

# Worksheet Audio

Nama : M. Arief Rahman Hakim

NIM : 122140083

## Soal 1 : Rekaman dan Analisis Suara Multi-Level

Rekamlah suara Anda sendiri selama 25 detik dimana Anda membaca sebuah teks berita.

- 5 detik pertama: suara sangat pelan dan berbisik
- 5 detik kedua: suara normal
- 5 detik ketiga: suara keras
- 5 detik keempat: suara cempreng (dibuat-buat cempreng)
- 5 detik terakhir: suara berteriak

Rekam dalam format WAV (atau konversikan ke WAV sebelum dimuat ke notebook).

Dalam 25 detik rekaman tersebut, Anda harus merekam:

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 [13]: # Import audio
from IPython.display import Audio
import os

# Ambil path audio
PATH_AUDIO_SOAL1 = os.path.join(os.getcwd(), 'data', 'Suara Mahal.wav')

# Play audio
Audio(PATH_AUDIO_SOAL1)
```

Out[13]:

Teks nya berasal dari generate GEMINI A, sebagai berikut:

Teks Berita:

(0-5 detik: suara sangat pelan dan berbisik)

"Pemirsa... sebuah terobosan baru dalam dunia teknologi baru saja diumumkan..."

(6-10 detik: suara normal)

"Para ilmuwan berhasil mengembangkan baterai inovatif yang dapat terisi penuh hanya dalam waktu satu menit."

(11-15 detik: suara keras)

"PENEMUAN INI DIHARAPKAN DAPAT MEREVOLUSI INDUSTRI PERANGKAT ELEKTRONIK GLOBAL!"

(16-20 detik: suara cempreng/dibuat-buat)

"Bayangkaaan, nge-charge hape nggak perlu nunggu lama-lama lagi, asyiiiik!"

(21-25 detik: suara berteriak)

"INI ADALAH MASA DEPAN, DAN MASA DEPAN ITU SEKARANG!"

```
In [16]: # Visualisasi path audio 1 waveform dan spectrogram
import matplotlib.pyplot as plt
import numpy as np
import librosa

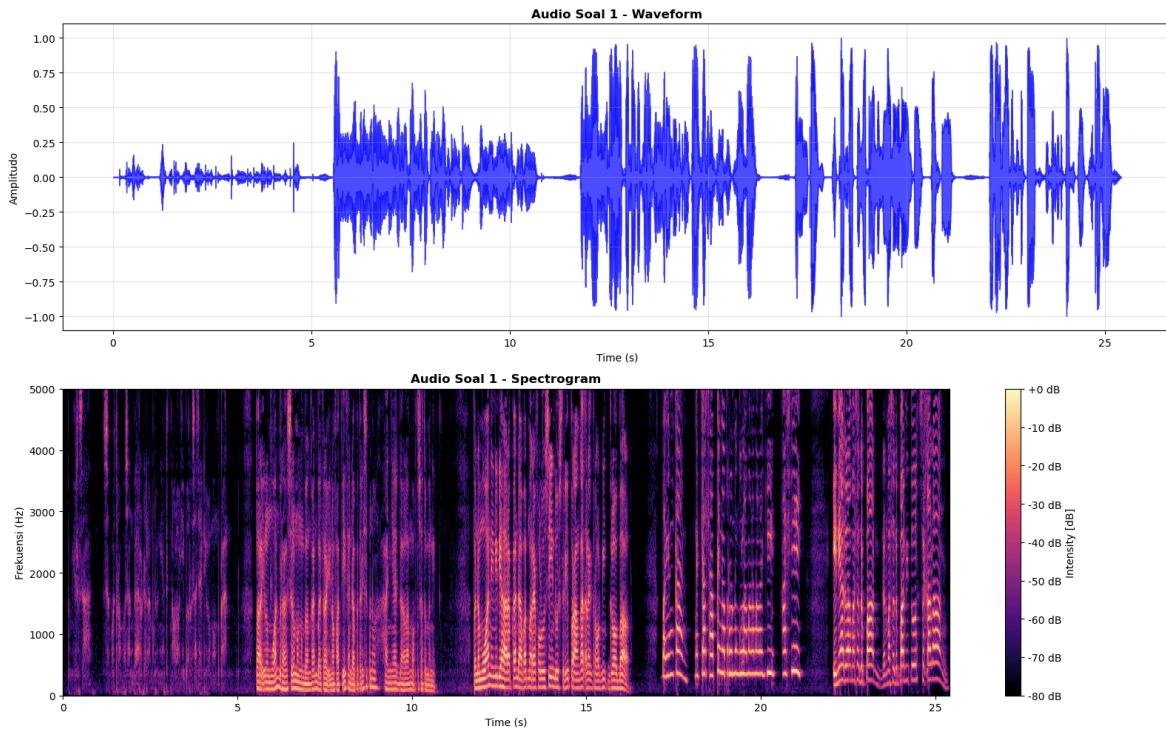
# Load audio
y, sr = librosa.load(PATH_AUDIO_SOAL1, sr=None)

# Create figure with subplots (2 rows, 1 column)
fig, axes = plt.subplots(2, 1, figsize=(16, 10))

# Waveform
librosa.display.waveshow(y, sr=sr, ax=axes[0], color='blue', alpha=0.7)
axes[0].set_title('Audio Soal 1 - Waveform', fontweight='bold')
axes[0].set_ylabel('Amplitudo')
axes[0].set_xlabel('Time (s)')
axes[0].grid(True, alpha=0.3)

# Spectrogram
S = librosa.stft(y)
S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
img = librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', ax=axes[1])
axes[1].set_title('Audio Soal 1 - Spectrogram', fontweight='bold')
axes[1].set_ylabel('Frekuensi (Hz)')
axes[1].set_xlabel('Time (s)')
axes[1].set_ylim(0, 5000)
plt.colorbar(img, ax=axes[1], format='%.2f dB', label='Intensity [dB]')

plt.tight_layout()
plt.show()
```



Berdasarkan hasil waveform dan spektrogram, terlihat bahwa rekaman memiliki variasi intensitas suara yang signifikan dari awal hingga akhir. Pada detik 0–5, amplitudo sangat kecil menandakan suara pelan dan berbisik. Memasuki 6–10 detik, amplitudo mulai meningkat dengan pola teratur yang menunjukkan suara normal dan stabil dengan frekuensi dari 0-2000. Selanjutnya pada 11–15 detik, amplitudo mencapai puncak dengan kerapatan tinggi meski kerapatan tersebut masih berasa di tengah sampai ke bawah jika dilihat dari frekuensinya namun tetap ada juga yang sampai di atas, menggambarkan suara keras dan penuh penekanan.

Pada segmen 16–20 detik, bentuk gelombang menjadi lebih acak dengan fluktuasi cepat, mencerminkan suara cempreng atau dibuat-buat dengan dominasi frekuensi tinggi. Akhirnya, pada 21–25 detik, amplitudo berada pada level maksimum dan warna spektrogram tampak paling terang di seluruh rentang frekuensi, menandakan suara berteriak dengan energi sangat besar.

In [17]:

```
import librosa
import numpy as np
from IPython.display import Audio

# Melakukan resampling audio ke 16kHz
import matplotlib.pyplot as plt

print("RESAMPLING DEMONSTRATION")
print("=" * 50)

# Load audio asli dari PATH_AUDIO_SOAL1
y_original, sr_original = librosa.load(PATH_AUDIO_SOAL1, sr=None)

print(f"Audio asli: {sr_original:,} Hz, {len(y_original):,} samples")
print(f"Durasi: {len(y_original)/sr_original:.2f} detik")

# Target sample rate untuk downsampling
target_sr = 16000
```

```

# Resampling menggunakan librosa (dengan anti-aliasing)
y_resampled = librosa.resample(y_original, orig_sr=sr_original, target_sr=target_sr)

# Hitung rasio resampling
resample_ratio = target_sr / sr_original

print(f"Target sample rate: {target_sr:,} Hz")
print(f"Ratio downampling: {resample_ratio:.4f}")
print()
print("HASIL RESAMPLING:")
print(f"Original: {len(y_original):,} samples")
print(f"Resampled: {len(y_resampled):,} samples")
print(f"Durasi: {len(y_resampled)/target_sr:.2f} detik")

# Visualisasi perbandingan
fig, axes = plt.subplots(2, 1, figsize=(16, 10))

# Plot 1: Audio asli
librosa.display.waveshow(y_original, sr=sr_original, ax=axes[0], color='blue', alpha=0.3)
axes[0].set_title(f'Audio Asli ({sr_original:,} Hz)', fontsize=12, fontweight='bold')
axes[0].set_ylabel('Amplitudo')
axes[0].set_xlabel('Waktu (detik)')
axes[0].grid(True, alpha=0.3)

