

Hands-On Audio Processing

IF4021 - Multimedia Information Processing

Nama: Dzaki Gastiadirrjal

NIM: 122140030

Link Github Repository: <https://github.com/Dzaki-G/Hands-on-audio>

Deskripsi Tugas Tugas ini dirancang untuk menguji pemahaman mahasiswa terhadap konsep-konsep fundamental dalam pemrosesan audio digital termasuk manipulasi sinyal audio, filtering, pitch shifting, normalisasi, dan teknik remix audio. Mahasiswa diharapkan dapat menerapkan teori yang telah dipelajari dalam praktik langsung menggunakan Python dan pustaka pemrosesan audio.

Library

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import librosa
import soundfile as sf
import scipy
from scipy.signal import butter, filtfilt, lfilter
from IPython.display import Audio, HTML, display
import os
import pyloudnorm as pyln

print("Library yang digunakan:")
print(f"NumPy      : {np.__version__}")
print(f"Matplotlib : {plt.matplotlib.__version__}")
print(f"Librosa    : {librosa.__version__}")
print(f"SciPy      : {scipy.__version__}")
print(f"SoundFile   : {sf.__version__}")
```

Library yang digunakan:

```
NumPy      : 2.2.6
Matplotlib : 3.10.3
Librosa    : 0.11.0
SciPy      : 1.16.1
SoundFile   : 0.13.1
```

Soal 1: Rekaman dan Analisis Suara Multi-Level

- Rekamlah suara Anda sendiri selama 25 detik dimana Anda membaca sebuah teks berita.
- Visualisasikan waveform dan spektrogram dari rekaman suara Anda.
- Sertakan penjelasan singkat mengenai hasil visualisasi tersebut.
- Lakukan resampling pada file audio Anda kemudian bandingkan kualitas dan durasinya.

```
In [ ]: Path_Soal_Audio1 = os.path.join(os.getcwd(), 'data', 'soal1_audio.wav')

if os.path.exists(Path_Soal_Audio1):
    y, sr = librosa.load(Path_Soal_Audio1, sr=None)
    source_info = f'Lokasi File: {Path_Soal_Audio1}'


print(source_info)
print(f"Shape: {y.shape}")
print(f"Sample rate: {sr:,} Hz")
print(f"Durasi: {len(y)/sr:.2f} detik")
```

Lokasi File: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\data\soal1_audio.wav
 Shape: (1231488,)
 Sample rate: 48,000 Hz
 Durasi: 25.66 detik

Visualisasi Waveform dan Spectrogram

```
In [ ]: # Plot waveform
duration = len(y) / sr
time = np.linspace(0, duration, len(y))

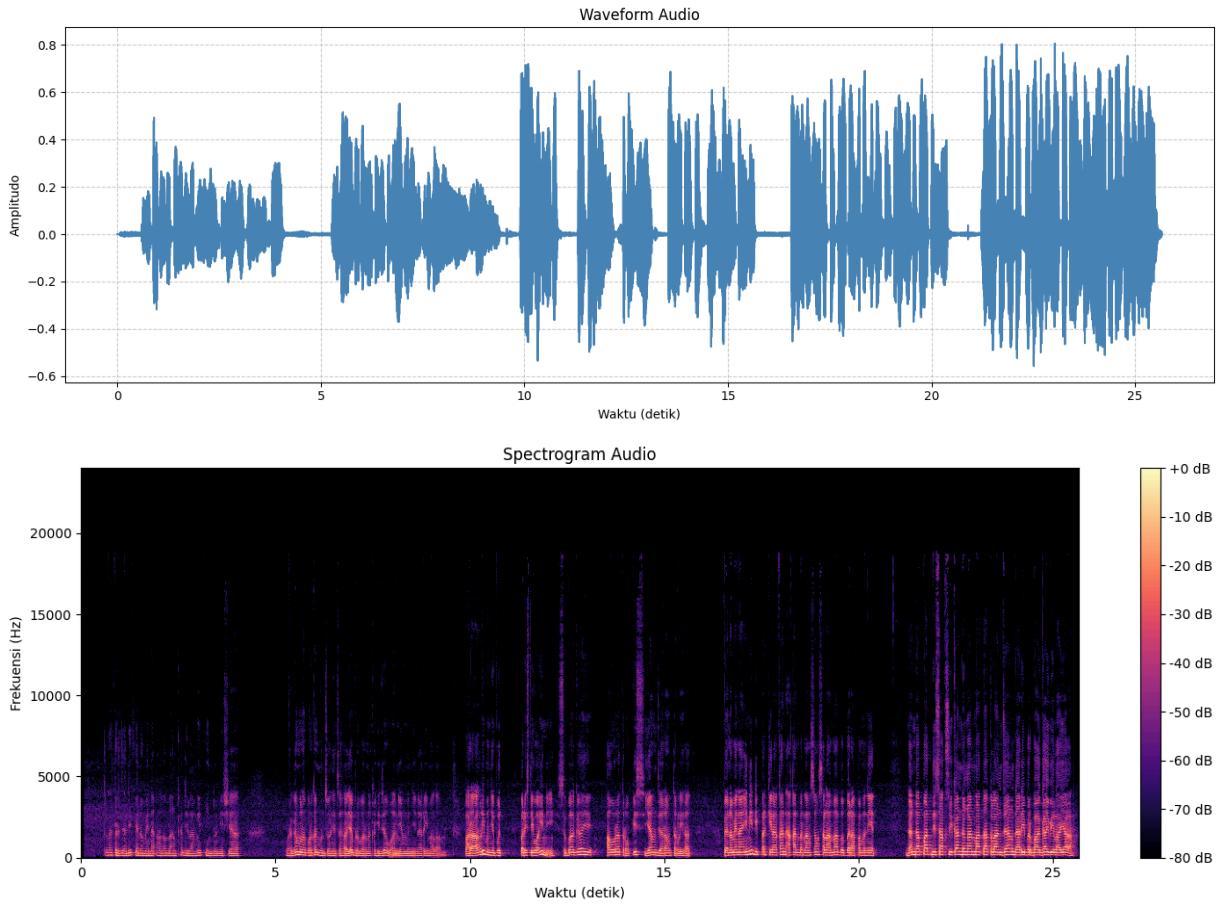
plt.figure(figsize=(14, 5))
plt.plot(time, y, color='steelblue')
plt.title('Waveform Audio')
plt.xlabel('Waktu (detik)')
plt.ylabel('Amplitudo')
plt.grid(True, which='both', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

# Plot spectrogram
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)

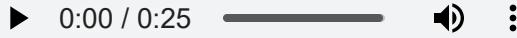
plt.figure(figsize=(14, 5))
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram Audio')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')
plt.tight_layout()
plt.show()

# Audio player
```

```
print("Audio Player:")
display(Audio(y, rate=sr))
```



Audio Player:



Penjelasan:

Dari hasil visualisasi rekaman audio selama 25 detik ini terlihat menggambarkan peningkatan volume dan energi suara secara bertahap.

5 detik pertama diawali dengan suara bisikan yang ditandai oleh amplitudo sangat kecil dan spektrum frekuensi yang gelap.

detik ke-5 hingga ke-10, suara berubah menjadi level normal dengan amplitudo yang lebih besar dan warna spectrogram yang lebih cerah.

Selanjutnya, pada detik ke-10 hingga ke-15, suara menjadi keras, yang tercermin dari amplitudo yang jauh lebih tinggi dan spectrogram yang sangat terang di berbagai frekuensi. Titik paling tinggi ada pada 10 detik terakhir, suara cempreng dan teriakan membuat amplitudo mencapai titik maksimal.

Pada spectrogram, periode ini ditandai dengan penekanan energi di frekuensi tinggi untuk suara cempreng, yang diakhiri dengan warna paling terang di seluruh spektrum untuk suara teriakan.

Tidak banyak frekuensi tinggi mungkin dikarenakan microphone Hp yang tidak bagus menyebabkan suara menjadi bass dan sedikit dengan frekuensi tinggi.

Resampling Audio

```
In [ ]: # Resampling Audio
print("Sample rate asli:", sr)

# Target resampling 16000 Hz
target_sr = 16000
y_resampled = librosa.resample(y=y, orig_sr=sr, target_sr=target_sr)
print("Sample rate baru:", target_sr)

# Menyimpan output resampling
output_dir = os.path.join(os.getcwd(), 'output')
os.makedirs(output_dir, exist_ok=True)
output_filename = 'Soal1_Audio_Resampled.wav'
output_path = os.path.join(output_dir, output_filename)

sf.write(output_path, y_resampled, target_sr)

# Muat ulang hasil resampling untuk memastikan
Path_Output_Resampled = os.path.join(os.getcwd(), 'output', 'Soal1_Audio_Resampled.wav')
if os.path.exists(Path_Output_Resampled):
    y_resampled, sr_resampled = librosa.load(Path_Output_Resampled, sr=None)

# Spectrogram sebelum (Original)
D_before = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
plt.figure(figsize=(14, 5))
librosa.display.specshow(D_before, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar(format='%.2f dB')
plt.title('Spectrogram Original 48000 kHz')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')
plt.tight_layout()
plt.show()

# Spectrogram sesudah (Resampled)
D_after = librosa.amplitude_to_db(np.abs(librosa.stft(y_resampled)), ref=np.max)
plt.figure(figsize=(14, 5))
librosa.display.specshow(D_after, sr=sr_resampled, x_axis='time', y_axis='hz')
plt.colorbar(format='%.2f dB')
plt.title('Spectrogram Sesudah Resampling 16000 kHz')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')
plt.tight_layout()
plt.show()

