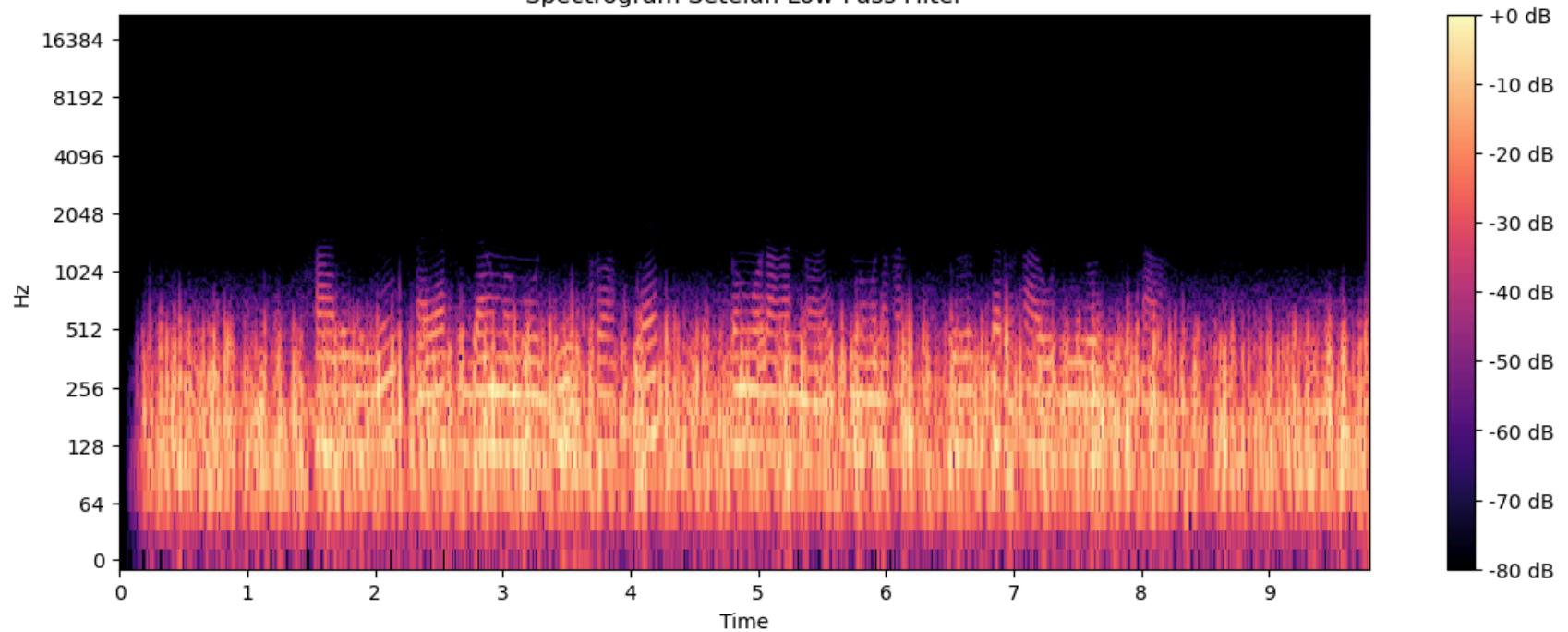


Spectrogram Setelah High-Pass Filter

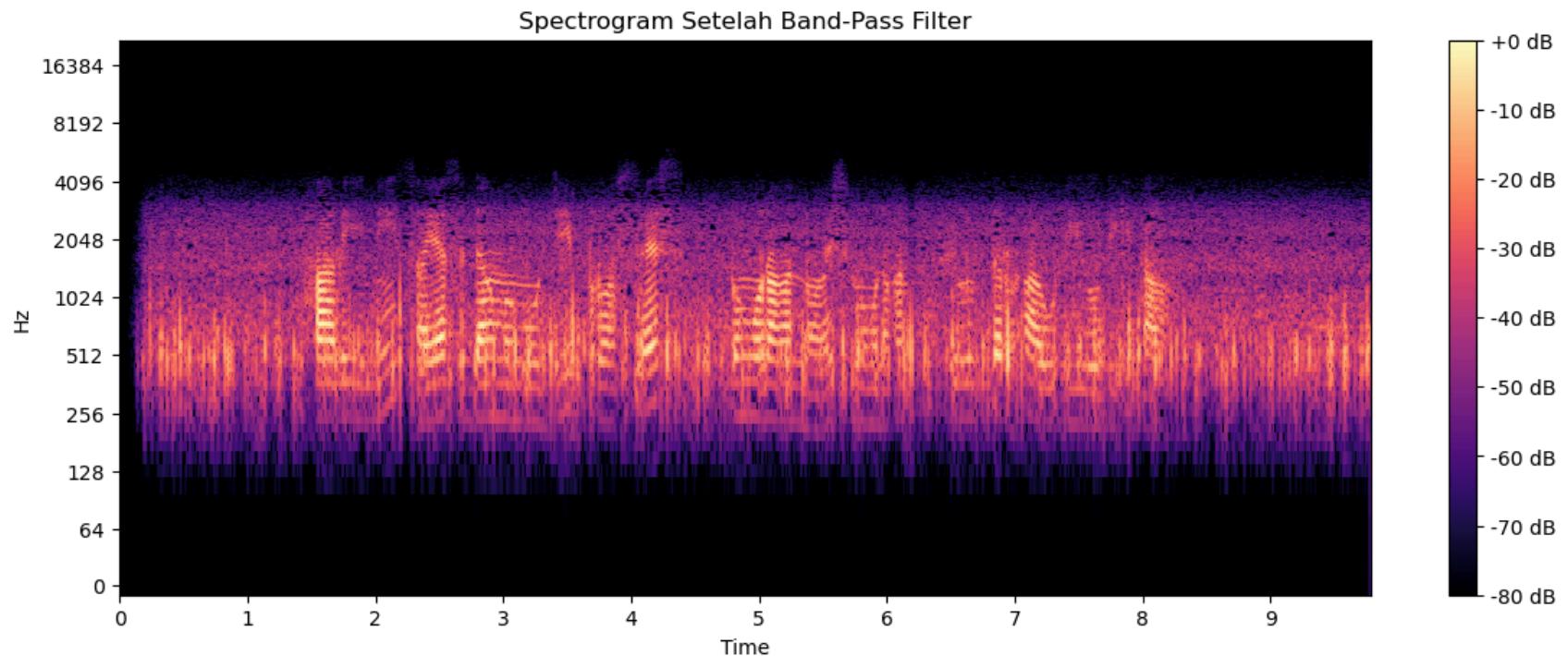


Audio setelah High-Pass Filter disimpan ke: Data/rekaman_high_passed.wav

Spectrogram Setelah Low-Pass Filter



Audio setelah Low-Pass Filter disimpan ke: Data/rekaman_low_passed.wav



Audio setelah Band-Pass Filter disimpan ke: Data/rekaman_band_passed.wav

==== Perbandingan Audio ===

Audio Asli dengan Noise:

▶ 0:00 / 0:09 ━━━━ 🔊 ⋮

Audio Setelah High-Pass Filter (frekuensi rendah dihilangkan):

▶ 0:00 / 0:09 ━━━━ 🔊 ⋮

Audio Setelah Low-Pass Filter (frekuensi tinggi dihilangkan):

▶ 0:00 / 0:09 ━━━━ 🔊 ⋮

Audio Setelah Band-Pass Filter (hanya frekuensi 500-2500 Hz):



Analisis

Noise yang ada adalah noise Kipas Angin yang stabil. Pada spectrogram terlihat pada frekuensi rendah.

Filter yang lebih cocok untuk audio ini: Filter yang paling efektif adalah **high-pass filter** karena noise kipas angin berada di frekuensi rendah, sementara suara vokal berada di frekuensi yang lebih tinggi. Filter ini berhasil memotong frekuensi rendah tanpa terlalu banyak mengganggu suara vokal.

High-Pass Filter: Menghilangkan noise kipas angin dengan baik, suara vokal tetap jelas.

Low-Pass Filter: Justru mempertahankan noise kipas angin dan menghilangkan detail vokal di frekuensi tinggi. Tidak cocok untuk kasus ini.

Band-Pass Filter: Mempertahankan frekuensi tertentu (500-2500 Hz), cukup efektif tapi mengurangi sebagian kualitas vokal yang ada di luar rentang tersebut.

Jadi kesimpulannya, **High-Pass Filter adalah yang terbaik** untuk mengurangi noise kipas angin pada rekaman ini.