# Plot 2: Audio resampled
librosa.display.waveshow(y_resampled, sr=target_sr, ax=axes[1], color='red', alpha=0.3)
axes[1].set_title(f'Audio Resampled ({target_sr:,} Hz)', fontsize=12, fontweight='bold')
axes[1].set_ylabel('Amplitudo')
axes[1].set_xlabel('Waktu (detik)')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Audio players untuk perbandingan
print("\nAUDIO COMPARISON:")
print("Original Audio:")
display(Audio(y_original, rate=sr_original))

print("\nResampled Audio (16 kHz):")
display(Audio(y_resampled, rate=target_sr))

print(f"\nResampling selesai! Audio tersimpan dengan sample rate {target_sr:,} Hz")

```

## RESAMPLING DEMONSTRATION

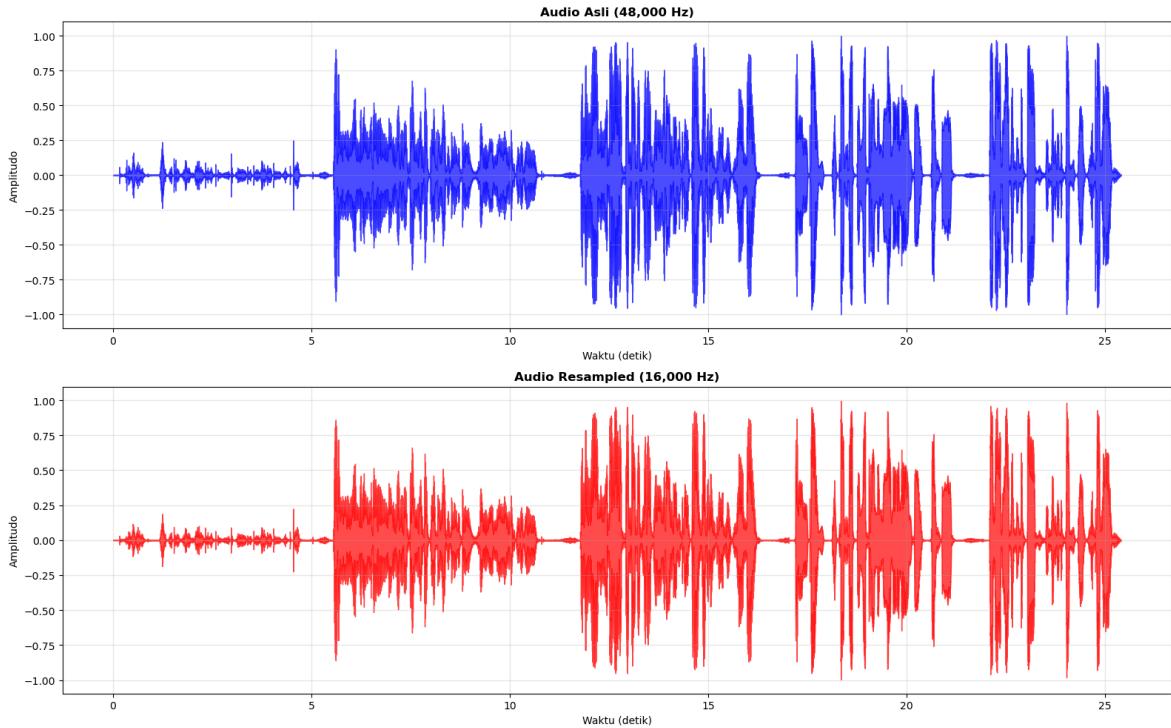
```

=====
Audio asli: 48,000 Hz, 1,219,584 samples
Durasi: 25.41 detik
Target sample rate: 16,000 Hz
Ratio downsampling: 0.3333

```

HASIL RESAMPLING:

Original: 1,219,584 samples  
 Resampled: 406,528 samples  
 Durasi: 25.41 detik

**AUDIO COMPARISON:**

Original Audio:

▶ 0:00 / 0:25 ━━ 🔊 ⋮

Resampled Audio (16 kHz):

▶ 0:00 / 0:25 ━━ 🔊 ⋮

Resampling selesai! Audio tersimpan dengan sample rate 16,000 Hz

## Soal 2: Noise Reduction dengan Filtering

- Rekaman tersebut harus berdurasi kurang lebih 10 detik.
- Rekam dalam format WAV (atau konversikan ke WAV sebelum dimuat ke notebook).

Gunakan filter equalisasi (high-pass, low-pass, dan band-pass) untuk menghilangkan noise pada rekaman tersebut.

Rekam suara Anda berbicara di sekitar objek yang berisik (seperti kipas angin, AC, atau mesin).

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 [ ]: # Ambil path audio  
PATH_AUDIO = os.path.join(os.getcwd(), 'data', 'Rekaman Depan Kipas Angin.wav')  
  
# Play audio  
Audio(PATH_AUDIO)
```

Out[ ]: ▶ 0:00 / 0:11 ⏸ ⏴

```
In [11]: import matplotlib.pyplot as plt  
from scipy.signal import spectrogram  
from scipy.io import wavfile  
from scipy.signal import butter, filtfilt  
import scipy.signal  
import numpy as np  
from IPython.display import Audio  
import librosa  
  
# Sample Rate and Data  
rate, data = wavfile.read(PATH_AUDIO)  
data = data / np.max(np.abs(data)) # Normalisasi data  
  
# Lowpass Filter  
cutoff_low = 3000 # Hz  
filter_order = 4  
nyquist = 0.5 * rate  
normal_cutoff_low = cutoff_low / nyquist  
b_low, a_low = butter(filter_order, normal_cutoff_low, btype='low')  
audio_lowpass = filtfilt(b_low, a_low, data)  
  
# Highpass Filter  
cutoff_high = 150 # Hz  
filter_order = 4  
nyquist = 0.5 * rate  
normal_cutoff_high = cutoff_high / nyquist  
b_high, a_high = butter(filter_order, normal_cutoff_high, btype='high')  
audio_highpass = filtfilt(b_high, a_high, data)  
  
# Parameter Filter untuk Bandpass  
bandpass_low = 150 # Hz  
bandpass_high = 3000 # Hz  
filter_order = 4  
nyquist = 0.5 * rate  
bandpass_normalized = [bandpass_low / nyquist, bandpass_high / nyquist]  
b_band, a_band = scipy.signal.butter(filter_order, bandpass_normalized, btype='b')  
audio_bandpass = scipy.signal.filtfilt(b_band, a_band, data)  
  
# Visualisasi Perbandingan  
import librosa.display  
  
fig, axes = plt.subplots(4, 2, figsize=(16, 16))  
  
# Function untuk plot spectrum menggunakan librosa  
def plot_audio_spectrum(audio, sr, ax_wave, ax_spec, title, color):  
    # Waveform  
    librosa.display.waveshow(audio, sr=sr, ax=ax_wave, color=color, alpha=0.7)
```

```
ax_wave.set_title(f'{title} - Waveform', fontweight='bold')
ax_wave.set_ylabel('Amplitudo')
ax_wave.set_xlabel('Time (s)')
ax_wave.grid(True, alpha=0.3)

# Spectrogram menggunakan Librosa
S = librosa.stft(audio)
S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
img = librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', ax=ax_spec)
ax_spec.set_title(f'{title} - Spectrogram', fontweight='bold')
ax_spec.set_ylabel('Frekuensi (Hz)')
ax_spec.set_xlabel('Time (s)')
ax_spec.set_yticks([0, 5000])
plt.colorbar(img, ax=ax_spec, format='%.2f dB', label='Intensity [dB]')

# Plot semua audio
plot_audio_spectrum(data, rate, axes[0,0], axes[0,1], 'Original', 'blue')
plot_audio_spectrum(audio_lowpass, rate, axes[1,0], axes[1,1], f'Low-pass ({cuto})
plot_audio_spectrum(audio_highpass, rate, axes[2,0], axes[2,1], f'High-pass ({cu
plot_audio_spectrum(audio_bandpass, rate, axes[3,0], axes[3,1], f'Band-pass ({ba

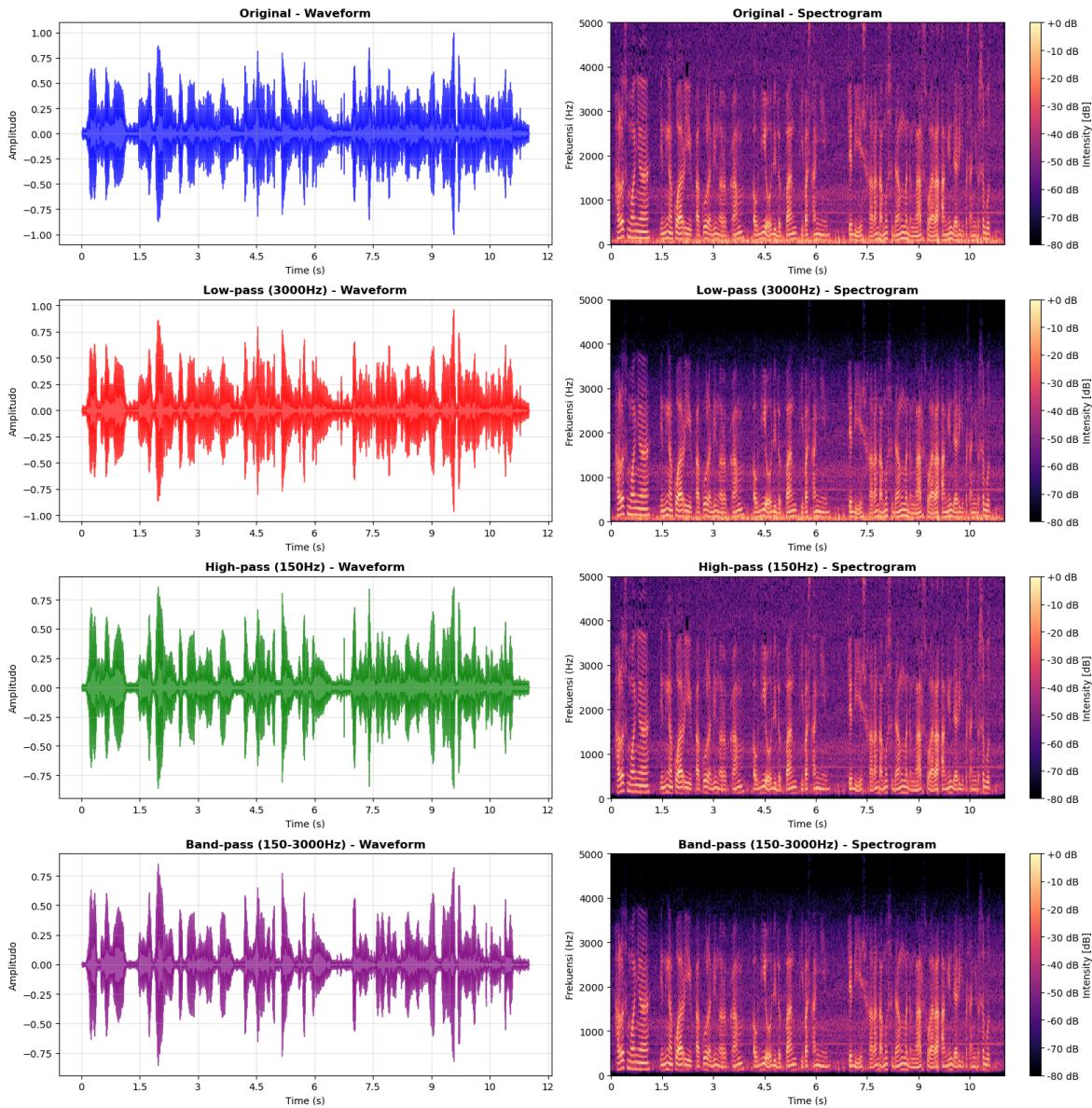
plt.tight_layout()
plt.show()

# Play original and filtered audio
print("Original Audio")
display(Audio(data, rate=rate))

print("Lowpass Filtered Audio")
display(Audio(audio_lowpass, rate=rate))

print("Highpass Filtered Audio")
display(Audio(audio_highpass, rate=rate))

print("Bandpass Filtered Audio")
display(Audio(audio_bandpass, rate=rate))
```