# Perbandingan Audio Player
print("Audio Player Sebelum Resampling:")
```

```

display(Audio(y, rate=sr))

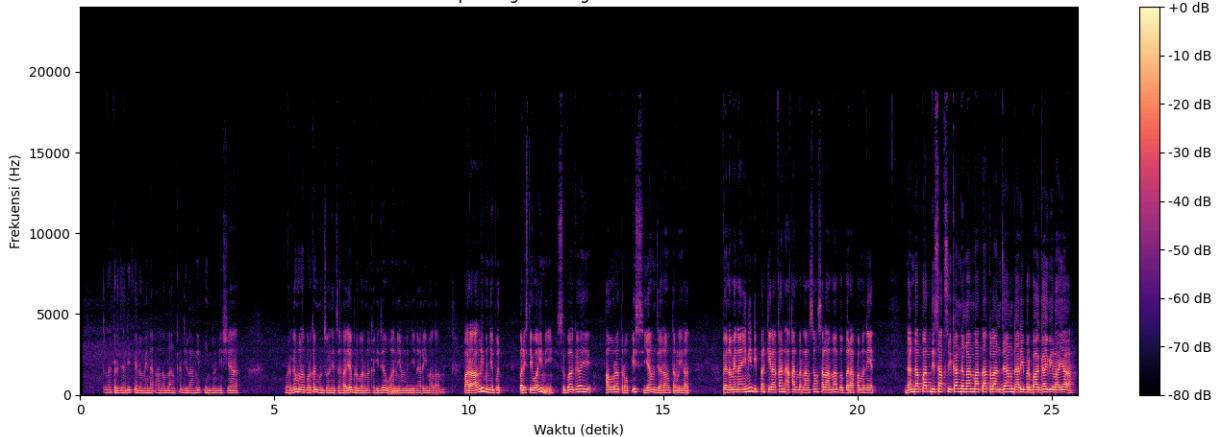
print("Audio Player Sesudah Resampling:")
display(Audio(y_resampled, rate=sr_resampled))

```

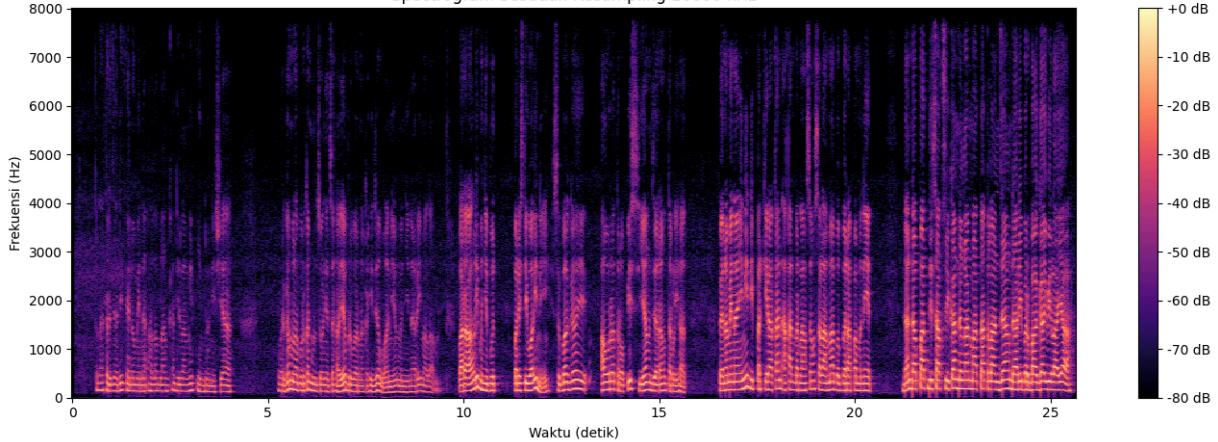
Sample rate asli: 48000

Sample rate baru: 16000

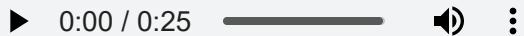
Spectrogram Original 48000 kHz



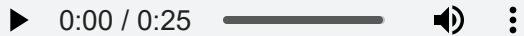
Spectrogram Sesudah Resampling 16000 kHz



Audio Player Sebelum Resampling:



Audio Player Sesudah Resampling:



Penjelasan

Terlihat bahwa perbedaan dari hasil setelah dilakukan downsampling. Pada audio asli, frekuensi bisa terlihat hingga ke 15000 Hz, sedangkan setelah dilakukan downsampling menjadi 8000 Hz. Hal ini menyebabkan hasil perbandingan audio player juga menunjukkan kualitas suara yang menurun untuk hasil setelah downsampling, dimana suara menjadi sedikit lebih pecah dan kurang jernih dibandingkan audio asli.

Soal 2: Noise Reduction dengan Filtering

- Rekam suara Anda berbicara di sekitar objek yang berisik (seperti kipas angin, AC, atau mesin)
- Gunakan filter equalisasi (high-pass, low-pass, dan band-pass) untuk menghilangkan noise pada rekaman tersebut.
- Lakukan eksperimen dengan berbagai nilai frekuensi cutoff (misalnya 500 Hz, 1000 Hz, 2000 Hz).
- Visualisasikan hasil dari tiap filter dan bandingkan spektrogramnya

Jelaskan: – Jenis noise yang muncul pada rekaman Anda – Filter mana yang paling efektif untuk mengurangi noise tersebut – Nilai cutoff yang memberikan hasil terbaik – Bagaimana kualitas suara (kejelasan ucapan) setelah proses filtering

```
In [ ]: Path_Soal_Audio2 = os.path.join(os.getcwd(), 'data', 'soal2_audio.wav')

if os.path.exists(Path_Soal_Audio2):
    y2, sr2 = librosa.load(Path_Soal_Audio2, sr=None)
    source_info2 = f'Lokasi File: {Path_Soal_Audio2}'

print(source_info2)
print(f"Shape: {y2.shape}")
print(f"Sample rate: {sr2:,} Hz")
print(f"Durasi: {len(y2)/sr2:.2f} detik")
```

```
Lokasi File: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\data\soal2_audio.wav
Shape: (486144,)
Sample rate: 48,000 Hz
Durasi: 10.13 detik
```

Low-Pass Filtering

```
In [ ]: # === Low-pass Filtering & Visualisasi Singkat ===

# Desain & terapkan Low-pass filter Langsung
cutoff_lp = 5000
b, a = butter(4, cutoff_lp / (0.5 * sr2), btype='low', analog=False)
y2_lowpassed = filtfilt(b, a, y2)

# Simpan hasil filter
output_path = os.path.join(os.getcwd(), 'output', 'Soal2_LowPass.wav')
os.makedirs(os.path.dirname(output_path), exist_ok=True)
sf.write(output_path, y2_lowpassed, sr2)
print(f"\n==== Low-pass filtering selesai! ===\nHasil disimpan di: {output_path}")

# Tampilkan spektrogram sebelum & sesudah
signals = {'Sebelum Filtering': y2, 'Sesudah Low-pass': y2_lowpassed}
plt.figure(figsize=(14, 8))
```

```

for i, (title, sig) in enumerate(signals.items(), 1):
    S_db = librosa.amplitude_to_db(np.abs(librosa.stft(sig)), ref=np.max)
    plt.subplot(2, 1, i)
    librosa.display.specshow(S_db, sr=sr2, x_axis='time', y_axis='hz')
    plt.colorbar(format='%+2.0f dB')
    plt.title(f"Spectrogram {title}")
    plt.xlabel('Waktu (detik)')
    plt.ylabel('Frekuensi (Hz)')

plt.tight_layout()
plt.show()

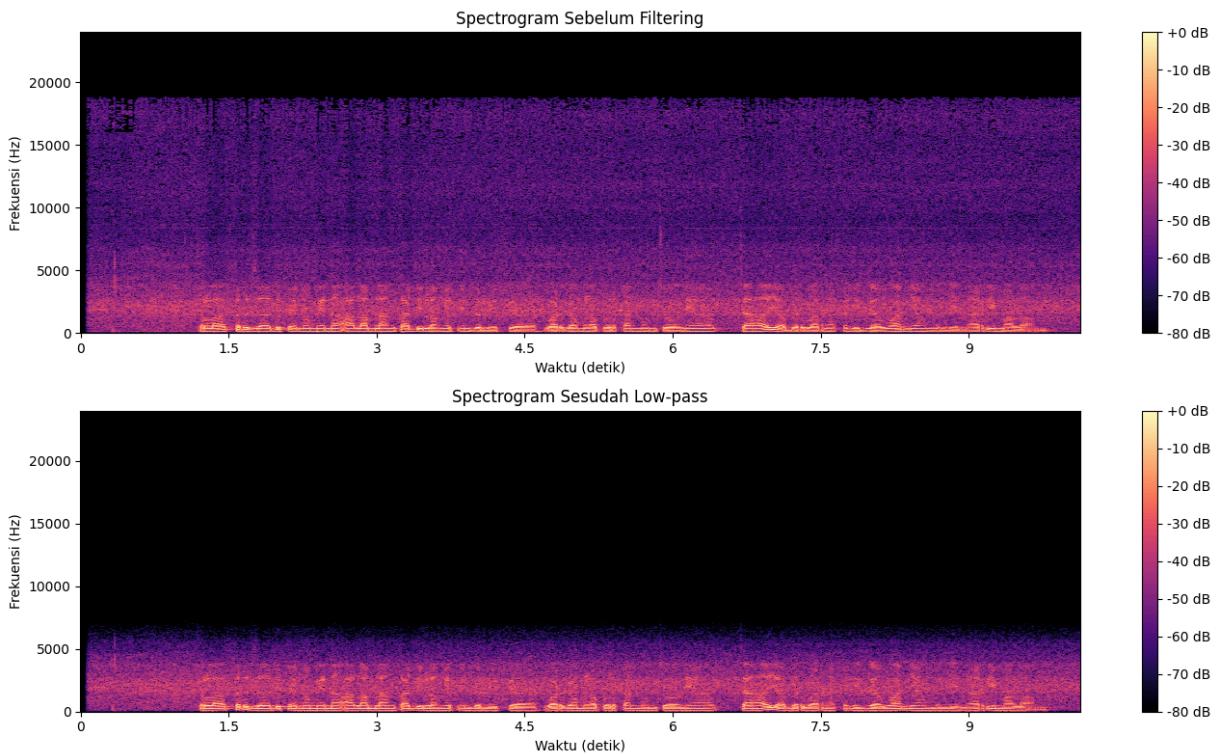
# Audio playback
print("Audio Player Sebelum Filtering:")
display(Audio(y2, rate=sr2))

print("Audio Player Sesudah Low-pass Filtering:")
display(Audio(y2_lowpassed, rate=sr2))