Soal 3. Pitch Shifting dan Audio Manipulation

In [21]:

```
# Fungsi untuk visualisasi waveform dan spectrogram
def visualize_audio(y, sr, title="Audio Signal"):
    plt.figure(figsize=(20, 12))

    # Waveform
    plt.subplot(2, 1, 1)
    librosa.display.waveshow(y, sr=sr)
    plt.title(f"{title} - Waveform")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")

    # Spectrogram
    plt.subplot(2, 1, 2)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
```

```
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
plt.colorbar(format='%.2f dB')
plt.title(f"{title} - Spectrogram")
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")

plt.tight_layout()
plt.show()

# Load Audio File
audio_path = 'Data/rekaman_berita_multilevel.wav'
y, sr = librosa.load(audio_path, sr=None)

# Visualisasi Audio Asli
visualize_audio(y, sr, title="Original Audio")

# Pitch Shifting (+7 and +12 semitones)
y_pitch_7 = librosa.effects.pitch_shift(y=y, sr=sr, n_steps=7)
y_pitch_12 = librosa.effects.pitch_shift(y=y, sr=sr, n_steps=12)

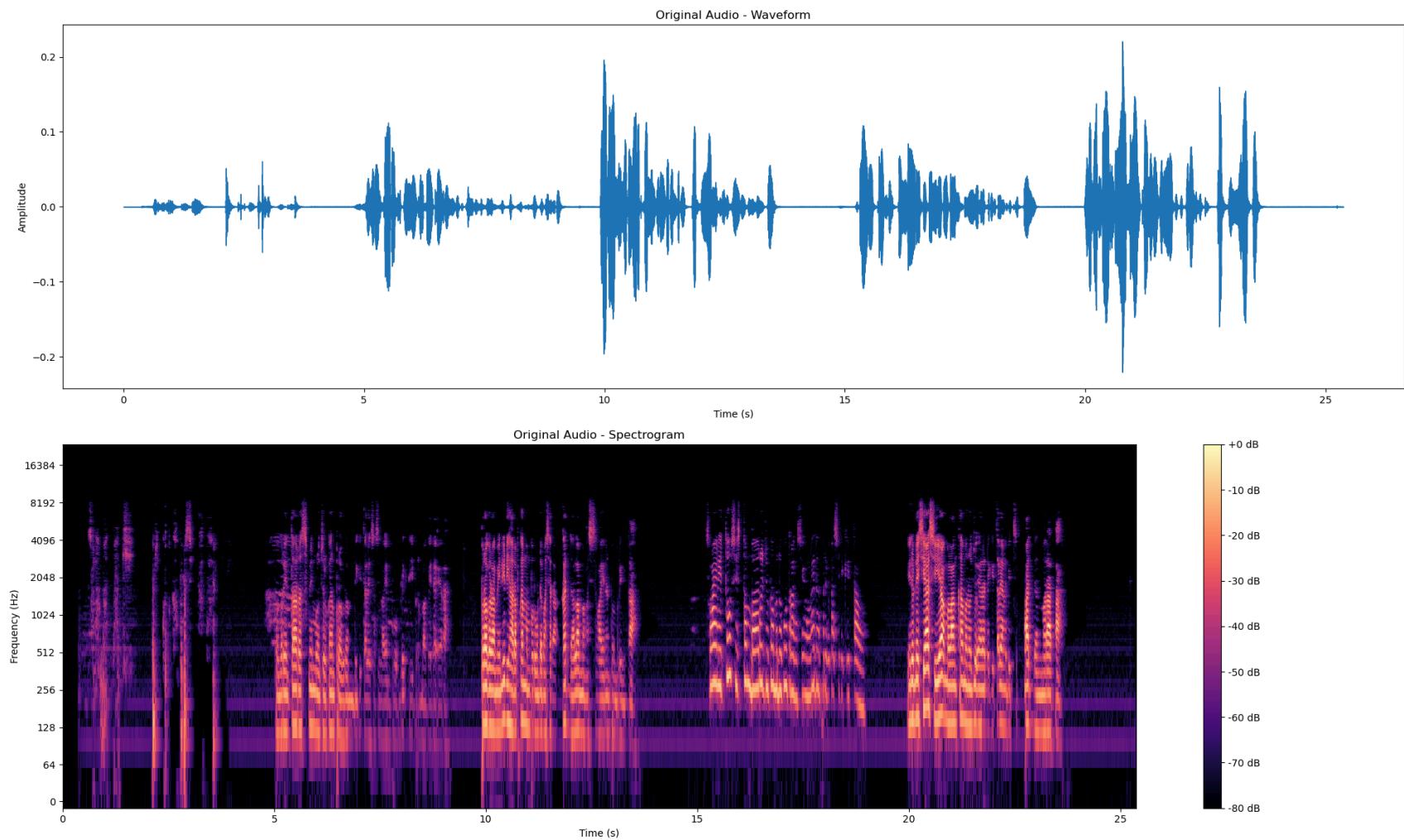
# Visualisasi Audio Setelah Pitch Shifting
visualize_audio(y_pitch_7, sr, title="Pitch Shifted +7 Semitones")
visualize_audio(y_pitch_12, sr, title="Pitch Shifted +12 Semitones")

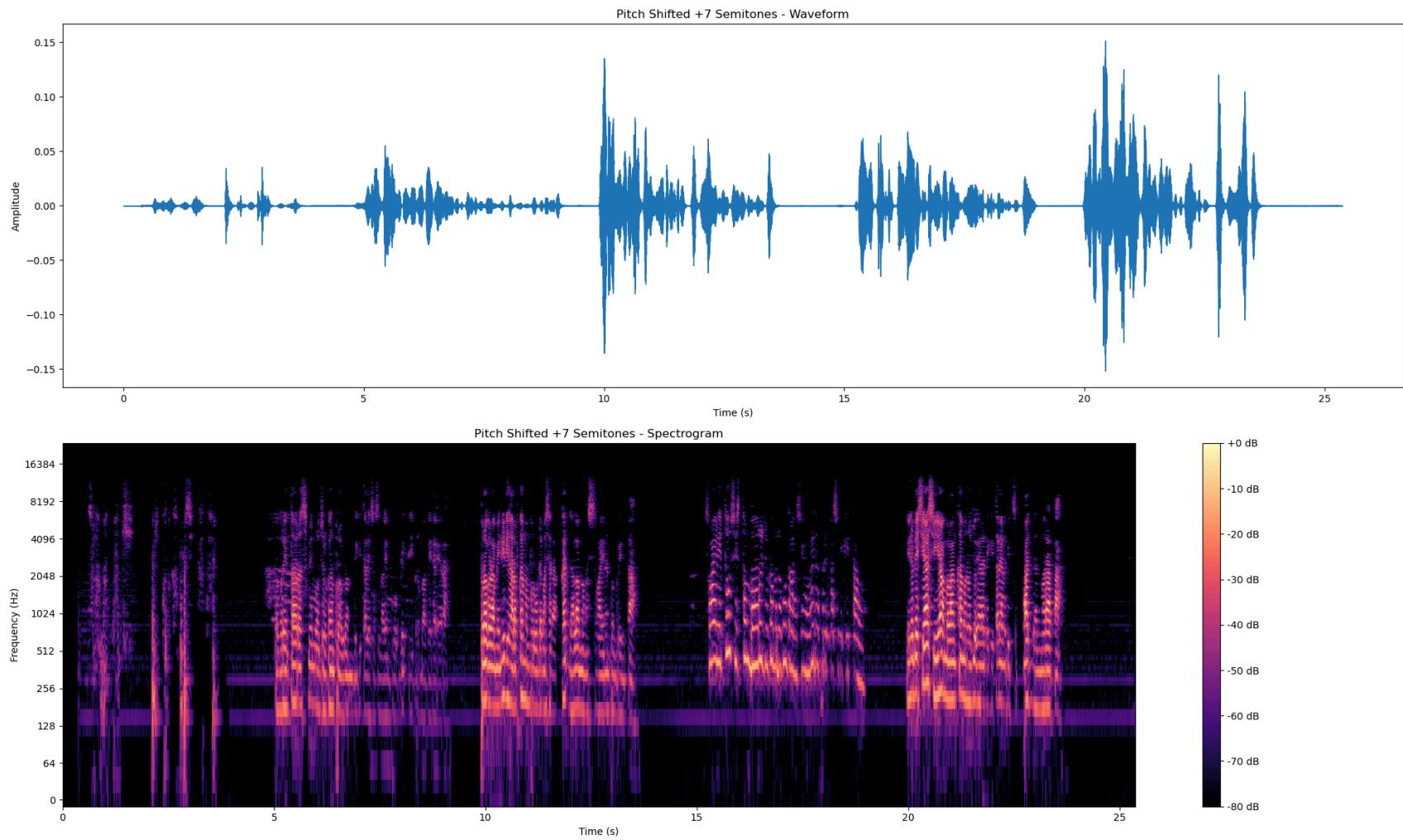
# Simpan Audio Setelah Pitch Shifting
sf.write('Data/rekaman_pitch_shifted_7.wav', y_pitch_7, sr)
sf.write('Data/rekaman_pitch_shifted_12.wav', y_pitch_12, sr)