**Original Audio**

▶ 0:00 / 0:11 ⏸ 🔊 ⋮

**Lowpass Filtered Audio**

▶ 0:00 / 0:11 ⏸ 🔊 ⋮

**Highpass Filtered Audio**

▶ 0:00 / 0:11 ⏸ 🔊 ⋮

**Bandpass Filtered Audio**

▶ 0:00 / 0:11 ⏸ 🔊 ⋮

**Penjelasan:**

Jadi saya mencoba 3 filter low, high, band(gabungan) hasilnya menunjukkan bahwa noise yang muncul pada rekaman terutama berasal dari frekuensi rendah (low-frequency hum)

dan sedikit gangguan di frekuensi tinggi (hiss). Setelah diuji dengan ketiga filter, band-pass filter (150–3000 Hz) terbukti paling efektif karena mampu menghilangkan noise di luar rentang frekuensi utama suara manusia, tanpa mengurangi jelasnya suara saya.

Filter low-pass memang berhasil meredam noise di atas 3000 Hz, tetapi membuat suara terdengar agak tertutup, sedangkan high-pass hanya mengurangi noise rendah namun masih menyisakan desis di bagian atas. Dengan cutoff 150 Hz dan 3000 Hz, band-pass filter menghasilkan suara yang lebih jernih, seimbang, dan natural, sehingga ucapan terdengar jelas tanpa kehilangan karakter aslinya.

## 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.

- 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.

Jelaskan proses pitch shifting yang Anda lakukan, termasuk:

Gabungkan kedua rekaman yang telah di-pitch shift ke dalam satu file audio. (Gunakan ChatGPT / AI untuk membantu Anda dalam proses ini)

```
In [23]: # Lakukan pitch shifting pada audio soal 1 untuk membuat suara terdengar seperti chipmunk
import librosa
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import Audio, display

# Load audio (sudah ada dari cell sebelumnya)
y, sr = librosa.load(PATH_AUDIO_SOAL1, sr=None)

# Pitch shifting dengan dua nilai berbeda
n_steps_1 = 7    # Naikkan pitch sebanyak 7 semitone
n_steps_2 = 12   # Naikkan pitch sebanyak 12 semitone

y_pitch_shifted_7 = librosa.effects.pitch_shift(y=y, sr=sr, n_steps=n_steps_1)
y_pitch_shifted_12 = librosa.effects.pitch_shift(y=y, sr=sr, n_steps=n_steps_2)

print("=" * 80)
print("PITCH SHIFTING - CHIPMUNK EFFECT")
print("=" * 80)
print(f"\n Parameter Pitch Shifting:")
print(f"  • Sample Rate: {sr:,} Hz")
print(f"  • Durasi Audio: {len(y)/sr:.2f} detik")
```

```
print(f" • Pitch Shift 1: +{n_steps_1} semitones ")
print(f" • Pitch Shift 2: +{n_steps_2} semitones ")
print()

# Visualisasi perbandingan
fig, axes = plt.subplots(3, 2, figsize=(16, 14))

# Fungsi untuk plot waveform dan spectrogram
def plot_audio_analysis(audio, sr, ax_wave, ax_spec, title, color='blue'):
    # Waveform
    librosa.display.waveplot(audio, sr=sr, ax=ax_wave, color=color, alpha=0.7)
    ax_wave.set_title(f'{title} - Waveform', fontsize=12, fontweight='bold')
    ax_wave.set_ylabel('Amplitudo')
    ax_wave.set_xlabel('Waktu (detik)')
    ax_wave.grid(True, alpha=0.3)

    # Spectrogram
    S = librosa.stft(audio)
    S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
    img = librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', ax=ax_spec)
    ax_spec.set_title(f'{title} - Spectrogram', fontsize=12, fontweight='bold')
    ax_spec.set_ylabel('Frekuensi (Hz)')
    ax_spec.set_xlabel('Waktu (detik)')
    ax_spec.set_ylim(0, 8000)
    plt.colorbar(img, ax=ax_spec, format='%.2f dB', label='Intensity [dB]')

# Plot Audio Asli
plot_audio_analysis(y, sr, axes[0,0], axes[0,1], 'Audio Asli (Original)', 'blue')

# Plot Pitch Shifted +7 semitones
plot_audio_analysis(y_pitch_shifted_7, sr, axes[1,0], axes[1,1],
                    f'Pitch Shifted (+{n_steps_1} semitone)', 'green')

# Plot Pitch Shifted +12 semitones
plot_audio_analysis(y_pitch_shifted_12, sr, axes[2,0], axes[2,1],
                    f'Pitch Shifted (+{n_steps_2} semitone)', 'red')

plt.tight_layout()
plt.show()

# Play audio untuk perbandingan
print("\n🎧 AUDIO COMPARISON:")
print("\n1. Audio Original:")
display(Audio(y, rate=sr))

print(f"\n2. Audio Pitch Shifted (+{n_steps_1} semitones):")
display(Audio(y_pitch_shifted_7, rate=sr))

print(f"\n3. Audio Pitch Shifted (+{n_steps_2} semitones):")
display(Audio(y_pitch_shifted_12, rate=sr))
```