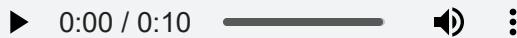
```

==== Low-pass filtering selesai! ===

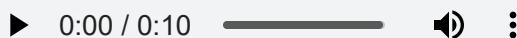
Hasil disimpan di: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\output\Soal2_LowPass.wav



Audio Player Sebelum Filtering:



Audio Player Sesudah Low-pass Filtering:



High-pass Filtering

```
In [ ]: # === High-pass Filtering & Visualisasi Singkat ===

# Desain dan terapkan high-pass filter langsung
cutoff_hp = 200
b, a = butter(4, cutoff_hp / (0.5 * sr2), btype='high', analog=False)
y2_highpassed = filtfilt(b, a, y2)

# Simpan hasil filter
output_path = os.path.join(os.getcwd(), 'output', 'Soal2_HighPass.wav')
os.makedirs(os.path.dirname(output_path), exist_ok=True)
sf.write(output_path, y2_highpassed, sr2)
print(f"\n==== High-pass filtering selesai! ===\nHasil disimpan di: {output_path}")

# Tampilkan spektrogram sebelum & sesudah
signals = {'Sebelum Filtering': y2, 'Sesudah High-pass': y2_highpassed}
plt.figure(figsize=(14, 8))

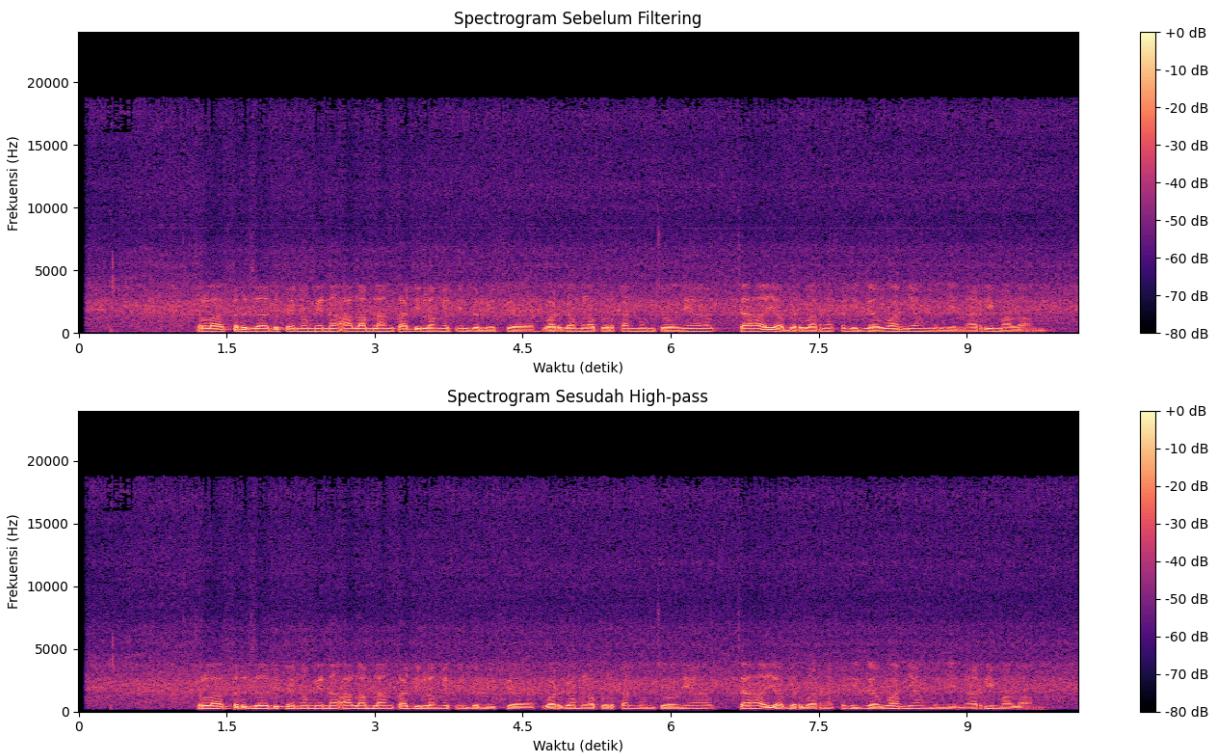
for i, (title, sig) in enumerate(signals.items(), 1):
    S_db = librosa.amplitude_to_db(np.abs(librosa.stft(sig))), ref=np.max
    plt.subplot(2, 1, i)
    librosa.display.specshow(S_db, sr=sr2, x_axis='time', y_axis='hz')
    plt.colorbar(format='%.2f dB')
    plt.title(f"Spectrogram {title}")
    plt.xlabel('Waktu (detik)')
    plt.ylabel('Frekuensi (Hz)')

plt.tight_layout()
plt.show()

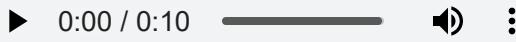
# Audio playback
print("Audio Player Sebelum Filtering:")
display(Audio(y2, rate=sr2))

print("Audio Player Sesudah High-pass Filtering:")
display(Audio(y2_highpassed, rate=sr2))

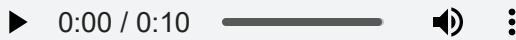
==== High-pass filtering selesai! ===
Hasil disimpan di: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\output\Soal2_HighPass.wav
```



Audio Player Sebelum Filtering:



Audio Player Sesudah High-pass Filtering:



Band-pass Filtering

```
In [ ]: # Band-pass filtering & visualisasi singkat
def bandpass_filter(y, sr, lowcut=200, highcut=5000, order=4):
    nyq = 0.5 * sr
    b, a = butter(order, [lowcut/nyq, highcut/nyq], btype='band')
    return filtfilt(b, a, y)

# Terapkan & simpan
y2_filtered = bandpass_filter(y2, sr2, 200, 5000)
out_path = os.path.join(os.getcwd(), 'output', 'Soal2_BandPass.wav')
os.makedirs(os.path.dirname(out_path), exist_ok=True)
sf.write(out_path, y2_filtered, sr2)
print(f"\n==== Band-pass filtering selesai! ====\nHasil disimpan di: {out_path}")

# Spectrogram sebelum & sesudah
def plot_spec(y, sr, title):
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz')
    plt.colorbar(format='%.2f dB'); plt.title(title); plt.xlabel('Waktu'); plt.ylabel('Frekuensi (Hz)')

plt.figure(figsize=(14, 10))
```

```

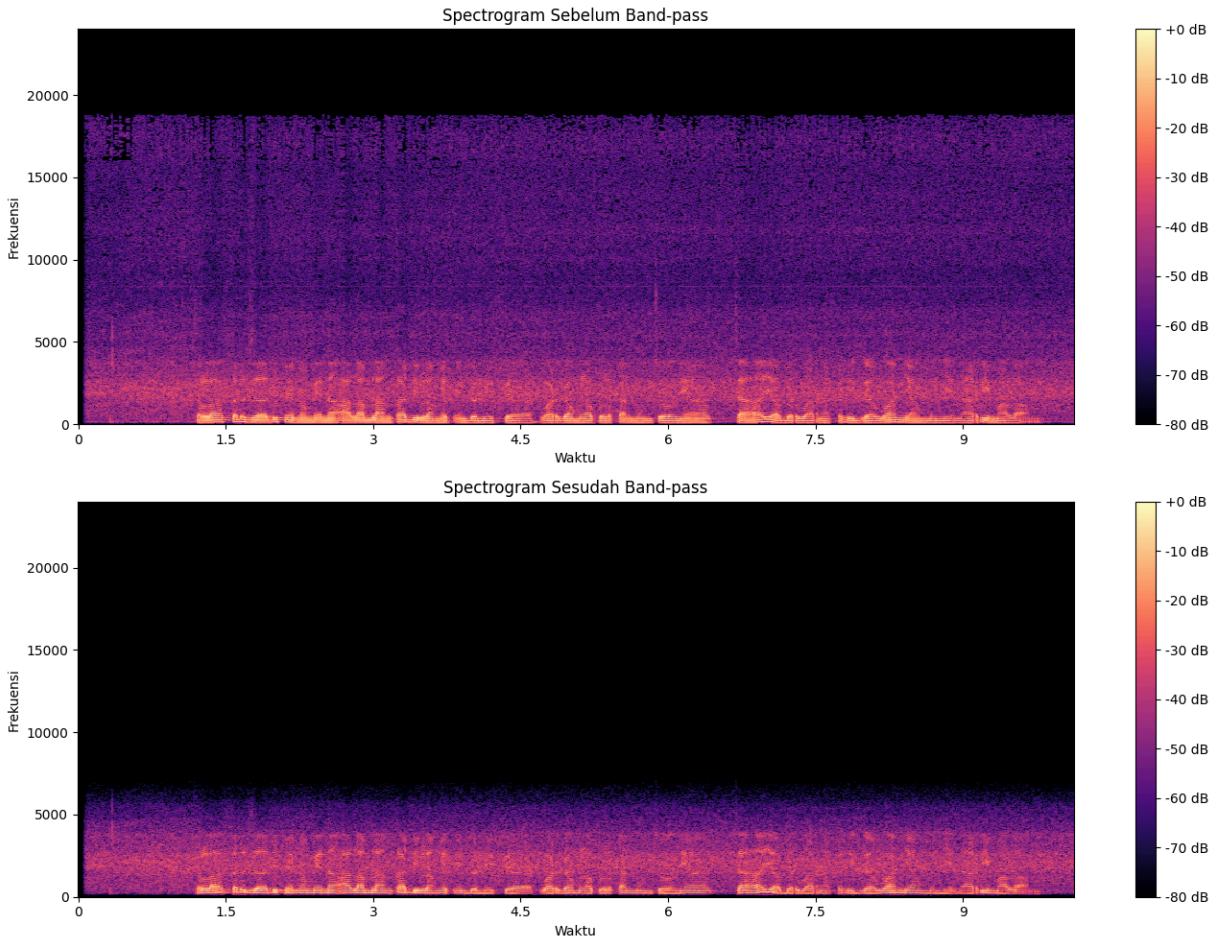
plt.subplot(2,1,1); plot_spec(y2, sr2, 'Spectrogram Sebelum Band-pass')
plt.subplot(2,1,2); plot_spec(y2_filtered, sr2, 'Spectrogram Sesudah Band-pass')
plt.tight_layout(); plt.show()