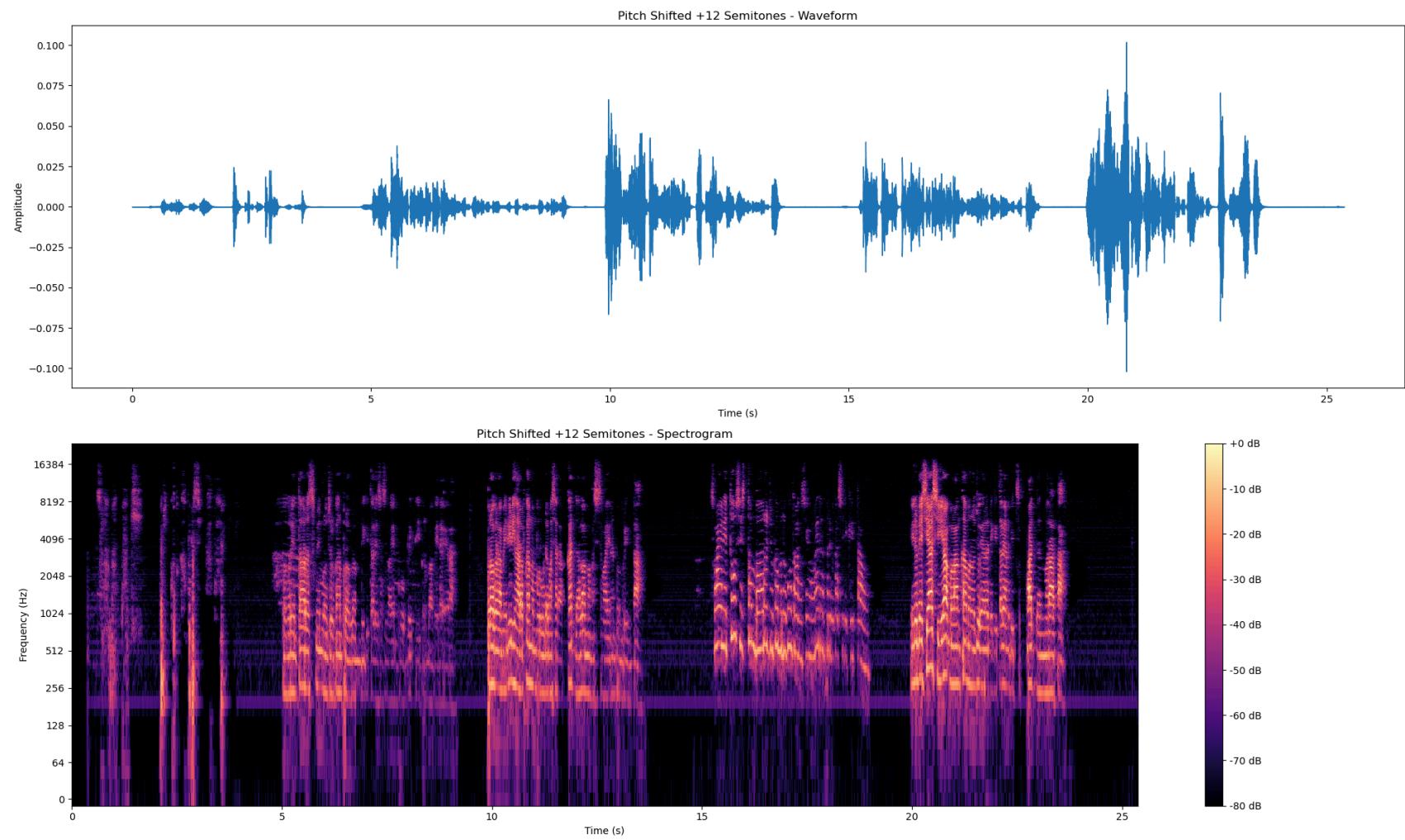
# Gabungkan Audio Pitch Shifted +7 dan +12
y_combined = np.concatenate((y_pitch_7, y_pitch_12))
visualize_audio(y_combined, sr, title="Combined Audio (Pitch Shifted +7 and +12)")
sf.write('Data/rekaman_combined.wav', y_combined, sr)

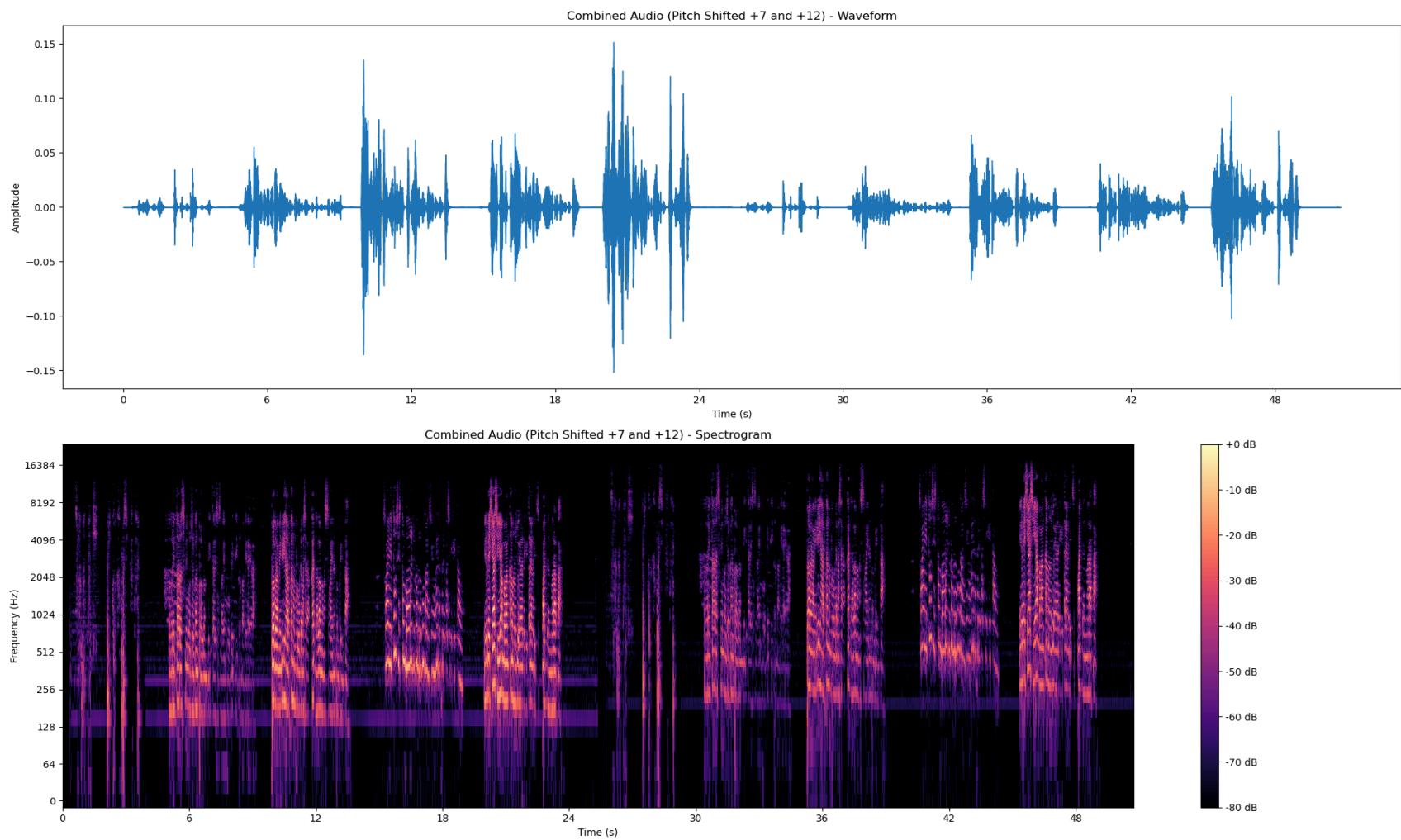
# Play Audio Before Pitch Shifting and After Pitch Shifting
print('Audio Before Pitch Shifting')
display(ipd.Audio(y, rate=sr))
print('Audio After Pitch Shifting +7 Semitones')
display(ipd.Audio(y_pitch_7, rate=sr))
print('Audio After Pitch Shifting +12 Semitones')
display(ipd.Audio(y_pitch_12, rate=sr))
```

```
print('Combined Audio (Pitch Shifted +7 and +12)')
display(ipd.Audio(y_combined, rate=sr))
```