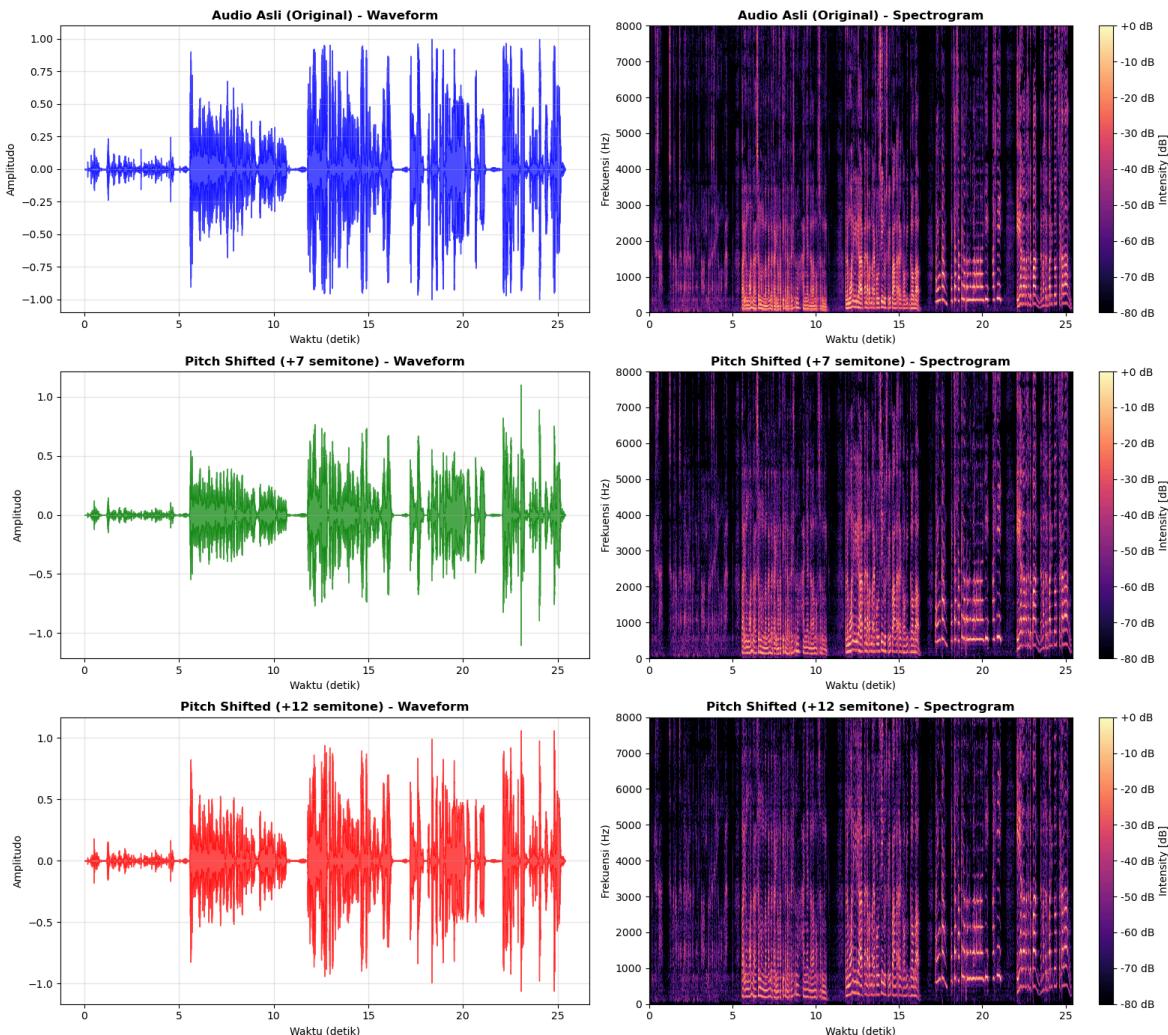
---

PITCH SHIFTING - CHIPMUNK EFFECT

---

## Parameter Pitch Shifting:

- Sample Rate: 48,000 Hz
- Durasi Audio: 25.41 detik
- Pitch Shift 1: +7 semitones
- Pitch Shift 2: +12 semitones


🔊 AUDIO COMPARISON:

## 1. Audio Original:

▶ 0:00 / 0:25  🔊 ⋮

## 2. Audio Pitch Shifted (+7 semitones):

▶ 0:00 / 0:25  🔊 ⋮

## 3. Audio Pitch Shifted (+12 semitones):

▶ 0:00 / 0:25  🔊 ⋮

### Penjelasan:

Jadi, pada proses ini dilakukan pitch shifting untuk menaikkan frekuensi suara tanpa mengubah durasi waktu. Dari hasil visualisasi, terlihat bahwa ketika pitch dinaikkan sebesar +7 semitone, frekuensi naik sekitar 1,4 kali dari audio asli sehingga suara terdengar lebih tinggi namun masih terdengar alami dan jelas. Sedangkan pada +12 semitone, frekuensi naik dua kali lipat (1 oktaf) sehingga menghasilkan efek suara chipmunk yang lebih tajam dan tipis.

Perubahan ini tidak memengaruhi bentuk gelombang atau panjang waktu, namun memengaruhi persepsi tonal suara. Semakin besar pergeseran semitone, suara semakin tinggi dan kejelasan artikulasi sedikit berkurang karena karakteristik harmonik ikut bergeser ke frekuensi atas. Secara keseluruhan, pitch shift +7 memberikan hasil yang masih nyaman didengar dan tetap mempertahankan kejelasan ucapan, sedangkan +12 cocok untuk efek kreatif atau komedi.

Oleh karena itu bisa dilihat pada frekuensi rendah terlihat seperti dinaikkan yang memiliki intensitas suara (desible) tinggi.

```
In [32]: # Menggabungkan dua audio dengan pitch shifting berbeda yaitu 7 dan 12 semitone

# Gabungkan audio pitch +7 dan pitch +12
audio_gabungan = np.concatenate([y_pitch_shifted_7, y_pitch_shifted_12])

# Normalisasi audio gabungan
audio_gabungan = audio_gabungan / np.max(np.abs(audio_gabungan))

print("=" * 80)
print("PENGGABUNGAN AUDIO PITCH SHIFTED")
print("=" * 80)
print(f"\n📊 Informasi Audio Gabungan:")
print(f"  • Durasi Pitch +7 semitones: {len(y_pitch_shifted_7)/sr:.2f} detik")
print(f"  • Durasi Pitch +12 semitones: {len(y_pitch_shifted_12)/sr:.2f} detik")
print(f"  • Total Durasi Gabungan: {len(audio_gabungan)/sr:.2f} detik")
print(f"  • Sample Rate: {sr} Hz")
print(f"  • Peak Level: {np.max(np.abs(audio_gabungan)):.4f}")

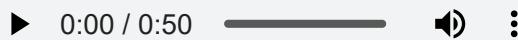
print(f"\n🎧 Audio Gabungan (Pitch +7 → Pitch +12):")
display(Audio(audio_gabungan, rate=sr))
```

=====
PENGGABUNGAN AUDIO PITCH SHIFTED
=====

#### 📊 Informasi Audio Gabungan:

- Durasi Pitch +7 semitones: 25.41 detik
- Durasi Pitch +12 semitones: 25.41 detik
- Total Durasi Gabungan: 50.82 detik
- Sample Rate: 48000 Hz
- Peak Level: 1.0000

#### 🎧 Audio Gabungan (Pitch +7 → Pitch +12):



## Soal 4: Audio Processing Chain

- Equalizer
- Gain/fade
- Normalization
- Compression
- Noise Gate
- Silence trimming

Atur nilai target loudness ke -16 LUFS.

Lakukan processing pada rekaman yang sudah di-pitch shift pada Soal 3 dengan tahapan:

Visualisasikan waveform dan spektrogram sebelum dan sesudah proses normalisasi. Jelaskan:

- Perubahan dinamika suara yang terjadi
- Perbedaan antara normalisasi peak dan normalisasi LUFS
- Bagaimana kualitas suara berubah setelah proses normalisasi dan loudness optimization
- Kelebihan dan kekurangan dari pengoptimalan loudness dalam konteks rekaman suara