print("Audio Player Sebelum Filtering:")
display(Audio(y2, rate=sr2))
print("Audio Player Sesudah Band-pass Filtering:")
display(Audio(y2_filtered, rate=sr2))

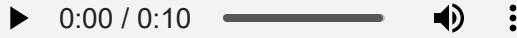
```

== Band-pass filtering selesai! ==

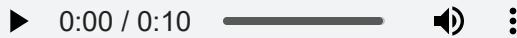
Hasil disimpan di: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\output\Soal2_BandPass.wav



Audio Player Sebelum Filtering:



Audio Player Sesudah Band-pass Filtering:



Penjelasan:

Jelaskan:

- Jenis noise yang muncul pada rekaman Anda: **white noise**

- Filter mana yang paling efektif untuk mengurangi noise tersebut: **Low-Pass** Dikarenakan suara saya terdegar lebih jelas dibandingkan kipas yang ada di background
 - Nilai cutoff yang memberikan hasil terbaik: **5000 kHz pada Low-Pass filter.**
 - Bagaimana kualitas suara (kejelasan ucapan) setelah proses filtering: **Setelah proses filtering, kualitas suara menjadi lebih jernih dan fokus pada frekuensi ucapan. Suara kipas di latar belakang berkurang secara signifikan, sehingga ucapan terdengar lebih jelas dan tidak tertutup oleh noise frekuensi tinggi. Namun, terdapat sedikit penurunan pada detail suara di frekuensi tinggi, tetapi tidak mengganggu kejelasan keseluruhan.**
-

Soal 3: Pitch Shifting dan Audio Manipulation

- Lakukan pitch shifting pada rekaman suara Soal 1 untuk membuat suara terdengar seperti chipmunk (dengan mengubah pitch ke atas).
- Visualisasikan waveform dan spektrogram sebelum dan sesudah pitch shifting.
- Jelaskan proses pitch shifting yang Anda lakukan, termasuk:
 - Parameter yang digunakan
 - Perbedaan dalam representasi visual antara suara asli dan suara yang telah dimodifikasi
 - Bagaimana perubahan pitch memengaruhi kualitas dan kejelasan suara
- Gunakan dua buah pitch tinggi, misalnya pitch +7 dan pitch +12.
- Gabungkan kedua rekaman yang telah di-pitch shift ke dalam satu file audio.

```
In [ ]: Path_Soal_Audio3 = os.path.join(os.getcwd(), 'data', 'soal1_audio.wav')

if os.path.exists(Path_Soal_Audio3):
    y3, sr3 = librosa.load(Path_Soal_Audio3, sr=None)
```

```
In [ ]: # --- Pitch Shifting ---
y_pitch7 = librosa.effects.pitch_shift(y3, sr=sr3, n_steps=7) # Naik 7 semitone
y_pitch12 = librosa.effects.pitch_shift(y3, sr=sr3, n_steps=12) # Naik 12 semitone

# --- Buat spektrogram untuk masing-masing ---
D_orig = librosa.amplitude_to_db(np.abs(librosa.stft(y3)), ref=np.max)
D_pitch7 = librosa.amplitude_to_db(np.abs(librosa.stft(y_pitch7)), ref=np.max)
D_pitch12 = librosa.amplitude_to_db(np.abs(librosa.stft(y_pitch12)), ref=np.max)

# --- Visualisasi Horizontal ---
plt.figure(figsize=(18, 5))
```

```

# Original
plt.subplot(1, 3, 1)
librosa.display.specshow(D_orig, sr=sr3, x_axis='time', y_axis='hz')
plt.colorbar(format='%.+2.0f dB')
plt.title('Original')

# Pitch +7
plt.subplot(1, 3, 2)
librosa.display.specshow(D_pitch7, sr=sr3, x_axis='time', y_axis='hz')
plt.colorbar(format='%.+2.0f dB')
plt.title('Pitch Shift +7 ')

# Pitch +12
plt.subplot(1, 3, 3)
librosa.display.specshow(D_pitch12, sr=sr3, x_axis='time', y_axis='hz')
plt.colorbar(format='%.+2.0f dB')
plt.title('Pitch Shift +12 ')

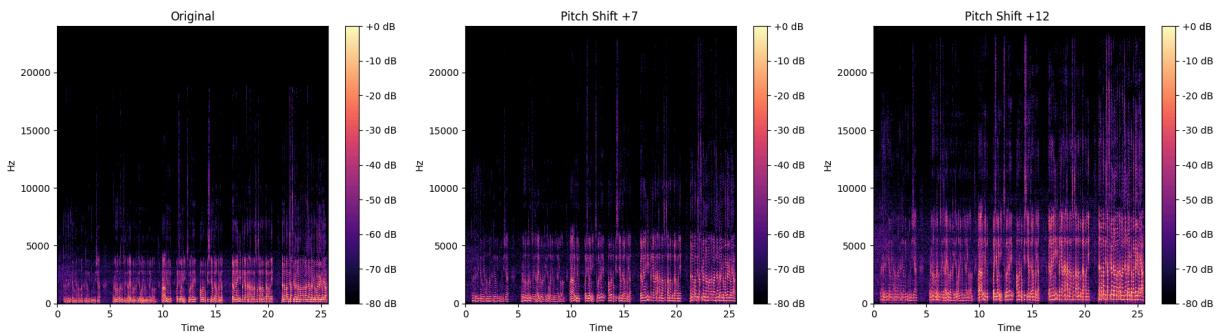
plt.tight_layout()
plt.show()

# --- Audio Player ---
print("Audio Original:")
display(Audio(y3, rate=sr3))

print("Audio Pitch +7:")
display(Audio(y_pitch7, rate=sr3))

print("Audio Pitch +12:")
display(Audio(y_pitch12, rate=sr3))

```



Audio Original:

▶ 0:00 / 0:25 ⏸ ⏴

Audio Pitch +7:

▶ 0:00 / 0:25 ⏸ ⏴

Audio Pitch +12:

▶ 0:00 / 0:25 ⏸ ⏴

Penjelasan:

Parameter yang digunakan:

Parameter **n_steps** merupakan jumlah semitone (nada) yang digeser saat melakukan perubahan pitch pada audio. Nilai ini menentukan tinggi atau rendahnya suara yang dihasilkan. Semakin besar nilai **n_steps** ke arah positif, suara akan menjadi semakin tinggi, sedangkan semakin besar nilai negatif akan membuat suara terdengar semakin rendah.

Semakin tinggi nilai n_steps → suara terdengar cempreng seperti chipmunk.

Semakin rendah nilai n_steps → suara terdengar berat.

Fungsi yang digunakan: **librosa.effects.pitch_shift()** dari library Librosa, yang menjaga durasi audio tetap sama meskipun pitch diubah.

Perbedaan dalam representasi visual antara suara asli dan suara yang telah dimodifikasi:

Pada Spectrogram terlihat bahwa frekuensi pada audio yang telah dimodifikasi (pitch dinaikkan) menjadi lebih rapat atau lebih tinggi dibandingkan dengan audio asli. Hal ini karena pitch yang dinaikkan menyebabkan frekuensi suara menjadi lebih tinggi.

Bagaimana perubahan pitch memengaruhi kualitas dan kejelasan suara:

Ketika pitch dinaikkan, suara akan terdengar lebih tipis dan cempreng, sehingga kejelasan ucapan sedikit berkurang. Meskipun durasi rekaman tidak berubah, suara seolah terdengar lebih cepat dan ringan.

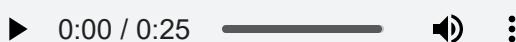
Gabungan 2 Pitch Shifting +7 dan +12