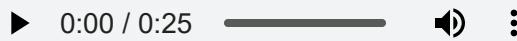
Audio Before Pitch Shifting

▶ 0:00 / 0:25 🔊 ⋮

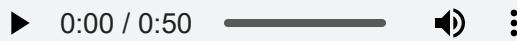
Audio After Pitch Shifting +7 Semitones

▶ 0:00 / 0:25 🔊 ⋮

Audio After Pitch Shifting +12 Semitones



Combined Audio (Pitch Shifted +7 and +12)



Penjelasan Proses Pitch Shifting

1. Parameter yang digunakan

- Proses pitch shifting dilakukan menggunakan fungsi librosa.effects.pitch_shift dari pustaka Librosa.
- Rekaman pertama dinaikkan sebesar +7 semitones (n_steps=7).
- Rekaman kedua dinaikkan sebesar +12 semitones (n_steps=12), atau setara dengan satu oktaf lebih tinggi.

2. Perbedaan dalam Representasi Visual

- Pada Wavefrom terlihat mirip dengan rekaman asli.
- Pada Spectrogram terlihat area terang yang sebelumnya berada di frekuensi rendah dan sekarang berada di frekuensi sedikit tinggi. Keliatan naik.

3. Pengaruh Perubahan Pitch pada Kualitas dan Kejelasan Suara

- Perubahan pitch secara drastis mengubah karakteristik suara menjadi seperti "chipmunk".
- Pada pitch +7, suara masih cukup jelas meskipun sudah tinggi.
- Pada pitch +12, kejelasan suara mulai menurun.

4. Proses Penggabungan Audio

- Proses ini dilakukan dengan metode konkatenasi (penyambungan), di mana rekaman +12 diputar tepat setelah rekaman +7 selesai.

Soal 4. Audio Processing Chain

```
In [22]: import pyloudnorm as pyln

# Fungsi untuk memotong bagian senyap dari audio
def apply_trim(y, top_db=20):
    yt, index = librosa.effects.trim(y, top_db=top_db)
    print(f"Pemangkasan senyap: Durasi berkurang dari {len(y)/sr:.2f}s menjadi {len(yt)/sr:.2f}s")
    return yt

# Fungsi untuk menerapkan equalisasi (filter low-pass)
def apply_eq(data, sr, cutoff=8000, order=5):
    nyquist = 0.5 * sr
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = lfilter(b, a, data)
    print("Equalisasi telah diterapkan")
    return y

# Fungsi untuk menerapkan noise gate (menghilangkan suara di bawah threshold)
def apply_noise_gate(y, sr, frame_length=2048, hop_length=512, threshold_db=-40):
    print("Menerapkan Noise Gate...")
    threshold = librosa.db_to_amplitude(threshold_db)
    y_gated = np.copy(y)

    # Menghitung RMS energy untuk setiap frame
    rms = librosa.feature.rms(y=y, frame_length=frame_length, hop_length=hop_length)[0]

    # Membisukan frame yang amplitudonya di bawah threshold
    for i in range(len(rms)):
        if rms[i] < threshold:
            start = i * hop_length
            end = min(start + frame_length, len(y))
            y_gated[start:end] = 0

    return y_gated

# Fungsi untuk menerapkan fade in dan fade out
def apply_fade(y, sr, fade_in_ms=100, fade_out_ms=200):
    fade_in_samples = int(sr * (fade_in_ms / 1000.0))
    fade_out_samples = int(sr * (fade_out_ms / 1000.0))

    # Membuat kurva fade in dan fade out
```

```
fade_in = np.linspace(0, 1, fade_in_samples)
fade_out = np.linspace(1, 0, fade_out_samples)

# Menerapkan fade in di awal dan fade out di akhir
y[:fade_in_samples] *= fade_in
y[-fade_out_samples:] *= fade_out

print("Fade in dan Fade out telah diterapkan")
return y

# Fungsi untuk normalisasi Loudness audio
def apply_normalization(y, sr, target_loudness=-16.0):
    meter = pyln.Meter(sr)
    loudness = meter.integrated_loudness(y)
    y_normalized = pyln.normalize.loudness(y, loudness, target_loudness)
    print(f"Normalisasi telah diterapkan: Loudness berubah dari {loudness:.2f} LUFS menjadi {target_loudness} LUFS")
    return y_normalized

# Fungsi Visualisasi Waveform dan Spectrogram
def visualize_audio(y, sr, title="Sinyal Audio"):
    plt.figure(figsize=(20, 12))

    # Waveform
    plt.subplot(2, 1, 1)
    librosa.display.waveshow(y, sr=sr)
    plt.title(f"{title} - Waveform")
    plt.xlabel("Waktu (s)")
    plt.ylabel("Amplitudo")

    # Spectrogram
    plt.subplot(2, 1, 2)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar(format='%.2f dB')
    plt.title(f"{title} - Spectrogram")
    plt.xlabel("Waktu (s)")
    plt.ylabel("Frekuensi (Hz)")

    plt.tight_layout()
    plt.show()

# Memuat File Audio
```

```
audio_path = 'Data/rekaman_combined.wav'
y, sr = librosa.load(audio_path, sr=None)

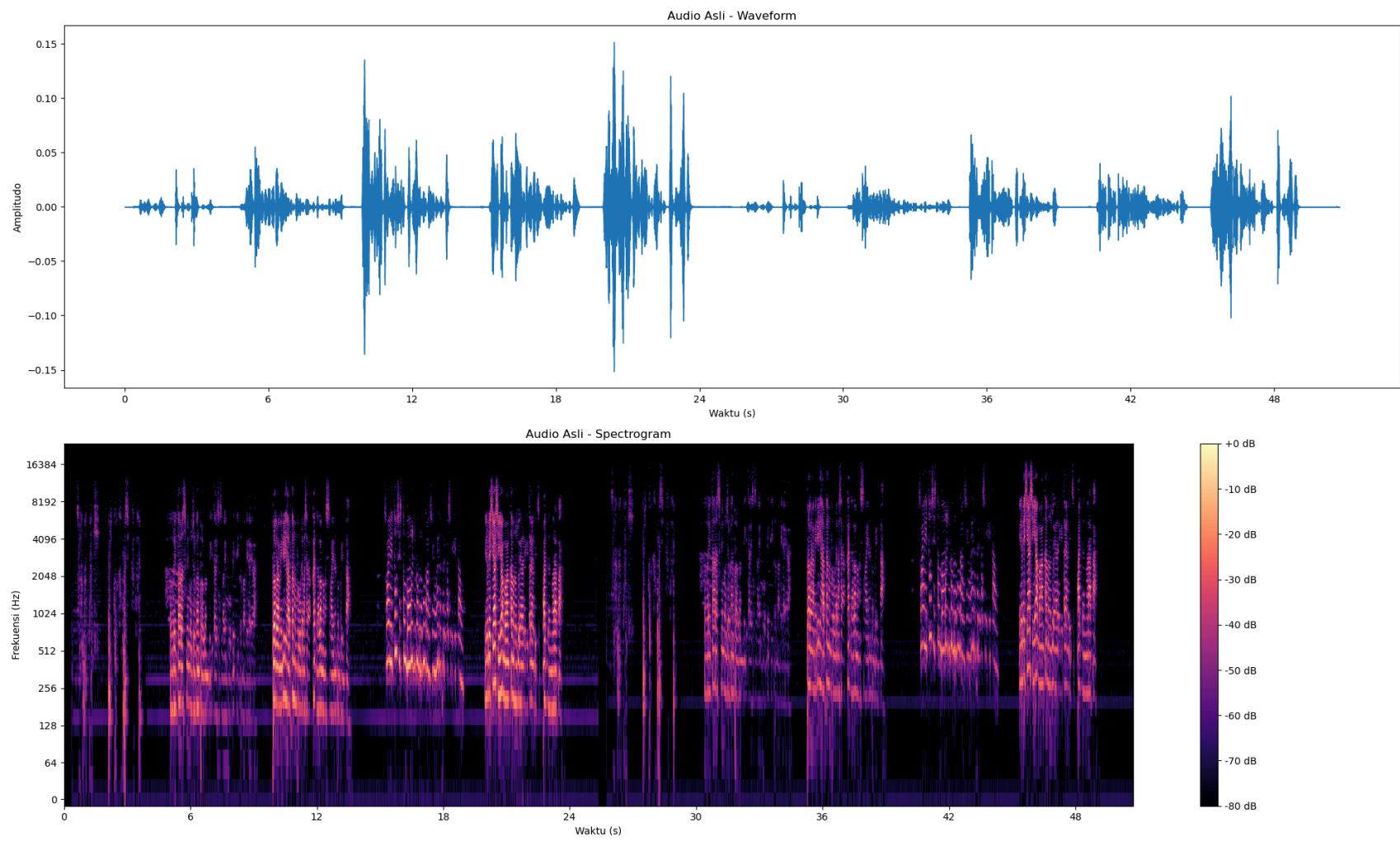
# Visualisasi Audio Asli
visualize_audio(y, sr, title="Audio Asli")

# Menerapkan Rangkaian Pemrosesan Audio
print("\n==== Memulai Pemrosesan Audio ====")
y_trimmed = apply_trim(y)
y_eq = apply_eq(y_trimmed, sr)
y_faded = apply_fade(y_eq, sr)
y_final = apply_normalization(y_faded, sr)
print("==== Pemrosesan Audio Selesai ===\n")

# Visualisasi Audio Setelah Processing Chain
visualize_audio(y_final, sr, title="Audio Setelah Pemrosesan")

# Menyimpan Audio Setelah Processing Chain
sf.write('Data/rekaman_processed.wav', y_final, sr)
print(f"Audio yang telah diproses disimpan ke: Data/rekaman_processed.wav")

# Play Audio Before and After Processing Chain
print('Audio Sebelum Pemrosesan')
display(ipd.Audio(y, rate=sr))
print('Audio Setelah Pemrosesan')
display(ipd.Audio(y_final, rate=sr))
```