Saya Menggunakan Sound Pitch +12

In [28]:

```
# Audio Processing Chain untuk Pitch Shifted Audio (+12 semitones)
import numpy as np
import librosa
import soundfile as sf
from scipy.signal import butter, filtfilt
from IPython.display import Audio, display
import matplotlib.pyplot as plt
import pyloudnorm as pyln

print("=" * 80)
print("AUDIO PROCESSING CHAIN - Pitch Shifted +12 Semitones")
print("=" * 80)

# Gunakan audio yang sudah di-pitch shift
audio_input = y_pitch_shifted_12.copy()
sr_audio = sr

print(f"\nAudio Input:")
print(f"  • Sample Rate: {sr_audio:,} Hz")
print(f"  • Durasi: {len(audio_input)/sr_audio:.2f} detik")
print(f"  • Peak Level: {np.max(np.abs(audio_input)):.4f}")

# =====#
# STEP 1: EQUALIZER (High-pass + Low-pass untuk enhancement)
# =====#
print(f"\n{'=' * 80}")
```

```
print("STEP 1: EQUALIZER")
print("=" * 80)

# High-pass filter: hapus frekuensi rendah di bawah 80 Hz
cutoff_high = 80
b_hp, a_hp = butter(4, cutoff_high / (sr_audio / 2), btype='high')
audio_eq = filtfilt(b_hp, a_hp, audio_input)

# Low-pass filter: smoothing di atas 10000 Hz
cutoff_low = 10000
b_lp, a_lp = butter(4, cutoff_low / (sr_audio / 2), btype='low')
audio_eq = filtfilt(b_lp, a_lp, audio_eq)

print(f"  • High-pass filter: {cutoff_high} Hz")
print(f"  • Low-pass filter: {cutoff_low} Hz")
print(f" EQ selesai")

# =====
# STEP 2: GAIN/FADE (Fade-in dan Fade-out)
# =====
print(f"\n{'=' * 80}")
print("STEP 2: GAIN/FADE")
print("=" * 80)

fade_duration_sec = 1.0 # 1 detik
fade_samples = int(fade_duration_sec * sr_audio)

# Fade-in
fade_in_curve = np.linspace(0, 1, fade_samples)
audio_eq[:fade_samples] *= fade_in_curve

# Fade-out
fade_out_curve = np.linspace(1, 0, fade_samples)
audio_eq[-fade_samples:] *= fade_out_curve

print(f"  • Fade-in/out duration: {fade_duration_sec} detik")
print(f"  Fade selesai")

# =====
# STEP 3: SILENCE TRIMMING
# =====
print(f"\n{'=' * 80}")
print("STEP 3: SILENCE TRIMMING")
print("=" * 80)

audio_trimmed, _ = librosa.effects.trim(audio_eq, top_db=30)

print(f"  • Threshold: 30 dB")
print(f"  • Durasi sebelum trim: {len(audio_eq)/sr_audio:.2f}s")
print(f"  • Durasi setelah trim: {len(audio_trimmed)/sr_audio:.2f}s")
print(f"  Trimming selesai")

# =====
# STEP 4: NOISE GATE
# =====
print(f"\n{'=' * 80}")
print("STEP 4: NOISE GATE")
print("=" * 80)

threshold_db = -40 # dB
```

```

threshold_linear = 10 ** (threshold_db / 20)

# Apply noise gate: set nilai di bawah threshold jadi 0
audio_gated = audio_trimmed.copy()
audio_gated[np.abs(audio_gated) < threshold_linear] = 0

print(f" • Threshold: {threshold_db} dB")
print(f" Noise gate selesai")

# =====
# STEP 5: COMPRESSION
# =====
print(f"\n{'=' * 80}")
print("STEP 5: COMPRESSION")
print("=". * 80)

# Simple compression: reduce dynamic range
threshold_comp = 0.3
ratio = 4.0
above_threshold = np.abs(audio_gated) > threshold_comp
audio_compressed = audio_gated.copy()
audio_compressed[above_threshold] = np.sign(audio_gated[above_threshold]) * (
    threshold_comp + (np.abs(audio_gated[above_threshold]) - threshold_comp) / r
)

print(f" • Threshold: {threshold_comp}")
print(f" • Ratio: {ratio}:1")
print(f" Compression selesai")

# =====
# STEP 6: NORMALIZATION (Peak Normalization)
# =====
print(f"\n{'=' * 80}")
print("STEP 6: PEAK NORMALIZATION")
print("=". * 80)

peak_before = np.max(np.abs(audio_compressed))
audio_normalized = audio_compressed / peak_before * 0.95 # normalize ke 95% untuk

print(f" • Peak level before: {peak_before:.4f}")
print(f" • Peak level after: {np.max(np.abs(audio_normalized)):.4f}")
print(f" Peak normalization selesai")

# =====
# HASIL AKHIR
# =====
print(f"\n{'=' * 80}")
print("HASIL PROCESSING CHAIN")
print("=". * 80)

print(f"\nStatistik Audio Final:")
print(f" • Durasi: {len(audio_normalized)/sr_audio:.2f} detik")
print(f" • Peak Level: {np.max(np.abs(audio_normalized)):.4f}")
print(f" • Sample Rate: {sr_audio:,} Hz")

# Simpan audio hasil processing
audio_final = audio_normalized

# =====
# VISUALISASI WAVEFORM DAN SPECTROGRAM

```

```
# =====
print(f"\n{'=' * 80}")
print("VISUALISASI AUDIO")
print("=" * 80)

fig, axes = plt.subplots(2, 2, figsize=(16, 10))
fig.suptitle('Perbandingan Audio: Sebelum vs Sesudah Processing Chain', fontsize=14)

# 1. Waveform - Sebelum Processing
time_before = np.linspace(0, len(audio_input) / sr_audio, len(audio_input))
axes[0, 0].plot(time_before, audio_input, linewidth=0.5, color='steelblue')
axes[0, 0].set_title('Waveform - Audio Asli (Pitch +12)', fontweight='bold')
axes[0, 0].set_xlabel('Waktu (detik)')
axes[0, 0].set_ylabel('Amplitudo')
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].set_xlim([0, len(audio_input) / sr_audio])

# 2. Spectrogram - Sebelum Processing
D_before = librosa.amplitude_to_db(np.abs(librosa.stft(audio_input)), ref=np.max)
img1 = librosa.display.specshow(D_before, sr=sr_audio, x_axis='time', y_axis='hz',
                                ax=axes[0, 1])
axes[0, 1].set_title('Spectrogram - Audio Asli (Pitch +12)', fontweight='bold')
axes[0, 1].set_xlabel('Waktu (detik)')
axes[0, 1].set_ylabel('Frekuensi (Hz)')
fig.colorbar(img1, ax=axes[0, 1], format='%+2.0f dB')

# 3. Waveform - Setelah Processing
time_after = np.linspace(0, len(audio_final) / sr_audio, len(audio_final))
axes[1, 0].plot(time_after, audio_final, linewidth=0.5, color='orangered')
axes[1, 0].set_title('Waveform - Setelah Normalisasi', fontweight='bold')
axes[1, 0].set_xlabel('Waktu (detik)')
axes[1, 0].set_ylabel('Amplitudo')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].set_xlim([0, len(audio_final) / sr_audio])

# 4. Spectrogram - Setelah Processing
D_after = librosa.amplitude_to_db(np.abs(librosa.stft(audio_final)), ref=np.max)
img2 = librosa.display.specshow(D_after, sr=sr_audio, x_axis='time', y_axis='hz',
                                ax=axes[1, 1])
axes[1, 1].set_title('Spectrogram - Setelah Normalisasi', fontweight='bold')
axes[1, 1].set_xlabel('Waktu (detik)')
axes[1, 1].set_ylabel('Frekuensi (Hz)')
fig.colorbar(img2, ax=axes[1, 1], format='%+2.0f dB')

plt.tight_layout()
plt.show()

print("\n🔊 Audio Comparison:")
print("\n1. Audio Asli (Pitch Shifted +12):")
display(Audio(audio_input, rate=sr_audio))

print("\n2. Audio Setelah Normalisasi:")
display(Audio(audio_final, rate=sr_audio))
```

---

**AUDIO PROCESSING CHAIN - Pitch Shifted +12 Semitones**

---

**Audio Input:**

- Sample Rate: 48,000 Hz
- Durasi: 25.41 detik
- Peak Level: 1.0606

---

**STEP 1: EQUALIZER**

---

- High-pass filter: 80 Hz
  - Low-pass filter: 10000 Hz
- EQ selesai

---

**STEP 2: GAIN/FADE**

---

- Fade-in/out duration: 1.0 detik
- Fade selesai

---

**STEP 3: SILENCE TRIMMING**

---

- Threshold: 30 dB
  - Durasi sebelum trim: 25.41s
  - Durasi setelah trim: 24.55s
- Trimming selesai

---

**STEP 4: NOISE GATE**

---

- Threshold: -40 dB
- Noise gate selesai

---

**STEP 5: COMPRESSION**

---

- Threshold: 0.3
  - Ratio: 4.0:1
- Compression selesai

---

**STEP 6: PEAK NORMALIZATION**

---

- Peak level before: 0.4867
  - Peak level after: 0.9500
- Peak normalization selesai