```
In [ ]: # --- Gabungkan hasil pitch shifting ---
# Panjang sinyal bisa berbeda, jadi samakan dulu panjang terpendek
min_len = min(len(y_pitch7), len(y_pitch12))
y_pitch7_trim = y_pitch7[:min_len]
y_pitch12_trim = y_pitch12[:min_len]

# Gabungan sederhana dengan rata-rata
y_pitch_combined = (y_pitch7_trim + y_pitch12_trim) / 2.0

# --- Audio Player ---
print("Gabungan Pitch Shift +7 dan +12:")
display(Audio(y_pitch_combined, rate=sr3))
```

Gabungan Pitch Shift +7 dan +12:



Soal 4: Audio Processing Chain

```
In [ ]: Path_Soal_Audio4 = os.path.join(os.getcwd(), 'data', 'soal3_gabungan_pitch.wav')

if os.path.exists(Path_Soal_Audio4):
    y4, sr4 = librosa.load(Path_Soal_Audio4, sr=None)
    print(f"File ditemukan: {Path_Soal_Audio4}")
else:
    print(f"File tidak ditemukan")
```

File ditemukan: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\data\soal3_gabungan_pitch.wav

Equalizer (High-pass 100 Hz, Low-pass 6000 Hz)

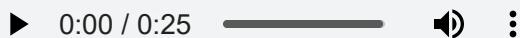
```
In [ ]: # --- Fungsi High-pass Filter ---
def highpass_filter(data, sr, cutoff=100, order=4):
    nyquist = 0.5 * sr
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return filtfilt(b, a, data)

# --- Fungsi Low-pass Filter ---
def lowpass_filter(data, sr, cutoff=6000, order=4):
    nyquist = 0.5 * sr
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data)

# --- Tahap Equalizer: High-pass lalu Low-pass ---
y4_eq = highpass_filter(y4, sr4, cutoff=100)
y4_eq = lowpass_filter(y4_eq, sr4, cutoff=6000)

# --- Audio Player ---
print("Audio setelah Equalizer (High-pass 100 Hz & Low-pass 6000 Hz):")
display(Audio(y4_eq, rate=sr4))
```

Audio setelah Equalizer (High-pass 100 Hz & Low-pass 6000 Hz):



```
In [ ]: # --- Fungsi High-pass Filter ---
def highpass_filter(data, sr, cutoff=100, order=4):
    nyquist = 0.5 * sr
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return filtfilt(b, a, data)

# --- Fungsi Low-pass Filter ---
def lowpass_filter(data, sr, cutoff=3000, order=4):
    nyquist = 0.5 * sr
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
```

```

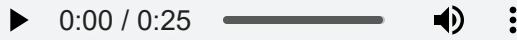
    return filtfilt(b, a, data)

# --- Tahap Equalizer: High-pass lalu Low-pass ---
y4_eq = highpass_filter(y4, sr4, cutoff=100)
y4_eq = lowpass_filter(y4_eq, sr4, cutoff=3000)

# --- Audio Player ---
print("Audio setelah Equalizer (High-pass 100 Hz & Low-pass 6000 Hz):")
display(Audio(y4_eq, rate=sr4))

```

Audio setelah Equalizer (High-pass 100 Hz & Low-pass 6000 Hz):



Gain 10dB

```

In [ ]: # --- Tahap Gain/Fade ---

# Fungsi untuk menerapkan gain dalam dB
def apply_gain(data, gain_db):
    factor = 10 ** (gain_db / 20) # Konversi dB ke faktor Linear
    return data * factor

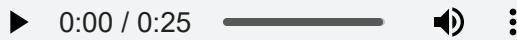
# Contoh: Gain +6 dB
gain_db = 10
y4_gain = apply_gain(y4_eq, gain_db)

# Pastikan tidak clipping (normalisasi cepat jika perlu)
max_val = np.max(np.abs(y4_gain))
if max_val > 1.0:
    y4_gain = y4_gain / max_val # Hindari distorsi digital

print(f"Gain diterapkan: {gain_db} dB")
print(f"Peak amplitude setelah gain: {np.max(np.abs(y4_gain)):.3f}")
display(Audio(y4_gain, rate=sr4))

```

Gain diterapkan: 10 dB
Peak amplitude setelah gain: 1.000



Normalisasi ke -0 dBFS

```

In [ ]: # --- Tahap Normalisasi ke -0 dBFS ---

def normalize_audio(data, target_dbfs=0):
    """
    Normalisasi audio ke target dBFS (default 0 dBFS = max amplitude 1.0)
    """
    peak = np.max(np.abs(data))
    if peak == 0:
        return data # Hindari pembagian nol

```

```

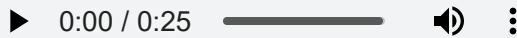
    target_linear = 10 ** (target_dbfs / 20) # 0 dBFS = 1.0
    normalized = data * (target_linear / peak)
    return normalized

y4_norm = normalize_audio(y4_gain, target_dbfs=0)

print("Normalisasi ke -0 dBFS selesai.")
print(f"Peak amplitude setelah normalisasi: {np.max(np.abs(y4_norm)):.3f}")
display(Audio(y4_norm, rate=sr4))

```

Normalisasi ke -0 dBFS selesai.
Peak amplitude setelah normalisasi: 1.000



Compression untuk menyeimbangkan dinamika audio

```

In [ ]: # --- Tahap Compression (Versi Simple Compressor) ---

def simple_compressor(y, threshold=1, ratio=4):
    """
    Kompresi sederhana:
    - Sinyal di bawah threshold: tidak berubah
    - Sinyal di atas threshold: dikecilkan dengan rasio tertentu
    """
    y_compressed = np.copy(y)
    mask = np.abs(y) > threshold
    y_compressed[mask] = np.sign(y[mask]) * (threshold + (np.abs(y[mask]) - threshold) / ratio)
    return y_compressed

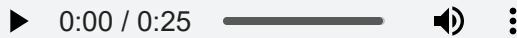
# Terapkan kompresi pada sinyal yang sudah dinormalisasi
y_comp = simple_compressor(y_norm, threshold=1, ratio=4)

print("☑️ Compression applied (Threshold=0.1, Ratio=4:1)")

# Audio Player setelah kompresi
print("Audio Player Setelah Kompresi:")
display(Audio(y_comp, rate=sr4))

```

☑️ Compression applied (Threshold=0.1, Ratio=4:1)
Audio Player Setelah Kompresi:



Noise Gate

```

In [ ]: # --- Tahap Noise Gate ---

def noise_gate(audio, threshold=0.0001):
    """
    Noise gate sederhana: jika amplitudo < threshold → di-mute.
    """

```

```

gated = np.where(np.abs(audio) < threshold, 0, audio)
return gated

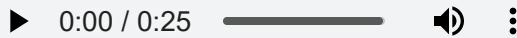
# Terapkan noise gate pada hasil compression
y4_gated = noise_gate(y4_compressed, threshold=0.0001)

print("Noise Gate selesai.")
print(f"Peak amplitude setelah noise gate: {np.max(np.abs(y4_gated)):.3f}")
display(Audio(y4_gated, rate=sr4))

```

Noise Gate selesai.

Peak amplitude setelah noise gate: 0.562



Silence Trimming

In []: # --- Tahap Silence Trimming ---

```

# Trim bagian awal dan akhir yang silence
y_trimmed, index = librosa.effects.trim(y_comp, top_db=30) # top_db 30 = ambang ke

print(f"✓ Silence trimming selesai. Panjang awal: {len(y_comp)}, panjang setelah trim: {len(y_trimmed)}")
print(f"Indeks trimming: start={index[0]}, end={index[1]}")

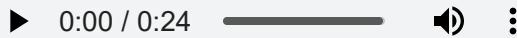
# Audio Player setelah trimming
print("Audio Player Setelah Silence Trimming:")
display(Audio(y_trimmed, rate=sr4))

```

✓ Silence trimming selesai. Panjang awal: 1231488, panjang setelah trim: 1197056 s
ampel

Indeks trimming: start=28672, end=1225728

Audio Player Setelah Silence Trimming:



Normalisasi Loudness ke -16 LUFS

In []: # --- Tahap Loudness Normalization ke -16 LUFS ---

```

# Buat meter untuk menghitung Loudness
meter = pyln.Meter(sr4) # menggunakan sample rate asli

# Hitung Loudness rekaman setelah trimming
loudness_before = meter.integrated_loudness(y_trimmed)

# Tentukan target Loudness
target_loudness = -16.0 # LUFS

# Normalisasi ke target LUFS
y_loudnorm = pyln.normalize.loudness(y_trimmed, loudness_before, target_loudness)

```

```

# Hitung ulang untuk memastikan
loudness_after = meter.integrated_loudness(y_loudnorm)

print(f"LOUDNESS sebelum normalisasi: {loudness_before:.2f} LUFS")
print(f"LOUDNESS setelah normalisasi: {loudness_after:.2f} LUFS")

# Audio Player setelah normalisasi Loudness
print("Audio Player Setelah Normalisasi Loudness:")
display(Audio(y_loudnorm, rate=sr4))

```

Loudness sebelum normalisasi: -18.26 LUFS

Loudness setelah normalisasi: -16.00 LUFS

Audio Player Setelah Normalisasi Loudness:

```

C:\Users\ACER\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra
8p0\LocalCache\local-packages\Python312\site-packages\pyloudnorm\normalize.py:62: Us
erWarning: Possible clipped samples in output.
    warnings.warn("Possible clipped samples in output.")

```

Visualisasi Waveform dan Spectrogram sebelum dan sesudah pemrosesan