== Memulai Pemrosesan Audio ==

Pemangkasan senyap: Durasi berkurang dari 50.73s menjadi 46.82s

Equalisasi telah diterapkan

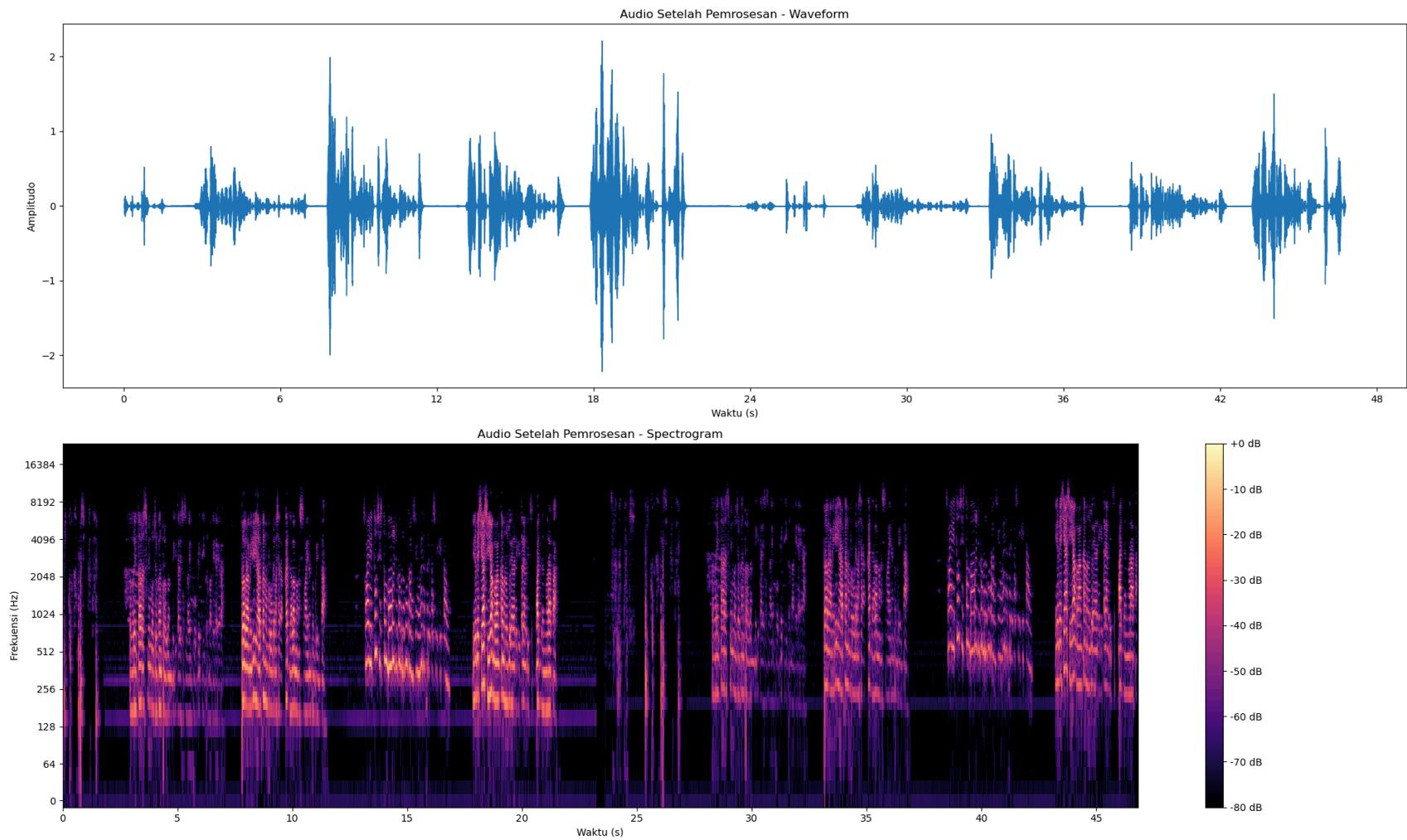
Fade in dan Fade out telah diterapkan

Normalisasi telah diterapkan: Loudness berubah dari -39.32 LUFS menjadi -16.0 LUFS

== Pemrosesan Audio Selesai ==

```
c:\Users\arwid\miniconda3\envs\multimedia\lib\site-packages\pyloudnorm\normalize.py:62: UserWarning: Possible clipped samples in output.
```

```
    warnings.warn("Possible clipped samples in output.")
```



Audio yang telah diproses disimpan ke: Data/rekaman_processed.wav

Audio Sebelum Pemrosesan

▶ 0:00 / 0:50 ━━ 🔊 ⋮

Audio Setelah Pemrosesan

▶ 0:00 / 0:46 ━━ 🔊 ⋮

Analisis

1. Perubahan Dinamika Suara yang Terjadi

- Sebelum: Waveform tampak kurus dengan perbedaan volume ekstrem antara pelan dan keras.
- Sesudah: Waveform lebih padat dan seimbang berkat kompresi dan normalisasi LUFS, membuat suara lebih rata dan terkontrol.

2. Perbedaan antara normalisasi peak dan normalisasi LUFS

- Normalisasi Peak adalah metode yang menyesuaikan volume berdasarkan titik suara paling keras dalam audio. Volume dinaikkan hingga puncak tersebut mencapai 0 dB (atau batas aman seperti -0,1 dB). Metode ini tidak mempertimbangkan persepsi kenyaringan manusia, sehingga audio tetap dapat terdengar pelan meskipun memiliki satu suara keras sesaat.
- Normalisasi LUFS (Loudness Units Full Scale) merupakan standar modern yang menyesuaikan tingkat kenyaringan berdasarkan persepsi pendengaran manusia. Dengan menargetkan -16 LUFS, audio memiliki tingkat kenyaringan yang konsisten dengan standar siaran dan platform musik, sehingga pendengar tidak perlu terus mengatur volume.

3. Kualitas suara berubah setelah proses normalisasi dan loudness optimization

- Kompresi dan normalisasi membuat bagian pelan terdengar jelas tanpa menyebabkan distorsi pada bagian keras.
- Efek fade-in dan fade-out membuat awal dan akhir audio terdengar alami tanpa potongan tiba-tiba.

4. Kelebihan dan kekurangan dari pengoptimalan loudness dalam konteks rekaman suara

kekurangan

- Perbedaan antara suara pelan dan keras jadi berkurang, sehingga ekspresi emosional terasa lebih datar.

kelebihan

- Memastikan setiap bagian audio dapat terdengar dengan jelas.

Soal 5. Music Analysis and Remix

In []: `from pydub import AudioSegment`

```
# Fungsi Analisis Audio
def analyze_audio(audio_path):
    # Menganalisis audio untuk mendapatkan tempo dan kunci nada
    y, sr = librosa.load(audio_path)
    tempo, _ = librosa.beat.beat_track(y=y, sr=sr)
    chromagram = librosa.feature.chroma_stft(y=y, sr=sr)
    key_val = np.argmax(np.mean(chromagram, axis=1))
    notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
    key = notes[key_val]
    return tempo, key, y, sr

# Fungsi Visualisasi Audio Waveform dan Spectrogram
def visualize_audio(y, sr, title="Sinyal Audio"):
    # Membuat visualisasi waveform dan spectrogram dari audio
    plt.figure(figsize=(20, 12))

    # Waveform
    plt.subplot(2, 1, 1)
    librosa.display.waveshow(y, sr=sr)
    plt.title(f"{title} - Bentuk Gelombang")
    plt.xlabel("Waktu (detik)")
    plt.ylabel("Amplitudo")

    # Spectrogram
    plt.subplot(2, 1, 2)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar(format='%.2f dB')
    plt.title(f"{title} - Spektrogram")
    plt.xlabel("Waktu (detik)")
    plt.ylabel("Frekuensi (Hz)")

    plt.tight_layout()
    plt.show()

# Memuat File Audio
file_lagu_sedih = "Data/ardhito_pramono_i_just_couldn't_save_you_tonight.wav"
file_lagu_ceria = "Data/bruno_mars_locked_out_of_heaven.wav"

# Analisis Audio untuk Mendeteksi Tempo dan Kunci Nada
tempo1, key1, y1, sr1 = analyze_audio(file_lagu_sedih)
tempo2, key2, y2, sr2 = analyze_audio(file_lagu_ceria)
```

```
print(f"File: {file_lagu_sedih}, Tempo: {tempo1[0]:.2f} BPM, Kunci Nada: {key1}")
print(f"File: {file_lagu_ceria}, Tempo: {tempo2[0]:.2f} BPM, Kunci Nada: {key2}")

# Meremix Audio Berdasarkan Tempo
stretch_rate = tempo2[0]/tempo1[0]
y1_stretched = librosa.effects.time_stretch(y1, rate=stretch_rate)
print(f"'I Just Couldn't Save You Tonight' direngangkan sebesar {stretch_rate:.2f} kali untuk menyesuaikan tempo 'Lock')

# Mengubah Pitch untuk Menyesuaikan Kunci Nada
notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
semitone_to_shift = (notes.index(key2) - notes.index(key1)) % 12
y1_shifted = librosa.effects.pitch_shift(y=y1_stretched, sr=sr1, n_steps=semitone_to_shift)
print(f"'I Just Couldn't Save You Tonight' digeser pitch sebesar {semitone_to_shift} semitone untuk menyesuaikan kunci nada")

# Menyamakan Panjang Audio untuk CrossFade
min_len = min(len(y1_shifted), len(y2))
sf.write('temp_song1.wav', y1_shifted[:min_len], sr1)
sf.write('temp_song2.wav', y2[:min_len], sr2)

# Memuat Audio dengan Pydub untuk CrossFade
sound1 = AudioSegment.from_wav('temp_song1.wav')
sound2 = AudioSegment.from_wav('temp_song2.wav')
crossfade_duration = 8000 # Durasi crossfade dalam milidetik

# Menggabungkan Kedua Lagu dengan Efek CrossFade
remix = sound1.fade_out(crossfade_duration).overlay(sound2.fade_in(crossfade_duration))
remix.export('Data/remix_song.wav', format='wav')
print(f'Menggabungkan kedua lagu dengan crossfade selama {crossfade_duration/1000:.2f} detik')

# Visualisasi Audio Setelah Remix
print("Visualisasi Audio Setelah Remix")
y_remix, sr_remix = librosa.load('Data/remix_song.wav', sr=None)
visualize_audio(y_remix, sr_remix, title="Audio Hasil Remix")

# Memutar Audio Hasil Remix
print('Audio Hasil Remix')
display(ipd.Audio(y_remix, rate=sr_remix))
```