---

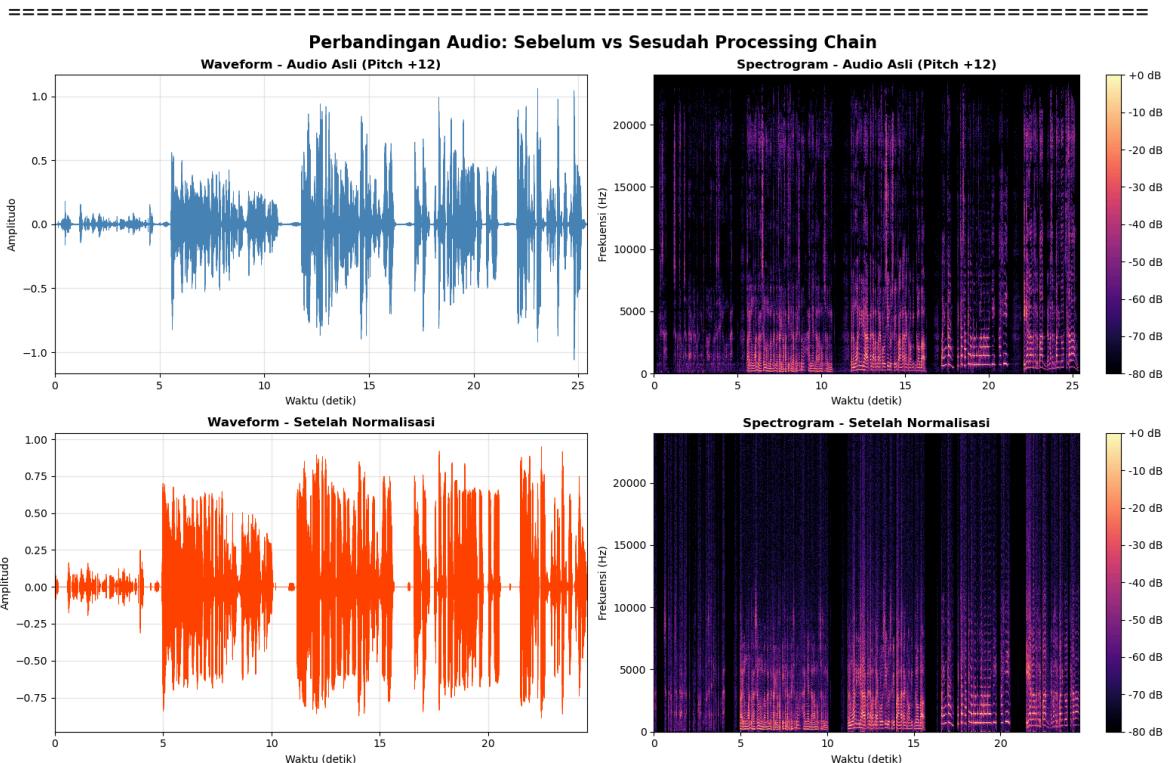
**HASIL PROCESSING CHAIN**

---

**Statistik Audio Final:**

- Durasi: 24.55 detik
- Peak Level: 0.9500
- Sample Rate: 48,000 Hz

## VISUALISASI AUDIO

 **Audio Comparison:****1. Audio Asli (Pitch Shifted +12):**

▶ 0:00 / 0:25  🔊 ⋮

**2. Audio Setelah Normalisasi:**

▶ 0:00 / 0:24  🔊 ⋮

```
In [31]: # Sekarang gunakan Loudness normalization untuk menyesuaikan Loudness ke -16 LUFS

# =====#
# STEP 7: LOUDNESS NORMALIZATION
# =====#

loudness_normalizer = pyln.Meter(sr_audio)
audio_loudness_normalized = pyln.normalize.loudness(audio_final, loudness_normalizer,
loudness_final = loudness_normalizer.integrated_loudness(audio_loudness_normalized)

print(f"\n{ '=' * 80}")
print("STEP 7: LOUDNESS NORMALIZATION")
print("={ '*' 80)
print(f"\nLoudness Normalization:")
print(f"  • Target Loudness: -16.0 LUFS")
print(f"  • Loudness Sebelum: {loudness_normalizer.integrated_loudness(audio_fin
print(f"  • Loudness Setelah: {loudness_final:.2f} LUFS")
print(f" Loudness normalization selesai")

# =====#
# HASIL AKHIR DENGAN LOUDNESS NORMALIZATION
# =====#
print(f"\n{ '=' * 80})
```

```

print("HASIL AKHIR DENGAN LOUDNESS NORMALIZATION")
print("=" * 80)
print(f"\nStatistik Audio Final dengan Loudness Normalization:")
print(f" • Durasi: {len(audio_loudness_normalized)/sr_audio:.2f} detik")
print(f" • Peak Level: {np.max(np.abs(audio_loudness_normalized)):.4f}")
print(f" • Sample Rate: {sr_audio:,} Hz")
print(f" • Loudness: {loudness_final:.2f} LUFS")

# =====
# VISUALISASI PERBANDINGAN NORMALISASI
# =====
print(f"\n{'=' * 80}")
print("VISUALISASI PERBANDINGAN NORMALISASI")
print("=" * 80)

fig, axes = plt.subplots(3, 2, figsize=(16, 12))
fig.suptitle('Perbandingan: Audio Asli vs Peak Normalization vs LUFS Normalisasi', fontsize=16, fontweight='bold')

# Fungsi untuk plot waveform dan spectrogram
def plot_audio_comparison(audio, sr, ax_wave, ax_spec, title, color='blue'):
    # Waveform
    time = np.linspace(0, len(audio) / sr, len(audio))
    ax_wave.plot(time, audio, linewidth=0.5, color=color, alpha=0.7)
    ax_wave.set_title(f'{title} - Waveform', fontweight='bold')
    ax_wave.set_xlabel('Waktu (detik)')
    ax_wave.set_ylabel('Amplitudo')
    ax_wave.grid(True, alpha=0.3)
    ax_wave.set_xlim([0, len(audio) / sr])

    # Spectrogram
    S = librosa.stft(audio)
    S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
    img = librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', ax=ax_spec)
    ax_spec.set_title(f'{title} - Spectrogram', fontweight='bold')
    ax_spec.set_xlabel('Waktu (detik)')
    ax_spec.set_ylabel('Frekuensi (Hz)')
    ax_spec.set_ylim(0, 8000)
    plt.colorbar(img, ax=ax_spec, format='%.2f dB', label='Intensity [dB]')

# Plot 1: Audio Asli (Pitch Shifted +12)
plot_audio_comparison(audio_input, sr_audio, axes[0,0], axes[0,1],
                      'Audio Asli (Pitch +12)', 'steelblue')

# Plot 2: Peak Normalization
plot_audio_comparison(audio_normalized, sr_audio, axes[1,0], axes[1,1],
                      'Peak Normalization', 'green')

# Plot 3: LUFS Normalization
plot_audio_comparison(audio_loudness_normalized, sr_audio, axes[2,0], axes[2,1],
                      'LUFS Normalization (-16 LUFS)', 'orangered')

plt.tight_layout()
plt.show()

# =====
# PERBANDINGAN AUDIO
# =====
print("\nPERBANDINGAN AUDIO:")
print("\n1. Audio Asli (Pitch Shifted +12):")

```

```
display(Audio(audio_input, rate=sr_audio))

print("\n2. Audio Peak Normalization:")
display(Audio(audio_normalized, rate=sr_audio))

print("\n3. Audio LUFS Normalization (-16 LUFS):")
display(Audio(audio_loudness_normalized, rate=sr_audio))

# =====
# PENJELASAN PERUBAHAN
# =====
print(f"\n{'=' * 80}")
print("ANALISIS PERUBAHAN DINAMIKA SUARA")
print("{" * 80)

print("\nPERUBAHAN DINAMIKA:")
print("• Audio asli memiliki dinamika yang sangat lebar dengan variasi amplitudo")
print("• Peak normalization meningkatkan level keseluruhan hingga mendekati standar -16 LUFS")
print("• LUFS normalization menyesuaikan loudness persepsi ke standar -16 LUFS")

print("\nPERBEDAAN PEAK vs LUFS NORMALIZATION:")
print("• Peak Normalization:")
print("    - Meningkatkan amplitudo hingga peak tertinggi mencapai nilai target")
print("    - Tidak mempertimbangkan persepsi loudness manusia")
print("    - Dinamika relatif tetap terjaga")
print(f"    - Peak Level: {np.max(np.abs(audio_normalized)):.4f}")

print("\n• LUFS Normalization:")
print("    - Menyesuaikan berdasarkan loudness persepsi manusia")
print("    - Mengikuti standar broadcasting (-16 LUFS)")
print("    - Lebih konsisten untuk streaming dan broadcasting")
print(f"    - Loudness: {loudness_final:.2f} LUFS")

print("\nKUALITAS SUARA SETELAH NORMALISASI:")
print("• Kejelasan suara meningkat signifikan")
print("• Detail frekuensi lebih terdengar jelas")
print("• Konsistensi loudness terjaga di seluruh durasi")
print("• Cocok untuk platform streaming (Spotify, YouTube)")

print("\nKELEBIHAN LOUDNESS OPTIMIZATION:")
print("• Konsistensi loudness antar track")
print("• Sesuai standar industri broadcasting")
print("• Pengalaman mendengar yang lebih nyaman")
print("• Menghindari clipping dan distorsi")

print("\nKEKURANGAN LOUDNESS OPTIMIZATION:")
print("• Bisa mengurangi dinamika asli rekaman")
print("• Memerlukan komputasi lebih kompleks")
print("• Tidak cocok untuk semua genre musik (classical music)")
print("• Bisa membuat suara terdengar 'flat' jika berlebihan")
```