```

In [ ]: # --- VISUALISASI WAVEFORM & SPECTROGRAM: SEBELUM vs SESUDAH PEMROSESAN ---

# Waveform Sebelum dan Sesudah
plt.figure(figsize=(14, 6))

# Waveform Sebelum
plt.subplot(2, 1, 1)
plt.plot(np.linspace(0, len(y4)/sr4, len(y4)), y4, color='steelblue')
plt.title('Waveform Sebelum Pemrosesan')
plt.xlabel('Waktu (detik)')
plt.ylabel('Amplitudo')
plt.grid(True, which='both', linestyle='--', alpha=0.5)

# Waveform Sesudah
plt.subplot(2, 1, 2)
plt.plot(np.linspace(0, len(y_loudnorm)/sr4, len(y_loudnorm)), y_loudnorm, color='orange')
plt.title('Waveform Sesudah Pemrosesan')
plt.xlabel('Waktu (detik)')
plt.ylabel('Amplitudo')
plt.grid(True, which='both', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# --- Spectrogram Sebelum dan Sesudah ---
D_before = librosa.amplitude_to_db(np.abs(librosa.stft(y4)), ref=np.max)
D_after = librosa.amplitude_to_db(np.abs(librosa.stft(y_loudnorm)), ref=np.max)

plt.figure(figsize=(14, 6))

# Spectrogram Sebelum

```

```

plt.subplot(2, 1, 1)
librosa.display.specshow(D_before, sr=sr4, x_axis='time', y_axis='hz')
plt.colorbar(format='%.2f dB')
plt.title('Spectrogram Sebelum Pemrosesan')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')

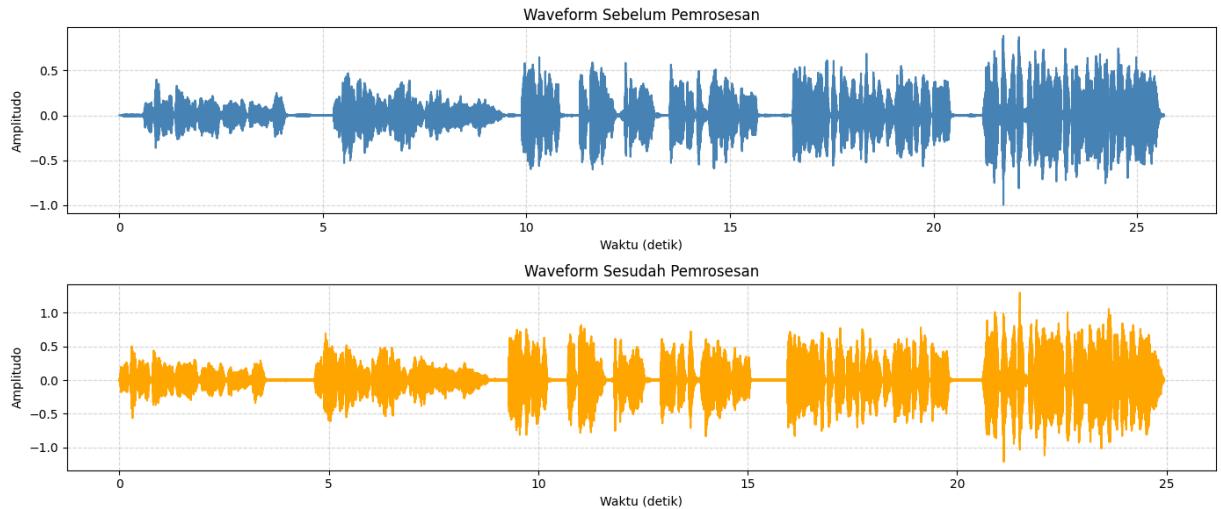
# Spectrogram Sesudah
plt.subplot(2, 1, 2)
librosa.display.specshow(D_after, sr=sr4, x_axis='time', y_axis='hz')
plt.colorbar(format='%.2f dB')
plt.title('Spectrogram Sesudah Pemrosesan')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')

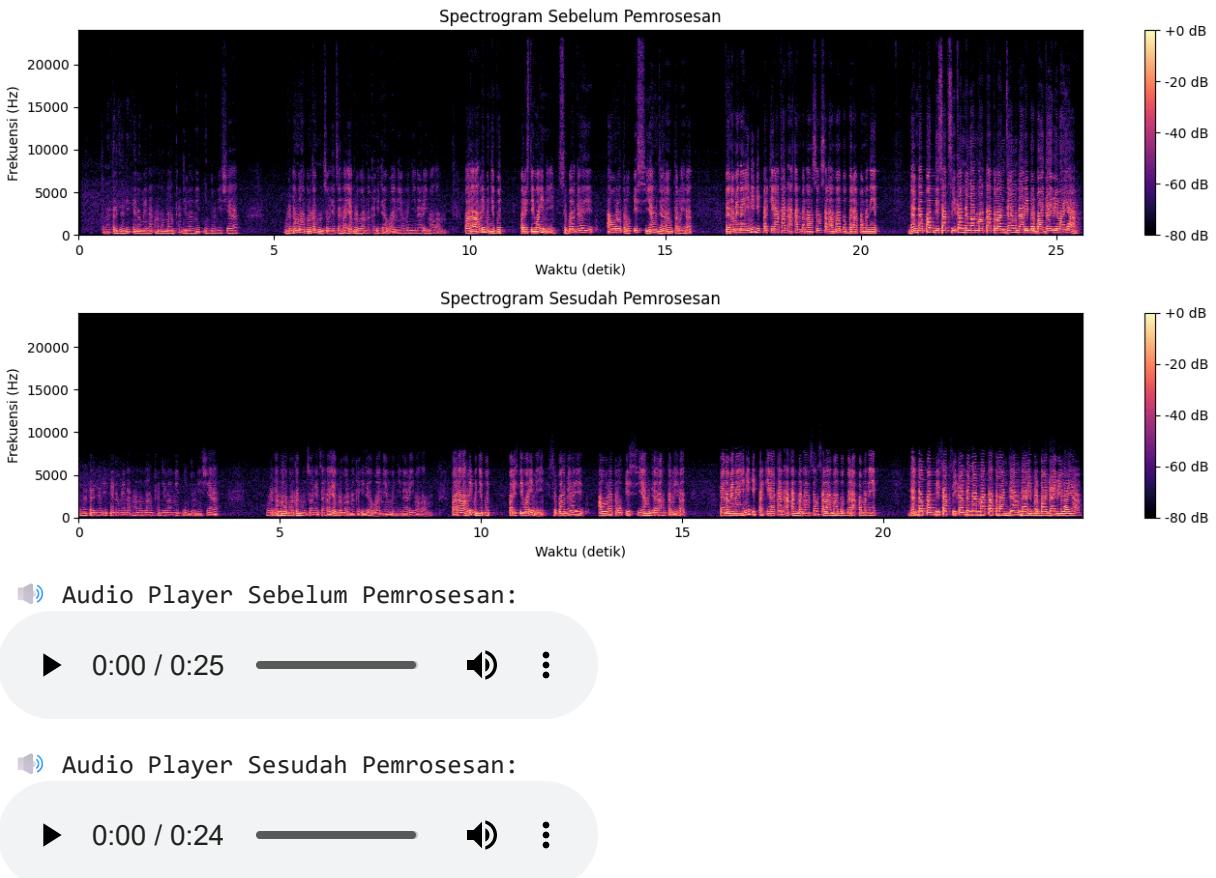
plt.tight_layout()
plt.show()

# --- Perbandingan Audio Player ---
print("🔊 Audio Player Sebelum Pemrosesan:")
display(Audio(y4, rate=sr4))

print("🔊 Audio Player Sesudah Pemrosesan:")
display(Audio(y_loudnorm, rate=sr4))

```





Penjelasan

- Perubahan yang Terjadi pada Audio

Berdasarkan perbedaan visual antara kedua spectrogram, beberapa proses pengolahan audio telah diterapkan:

- **Equalizer (Filter)**: Menghilangkan frekuensi di atas ~ 7500 Hz (area atas pada spectrogram kedua menjadi hitam) untuk menghilangkan **noise desis** frekuensi tinggi. Pengurangan energi di frekuensi sangat rendah (di bawah ~ 100 Hz) untuk menghilangkan **suara rumble** yang tidak diinginkan.
- **Noise Gate**: Bagian-bagian yang seharusnya hening (di antara ucapan) pada spectrogram kedua terlihat jauh lebih gelap dan bersih. Ini menunjukkan bahwa noise latar belakang dengan volume rendah telah dihilangkan.
- **Compression & Normalization**: **kompresi** digunakan untuk mengurangi rentang dinamis, dan **normalisasi** digunakan untuk menaikkan volume keseluruhan ke level yang optimal.

- Perbedaan Antara Normalisasi Peak dan Normalisasi LUFS

- Normalisasi peak menyesuaikan level tertinggi sinyal agar tidak melebihi batas, mencegah distorsi tanpa mempertimbangkan persepsi keras–lembut suara. Sedangkan

normalisasi LUFS mengatur rata-rata kenyaringan berdasarkan persepsi pendengaran, sehingga hasil terdengar konsisten antar file meski nilai puncaknya berbeda.

- Bagaimana Kualitas Suara Berubah Setelah Proses

- Setelah normalisasi LUFS dan optimasi loudness, volume audio menjadi lebih stabil, jelas, dan terdengar profesional. Namun, kompresi berlebihan bisa mengurangi dinamika alami sehingga suara terasa datar.

- Kelebihan dan Kekurangan Pengoptimalan Loudness

- Kelebihannya adalah konsistensi volume antar rekaman, kenyamanan dengar di berbagai perangkat, dan kesesuaian dengan standar platform. Kekurangannya meliputi hilangnya dinamika, potensi distorsi atau clipping, serta suara yang bisa terdengar datar dan melelahkan.

Soal 5: Music Analysis dan Remix

Pilih 2 buah (potongan) lagu yang memiliki vokal (penyanyi) dan berdurasi sekitar 1 menit: –
Lagu 1: Nuansa sedih, lambat – Lagu 2: Nuansa ceria, cepat

Lakukan deteksi tempo (BPM) dan estimasi kunci (key) dari masing-masing lagu dan berikan analisis singkat.

Lakukan remix terhadap kedua lagu: – Time Stretch – Pitch Shift – Crossfading – Filter Tambahan