File: Data/ardhito_pramono_i_just_couldn't_save_you_tonight.wav, Tempo: 112.35 BPM, Kunci Nada: C

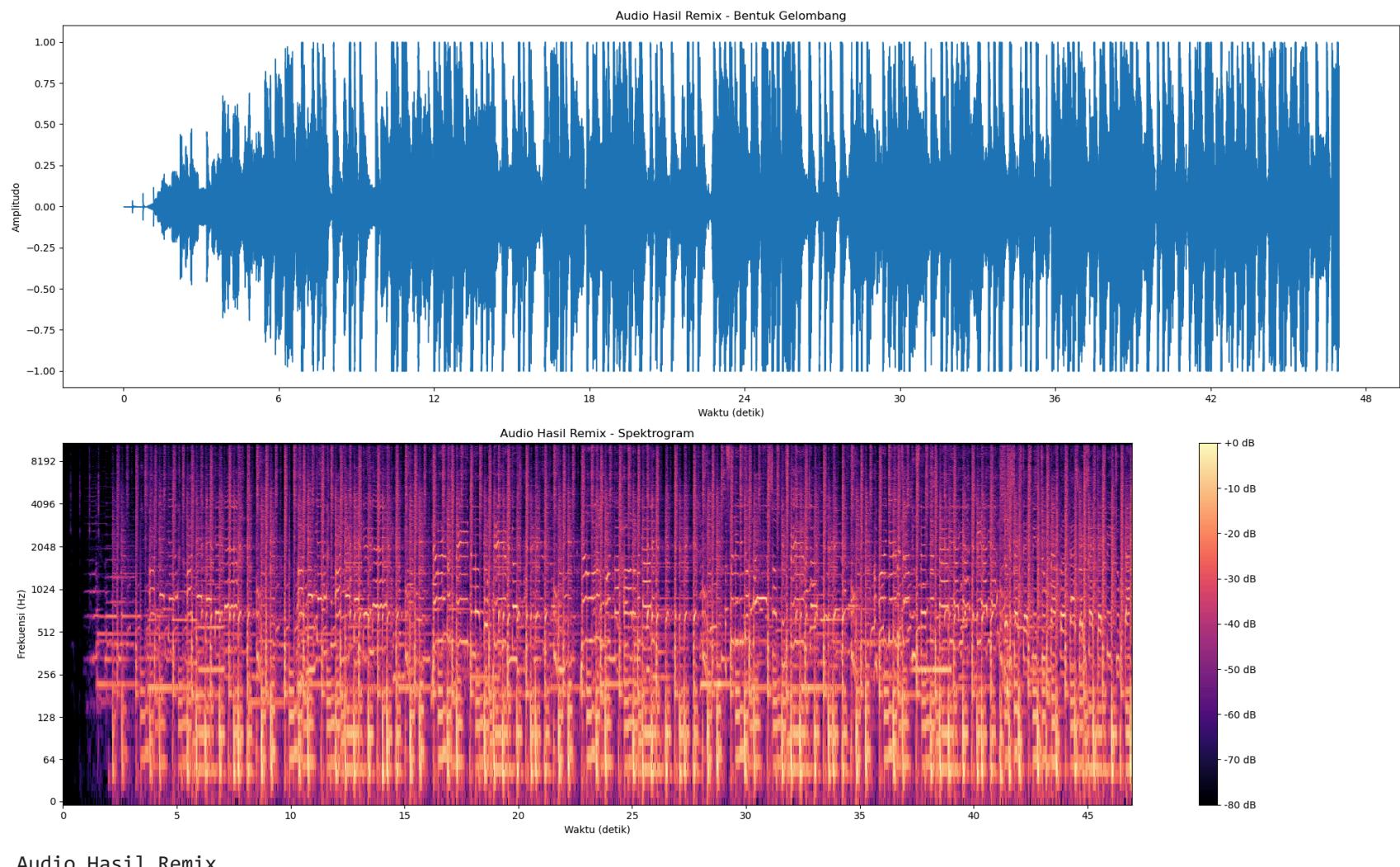
File: Data/bruno_mars_locked_out_of_heaven.wav, Tempo: 143.55 BPM, Kunci Nada: A

'I Just Couldn't Save You Tonight' diregangkan sebesar 1.28 kali untuk menyesuaikan tempo 'Locked Out of Heaven'

'I Just Couldn't Save You Tonight' digeser pitch sebesar 9 semitone untuk menyesuaikan kunci nada 'Locked Out of Heaven'

Menggabungkan kedua lagu dengan crossfade selama 8.00 detik

Visualisasi Audio Setelah Remix





Penjelasan dan Analisis

Lagu "I Just Couldn't Save You Tonight" memiliki Tempo: 112.35 BPM, Kunci Nada: C

Lagu "Locked Out of Heaven" memiliki Tempo: 143.55 BPM, Kunci Nada: A

Proses dan Parameter yang Digunakan: 'I Just Couldn't Save You Tonight' diregangkan sebesar 1.28 kali untuk menyesuaikan tempo 'Locked Out of Heaven' 'I Just Couldn't Save You Tonight' digeser pitch sebesar 9 semitone untuk menyesuaikan kunci nada 'Locked Out of Heaven' Menggabungkan kedua lagu dengan crossfade selama 8.00 detik.

Hasil Remix: Kualitas Audio terdengar jadi kurang jelas, terdengar sangat tinggi, dan lagu1 terlihat kalah dengan lagu2.

Github Repo: https://github.com/erc-a/Multimedia_Handson

Referensi : Gemini AI : <https://gemini.ai/>