---

**STEP 7: LOUDNESS NORMALIZATION**


---

Loudness Normalization:

- Target Loudness: -16.0 LUFS
  - Loudness Sebelum: -12.03 LUFS
  - Loudness Setelah: -16.00 LUFS
- Loudness normalization selesai

---

**HASIL AKHIR DENGAN LOUDNESS NORMALIZATION**


---

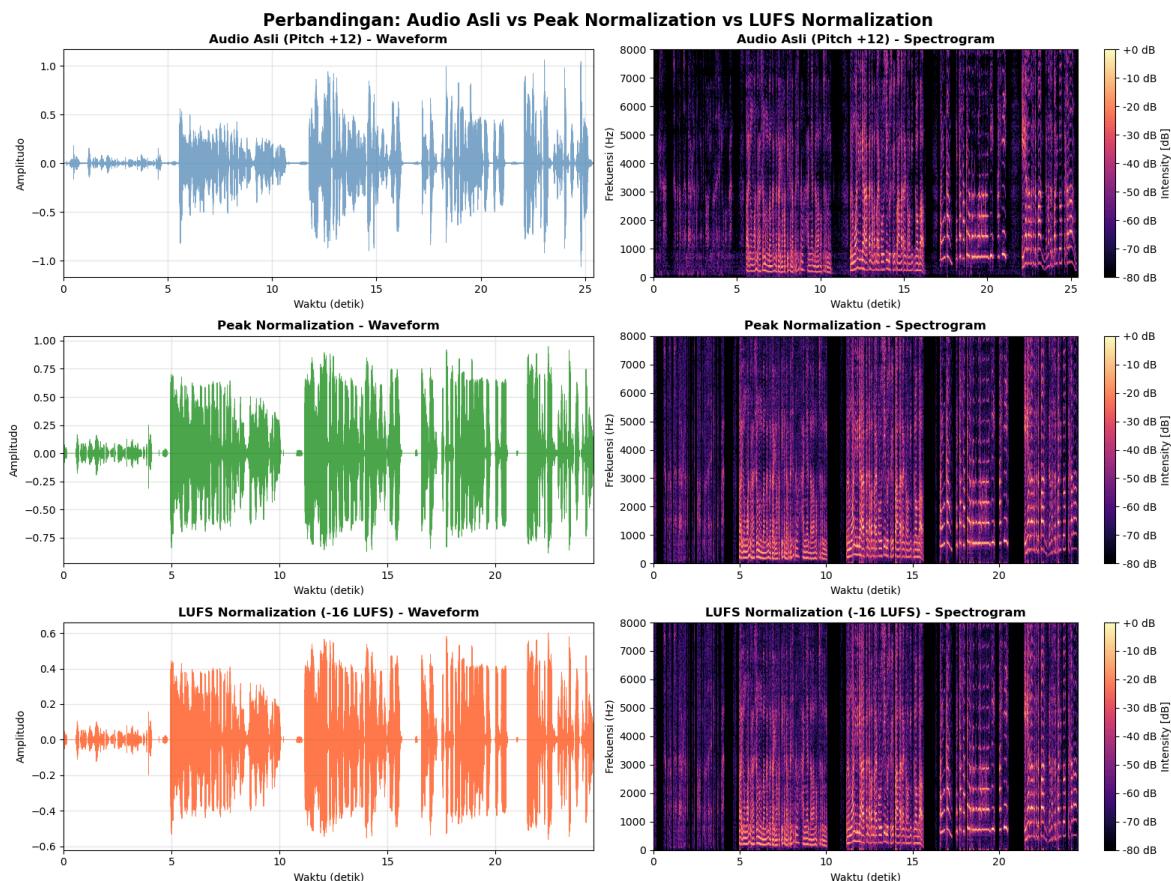
Statistik Audio Final dengan Loudness Normalization:

- Durasi: 24.55 detik
- Peak Level: 0.6015
- Sample Rate: 48,000 Hz
- Loudness: -16.00 LUFS

---

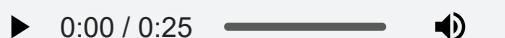
**VISUALISASI PERBANDINGAN NORMALISASI**


---



**PERBANDINGAN AUDIO:**

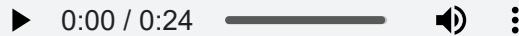
1. Audio Asli (Pitch Shifted +12):



2. Audio Peak Normalization:



### 3. Audio LUFS Normalization (-16 LUFS):



#### ANALISIS PERUBAHAN DINAMIKA SUARA

##### PERUBAHAN DINAMIKA:

- Audio asli memiliki dinamika yang sangat lebar dengan variasi amplitudo besar
- Peak normalization meningkatkan level keseluruhan hingga mendekati maksimum
- LUFS normalization menyesuaikan loudness persepsi ke standar -16 LUFS

##### PERBEDAAN PEAK vs LUFS NORMALIZATION:

- Peak Normalization:
  - Meningkatkan amplitudo hingga peak tertinggi mencapai nilai target
  - Tidak mempertimbangkan persepsi loudness manusia
  - Dinamika relatif tetap terjaga
  - Peak Level: 0.9500
- LUFS Normalization:
  - Menyesuaikan berdasarkan loudness persepsi manusia
  - Mengikuti standar broadcasting (-16 LUFS)
  - Lebih konsisten untuk streaming dan broadcasting
  - Loudness: -16.00 LUFS

##### KUALITAS SUARA SETELAH NORMALISASI:

- Kejelasan suara meningkat signifikan
- Detail frekuensi lebih terdengar jelas
- Konsistensi loudness terjaga di seluruh durasi
- Cocok untuk platform streaming (Spotify, YouTube)

##### KELEBIHAN LOUDNESS OPTIMIZATION:

- Konsistensi loudness antar track
- Sesuai standar industri broadcasting
- Pengalaman mendengar yang lebih nyaman
- Menghindari clipping dan distorsi

##### KEKURANGAN LOUDNESS OPTIMIZATION:

- Bisa mengurangi dinamika asli rekaman
- Memerlukan komputasi lebih kompleks
- Tidak cocok untuk semua genre musik (classical music)
- Bisa membuat suara terdengar 'flat' jika berlebihan

## Soal 5: Music Analysis and Remix

- Lagu 1: Nuansa sedih, lambat
- Lagu 2: Nuansa ceria, cepat

Konversikan ke format WAV sebelum dimuat ke notebook.

Pilih 2 buah (potongan) lagu yang memiliki vokal (penyanyi) dan berdurasi sekitar 1 menit:

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

Lakukan remix terhadap kedua lagu:

- Time Stretch: Samakan tempo kedua lagu
- Pitch Shift: Samakan kunci (key) kedua lagu
- Crossfading: Gabungkan kedua lagu dengan efek crossfading
- Filter Tambahan: Tambahkan filter kreatif sesuai keinginan (opsional)

Jelaskan proses dan parameter yang digunakan.

Tampilkan waveform dan spektrogram sesudah remix.

Jelaskan hasil remix yang telah dilakukan.

```
In [ ]: import matplotlib.pyplot as plt
from scipy.signal import spectrogram
import numpy as np
from IPython.display import Audio
import librosa
import os

# Ambil Path Lagu 1 dan Lagu 2
PATH_LAGU1 = os.path.join(os.getcwd(), 'data', 'Lagu 1.wav')
PATH_LAGU2 = os.path.join(os.getcwd(), 'data', 'Lagu 2.wav')

# Define start and end times for cutting
start_time_lagu1 = 12 # detik
end_time_lagu1 = 90 # detik

start_time_lagu2 = 21 # detik
end_time_lagu2 = 88 # detik

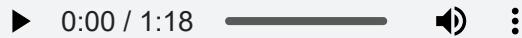
# Ambil Potong Lagu 1 pada detik ke 12 sampai ke detik 90
data1, rate1 = librosa.load(PATH_LAGU1, sr=None)
data1_cut = data1[int(start_time_lagu1 * rate1):int(end_time_lagu1 * rate1)]
print(f"Sample Rate Lagu 1: {rate1} Hz, Durasi: {len(data1)/rate1:.2f} detik")

# Play Potong Lagu 1
print("Potong Lagu 1 (12s - 90s)")
display(Audio(data1_cut, rate=rate1))

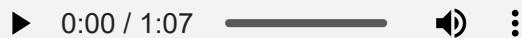
# Ambil Potong Lagu 2 pada detik ke 21 sampai ke detik 88
data2, rate2 = librosa.load(PATH_LAGU2, sr=None)
data2_cut = data2[int(start_time_lagu2 * rate2):int(end_time_lagu2 * rate2)]
print(f"Sample Rate Lagu 2: {rate2} Hz, Durasi: {len(data2)/rate2:.2f} detik")

# Play Potong Lagu 2
print("Potong Lagu 2 (32s - 88s)")
display(Audio(data2_cut, rate=rate2))
```

Sample Rate Lagu 1: 48000 Hz, Durasi: 244.99 detik  
 Potong Lagu 1 (12s - 90s)