```
In [ ]: # =====#
# Load Audio, Deteksi Tempo & Key
# =====#

# Daftar nada untuk estimasi key
notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']

# Fungsi untuk Load audio, deteksi tempo & estimasi key
def load_analyze_audio(path):
    if os.path.exists(path):
        y, sr = librosa.load(path, sr=None)
        print(f"File ditemukan: {path}")
        tempo, _ = librosa.beat.beat_track(y=y, sr=sr)
        tempo = tempo[0] if isinstance(tempo, np.ndarray) else tempo
        chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
        key = notes[np.argmax(chroma.mean(axis=1))]
        display(Audio(y, rate=sr))
        return y, sr, tempo, key
    else:
        print(f"File tidak ditemukan: {path}")
        return None, None, None, None
```

```
# Path Audio
path_happy = os.path.join(os.getcwd(), 'data', 'dj_1.wav')
path_sad   = os.path.join(os.getcwd(), 'data', 'lagu_sedih2.wav')

# Proses Happy
y5, sr5, tempo_happy, key_happy = load_analyze_audio(path_happy)
print(f"\ud83c\udc81 Tempo Happy: {tempo_happy:.2f} BPM, Key: {key_happy}")

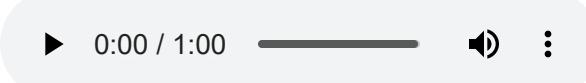
# Proses Sad
y6, sr6, tempo_sad, key_sad = load_analyze_audio(path_sad)
print(f"\ud83c\udc81 Tempo Sad: {tempo_sad:.2f} BPM, Key: {key_sad}")
```

File ditemukan: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\data\dj_1.wav



 Tempo Happy: 109.96 BPM, Key: F

File ditemukan: d:\00 kuliah s7\multimedia\hands-on pemrosesan audio\data\lagu_sedih 2.wav



 Tempo Sad: 123.05 BPM, Key: G#

Penjelasan

Tempo lagu "Sad" adalah 123,05 BPM dengan key G#, sedangkan lagu "Happy" memiliki tempo 109,96 BPM dengan key F. Untuk menyamakan tonalitas, lagu "Sad" perlu dipitch shift sebanyak -3 semitone agar sesuai dengan key "Happy". Perbedaan BPM ini menunjukkan bahwa meskipun lagu "Sad" terdengar lebih lambat secara subjektif, sistem deteksi tempo mendeteksi nilainya lebih tinggi, sehingga perlu penyesuaian tempo atau pitch agar kedua lagu terdengar harmonis ketika dibandingkan.

Time Stretch

```
In [ ]: # Sesuaikan tempo Lagu Sad agar mendekati tempo Happy (misalnya)
stretch_factor = tempo_happy / tempo_sad
y6_stretched = librosa.effects.time_stretch(y6, rate=stretch_factor)

print(f"⌚ Stretch Factor Sad → Happy Tempo: {stretch_factor:.2f}")

# Waveform perbandingan Sad vs Sad Stretched
plt.figure(figsize=(14, 5))
plt.subplot(2, 1, 1)
plt.plot(np.linspace(0, len(y6)/sr6, len(y6)), y6, color='steelblue')
plt.title('Waveform Lagu Sad - Original')
plt.xlabel('Waktu (detik)')
plt.ylabel('Amplitudo')

plt.subplot(2, 1, 2)
plt.plot(np.linspace(0, len(y6_stretched)/sr6, len(y6_stretched)), y6_stretched, color='red')
plt.title('Waveform Lagu Sad - Time Stretched')
```

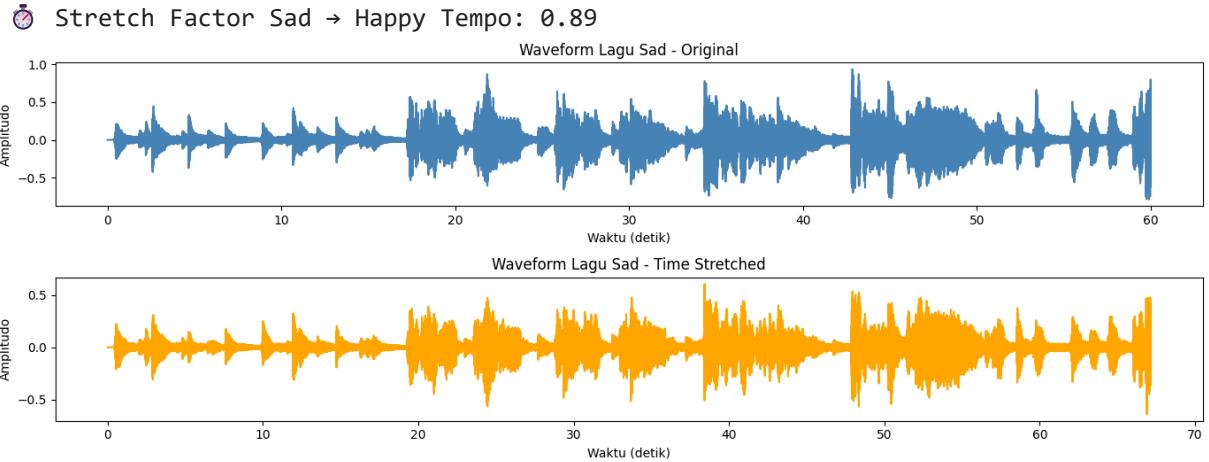
```

plt.xlabel('Waktu (detik)')
plt.ylabel('Amplitudo')

plt.tight_layout()
plt.show()

print("🔊 Audio Sad - Original:")
display(Audio(y6, rate=sr6))
print("🔊 Audio Sad - Time Stretched:")
display(Audio(y6_stretched, rate=sr6))

```



🔊 Audio Sad - Original:

▶ 0:00 / 1:00 ━━ 🔊 ⋮

🔊 Audio Sad - Time Stretched:

▶ 0:00 / 1:07 ━━ 🔊 ⋮

Penjelasan

Lagu sad dijadikan BPM yang sama dengan lagu Happy yang menjadikan lagu sad lebih lama(streched lebih panjang), hal ini menyebabkan kenaikan pada frekuensi Hz.

Pitch Shift

```

In [ ]: # Misalnya: Naikkan pitch Lagu Sad agar cocok dengan key Happy (perbedaan semiton)
notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
key_index_happy = notes.index(key_happy)
key_index_sad = notes.index(key_sad)
semitone_shift = key_index_happy - key_index_sad

y6_pitchshifted = librosa.effects.pitch_shift(y6_stretched, sr=sr6, n_steps=semiton

# Spectrogram perbandingan
D_before = librosa.amplitude_to_db(np.abs(librosa.stft(y6_stretched))), ref=np.max)
D_after = librosa.amplitude_to_db(np.abs(librosa.stft(y6_pitchshifted))), ref=np.max

```

```

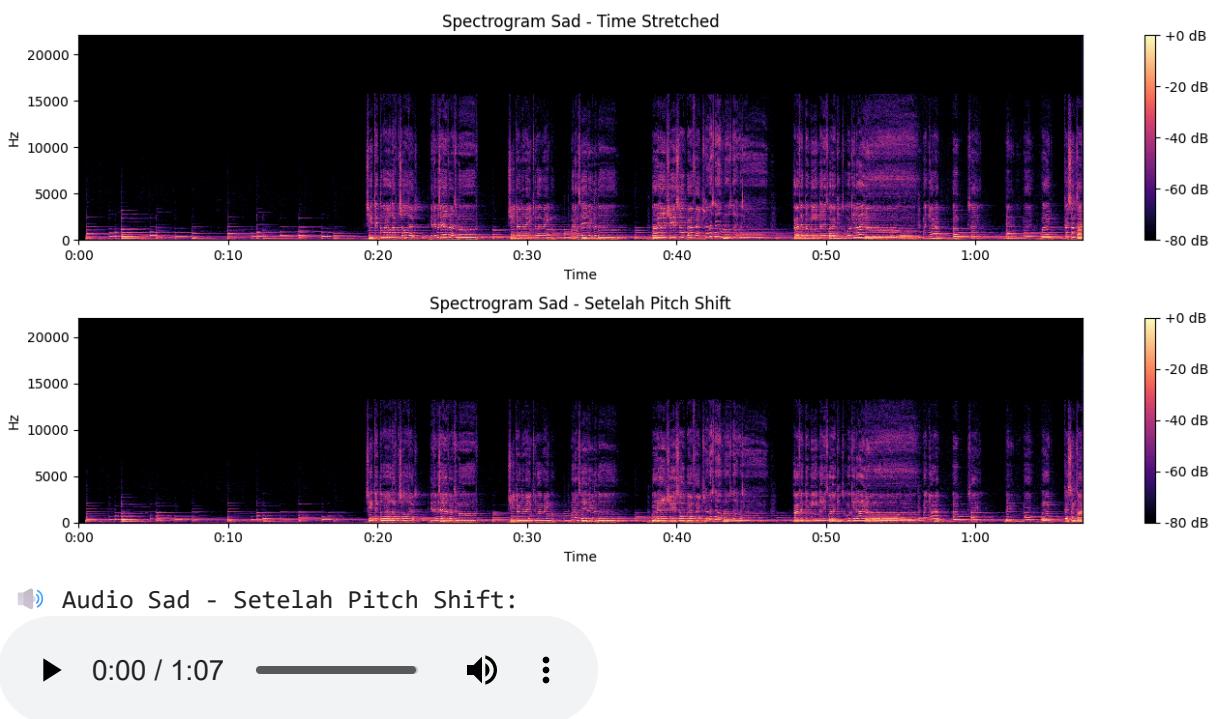
plt.figure(figsize=(14, 6))
plt.subplot(2, 1, 1)
librosa.display.specshow(D_before, sr=sr6, x_axis='time', y_axis='hz')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram Sad - Time Stretched')

plt.subplot(2, 1, 2)
librosa.display.specshow(D_after, sr=sr6, x_axis='time', y_axis='hz')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram Sad - Setelah Pitch Shift')

plt.tight_layout()
plt.show()

print("🔊 Audio Sad - Setelah Pitch Shift:")
display(Audio(y6_pitchshifted, rate=sr6))

```



Penjelasan

Karena BPM lagu sedih yang tedeteksi lebih besar dari lagu happy, maka yang terjadi adalah penurunan frekuensi penurunan pitch juga.

Crossfading

```

In [ ]: # Tentukan durasi potongan lagu pertama (Happy)
durasi_happy_awal = 60 # detik
happy_samples = int(durasi_happy_awal * sr5)
y5_cut = y5[:happy_samples]

# Pastikan Sad (yang sudah diproses) cukup panjang
if len(y6_pitchshifted) < sr5 * 5:
    raise ValueError("Lagu Sad terlalu pendek untuk crossfade 5 detik.")

```

```

# Durasi crossfade (detik)
fade_dur = 5
fade_samples = int(fade_dur * sr5)

# Ambil awal lagu Sad dan siapkan crossfade
y6_intro = y6_pitchshifted[:fade_samples]
y5_outro = y5_cut[-fade_samples:]

# Buat fade-out untuk akhir Lagu Happy dan fade-in untuk awal Lagu Sad
fade_out = np.linspace(1, 0, fade_samples)
fade_in = np.linspace(0, 1, fade_samples)

y5_outro_faded = y5_outro * fade_out
y6_intro_faded = y6_intro * fade_in

# Gabungkan:
# - Happy hingga sebelum crossfade
# - crossfade antara akhir Happy & awal Sad
# - sisa Sad setelah crossfade
y_remix = np.concatenate([
    y5_cut[:-fade_samples],           # Happy bagian awal
    y5_outro_faded + y6_intro_faded, # Crossfade 5 detik
    y6_pitchshifted[fade_samples:]   # Sisa Sad
])

# --- Visualisasi Waveform Happy, Sad, dan Crossfade ---
fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

# Waveform Happy
time_happy = np.linspace(0, len(y5_cut)/sr5, len(y5_cut))
axes[0].plot(time_happy, y5_cut, color='blue')
axes[0].set_title('Waveform Lagu Happy (Sebelum Crossfade)', fontweight='bold')
axes[0].set_ylabel('Amplitudo')
axes[0].grid(True, linestyle='--', alpha=0.5)

# Waveform Sad
time_sad = np.linspace(0, len(y6_pitchshifted)/sr5, len(y6_pitchshifted))
axes[1].plot(time_sad, y6_pitchshifted, color='green')
axes[1].set_title('Waveform Lagu Sad (Sebelum Crossfade)', fontweight='bold')
axes[1].set_ylabel('Amplitudo')
axes[1].grid(True, linestyle='--', alpha=0.5)

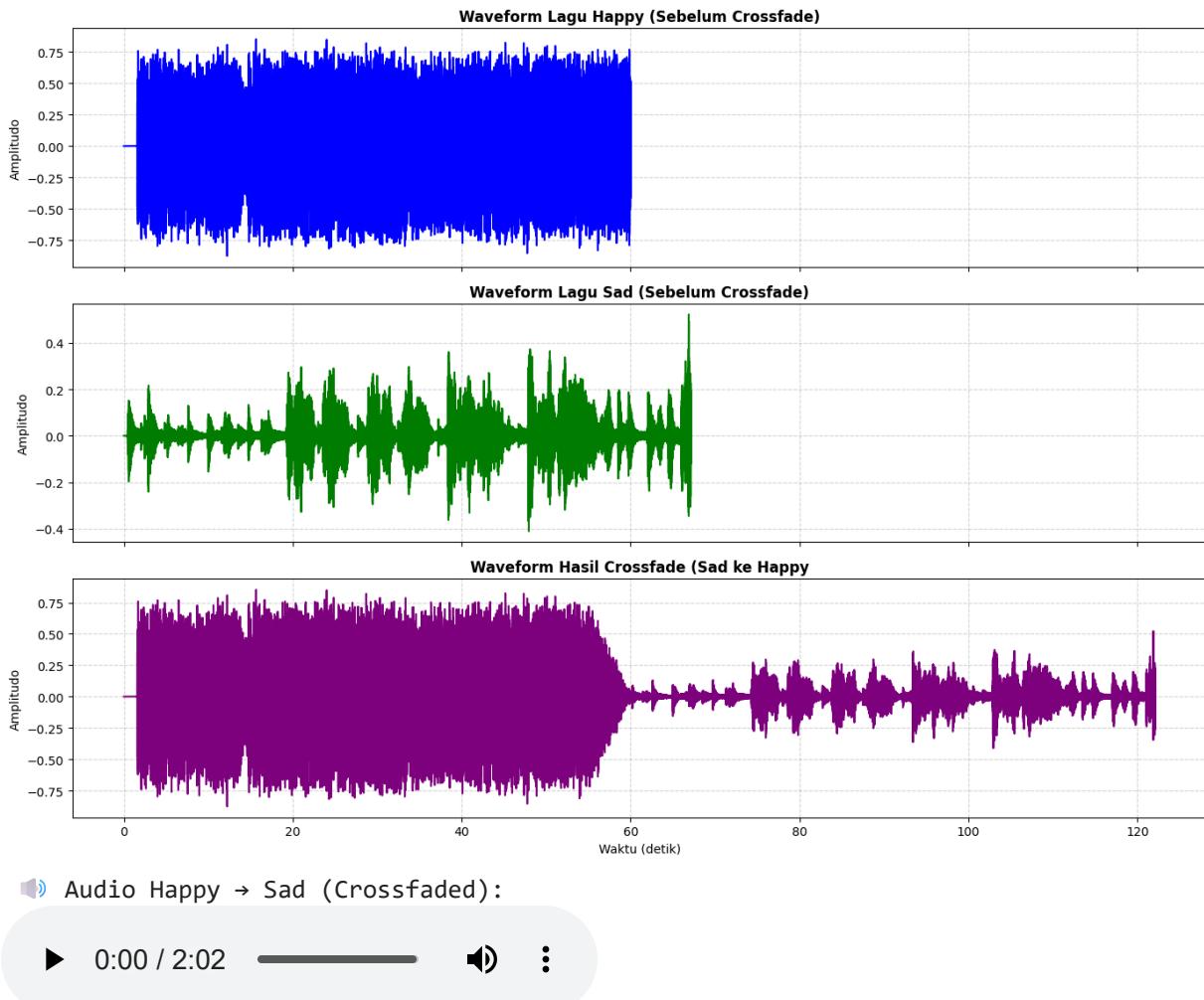
# Waveform Hasil Crossfade
time_remix = np.linspace(0, len(y_remix)/sr5, len(y_remix))
axes[2].plot(time_remix, y_remix, color='purple')
axes[2].set_title('Waveform Hasil Crossfade (Sad ke Happy)', fontweight='bold')
axes[2].set_xlabel('Waktu (detik)')
axes[2].set_ylabel('Amplitudo')
axes[2].grid(True, linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# --- Audio Player ---

```

```
print("🔊 Audio Happy → Sad (Crossfaded):")
display(Audio(y_remix, rate=sr5))
```



Penjelasan:

1. Perubahan durasi audio setelah time-stretching:

Lagu sedih memiliki BPM yang lebih besar, time stretch otomatis memanjangkan lagu sedih yang menyebabkan suara menjadi lebih bass.

2. Perubahan pitch audio setelah pitch-shifting:

Berdasarkan representasi not music dalam semitone, Lagu Happy yang berada di kunci F, sedangkan Lagu Sad yang berada di kunci G Minor. Saya tidak mengerti mengapa lagu happy terdeteksi memiliki BPM yang lebih kecil daripada lagu sedih yang saya pilih, karena ketika didengar dengan telinga lagu happy terdengar memiliki BPM yang lebih besar.

3. Efek crossfading pada transisi antara dua lagu:

Crossfading hanya digunakan untuk transisi antara lagu "Happy" dan "Sad". Bagian akhir lagu "Happy" secara bertahap memudar (fade out) sementara bagian awal lagu

"Sad" secara bertahap muncul (fade in).

Credit dan Referensi

Materi Pembelajaran

1. [Audio Processing Week 3](#)

Bantuan AI (Chat GPT)

1. [Chat GPT](#)

Referensi Berita

- [Berita Tribun](#)

Link Music yang dipakai

1. [DJ SAMPANG BANJIR POLE](#)
2. [Joji - Glimpse of us](#)