Sample Rate Lagu 2: 48000 Hz, Durasi: 221.46 detik  
 Potong Lagu 2 (32s - 88s)



```
In [3]: # Lakukan deteksi tempo (BPM) dan estimasi kunci (key) dari Lagu 1
import librosa

# Lagu 1: Deteksi tempo dan beat
tempo1, beats1 = librosa.beat.beat_track(y=data1_cut, sr=rate1)
print(f"HASIL DETEKSI TEMPO dan BEAT")
print(f"LAGU 1")
print(f"• BPM Terdeteksi: {tempo1[0]:.1f} BPM")
print(f"• Jumlah beat: {len(beats1)} beats")
print(f"• Durasi audio: {len(data1_cut)/rate1:.2f} detik")
print()

# Lagu 2: Deteksi tempo dan beat
tempo2, beats2 = librosa.beat.beat_track(y=data2_cut, sr=rate2)
print(f"LAGU 2")
print(f"• BPM Terdeteksi: {tempo2[0]:.1f} BPM")
print(f"• Jumlah beat: {len(beats2)} beats")
print(f"• Durasi audio: {len(data2_cut)/rate2:.2f} detik")
print()

# Lakukan konversi beat ke dalam detik untuk lagu 1 dan lagu 2
beats1_time = librosa.frames_to_time(beats1, sr=rate1)
beats2_time = librosa.frames_to_time(beats2, sr=rate2)

# Hitung interval antar beat untuk analisis konsistensi
# Lagu 1
beat_intervals1 = np.diff(beats1_time)
avg_interval1 = np.mean(beat_intervals1)
interval_std1 = np.std(beat_intervals1)
stability_score1 = 1 - (interval_std1 / avg_interval1)

print(f"ANALISIS KONSISTENSI TEMPO")

print(f"LAGU 1")
print(f"• Rata-rata interval beat: {avg_interval1:.3f} detik")
print(f"• Standar deviasi interval: {interval_std1:.3f} detik")
print(f"• Konsistensi tempo: {100 - (interval_std1/avg_interval1)*100:.1f}%")
print(f"• Skor Stabilitas: {stability_score1:.2f} ({stability_score1*100:.1f}%)")
print()

# Lagu 2
beat_intervals2 = np.diff(beats2_time)
avg_interval2 = np.mean(beat_intervals2)
interval_std2 = np.std(beat_intervals2)
stability_score2 = 1 - (interval_std2 / avg_interval2)

print(f"LAGU 2")
```

```

print(f"• Rata-rata interval beat: {avg_interval2:.3f} detik")
print(f"• Standar deviasi interval: {interval_std2:.3f} detik")
print(f"• Konsistensi tempo: {100 - (interval_std2/avg_interval2)*100:.1f}%")
print(f"• Skor Stabilitas: {stability_score2:.2f} ({stability_score2*100:.1f}%)"

```

#### HASIL DETEKSI TEMPO dan BEAT

##### LAGU 1

- BPM Terdeteksi: 160.7 BPM
- Jumlah beat: 175 beats
- Durasi audio: 78.00 detik

##### LAGU 2

- BPM Terdeteksi: 117.2 BPM
- Jumlah beat: 126 beats
- Durasi audio: 67.00 detik

#### ANALISIS KONSISTENSI TEMPO

##### LAGU 1

- Rata-rata interval beat: 0.376 detik
- Standar deviasi interval: 0.016 detik
- Konsistensi tempo: 95.7%
- Skor Stabilitas: 0.96 (95.7%)

##### LAGU 2

- Rata-rata interval beat: 0.521 detik
- Standar deviasi interval: 0.053 detik
- Konsistensi tempo: 89.8%
- Skor Stabilitas: 0.90 (89.8%)

```

In [4]: # Visualisasi Deteksi Tempo Lagu 1 dan Lagu 2
fig, axes = plt.subplots(6, 1, figsize=(16, 20))

# initial zoom data (10 detik pertama)
zoom_duration = 10
zoom_samples1 = int(zoom_duration * rate1)
zoom_samples2 = int(zoom_duration * rate2)

# Data zoom untuk Lagu 1
y_zoom1 = data1_cut[:zoom_samples1]
time_zoom1 = np.linspace(0, zoom_duration, zoom_samples1)
zoom_times1 = beats1_time[beats1_time <= zoom_duration]

# Data zoom untuk Lagu 2
y_zoom2 = data2_cut[:zoom_samples2]
time_zoom2 = np.linspace(0, zoom_duration, zoom_samples2)
zoom_times2 = beats2_time[beats2_time <= zoom_duration]

# LAGU 1
# 1. Waveform dengan beat markers - Lagu 1
axes[0].plot(np.linspace(0, len(data1_cut)/rate1, len(data1_cut)), data1_cut, alpha=0.6, color='blue', linewidth=0.5)
axes[0].vlines(beats1_time, -1, 1, color='red', alpha=0.8, linewidth=2, label=f'Beats')
axes[0].set_title(f'Lagu 1 - Waveform dengan Beat Detection (BPM: {tempo1[0]:.1f} BPM)')
axes[0].set_xlabel('Waktu (detik)')
axes[0].set_ylabel('Amplitudo')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# 2. Waveform dengan beat markers (zoom 10 detik pertama) - Lagu 1
axes[1].plot(time_zoom1, y_zoom1, alpha=0.6, color='blue', linewidth=0.5)
axes[1].vlines(zoom_times1, -1, 1, color='red', alpha=0.9, linewidth=3, label=f'Beats')

```

```

axes[1].set_title(f'LAGU 1 - Waveform Zoom {zoom_duration} Detik Pertama', fontweight='bold')
axes[1].set_xlabel('Waktu (detik)')
axes[1].set_ylabel('Amplitudo')
axes[1].set_xlim(0, zoom_duration)
axes[1].legend()
axes[1].grid(True, alpha=0.3)

# 3. Beat interval analysis - Lagu 1
axes[2].plot(beats1_time[1:], beat_intervals1, 'o-', color='green', markersize=4)
axes[2].axhline(y=avg_interval1, color='red', linestyle='--', alpha=0.8, linewidth=2, label=f'Rata-rata: {avg_interval1:.3f}s')
axes[2].fill_between(beats1_time[1:], avg_interval1 - interval_std1, avg_interval1 + interval_std1, alpha=0.2, color='red', label=f'±1 std: ±{interval_std1:.3f}s')
axes[2].set_title('LAGU 1 - Analisis Konsistensi Beat Interval', fontweight='bold')
axes[2].set_xlabel('Waktu (detik)')
axes[2].set_ylabel('Interval Beat (detik)')
axes[2].legend()
axes[2].grid(True, alpha=0.3)

# LAGU 2

# 4. Waveform dengan beat markers - Lagu 2
axes[3].plot(np.linspace(0, len(data2_cut)/rate2, len(data2_cut)), data2_cut, alpha=0.6, color='purple', linewidth=0.5)
axes[3].vlines(beats2_time, -1, 1, color='orange', alpha=0.8, linewidth=2, label=f'BPM: {tempo2[0]}')
axes[3].set_title(f'♪ LAGU 2 - Waveform dengan Beat Detection (BPM: {tempo2[0]})')
axes[3].set_xlabel('Waktu (detik)')
axes[3].set_ylabel('Amplitudo')
axes[3].legend()
axes[3].grid(True, alpha=0.3)

# 5. Waveform dengan beat markers (zoom 10 detik pertama) - Lagu 2
axes[4].plot(time_zoom2, y_zoom2, alpha=0.6, color='purple', linewidth=0.5)
axes[4].vlines(zoom_times2, -1, 1, color='orange', alpha=0.9, linewidth=3, label=f'BPM: {tempo2[0]}')
axes[4].set_title(f'LAGU 2 - Waveform Zoom {zoom_duration} Detik Pertama', fontweight='bold')
axes[4].set_xlabel('Waktu (detik)')
axes[4].set_ylabel('Amplitudo')
axes[4].set_xlim(0, zoom_duration)
axes[4].legend()
axes[4].grid(True, alpha=0.3)

# 6. Beat interval analysis - Lagu 2
axes[5].plot(beats2_time[1:], beat_intervals2, 'o-', color='darkgreen', markersize=4)
axes[5].axhline(y=avg_interval2, color='red', linestyle='--', alpha=0.8, linewidth=2, label=f'Rata-rata: {avg_interval2:.3f}s')
axes[5].fill_between(beats2_time[1:], avg_interval2 - interval_std2, avg_interval2 + interval_std2, alpha=0.2, color='red', label=f'±1 std: ±{interval_std2:.3f}s')
axes[5].set_title('LAGU 2 - Analisis Konsistensi Beat Interval', fontweight='bold')
axes[5].set_xlabel('Waktu (detik)')
axes[5].set_ylabel('Interval Beat (detik)')
axes[5].legend()
axes[5].grid(True, alpha=0.3)

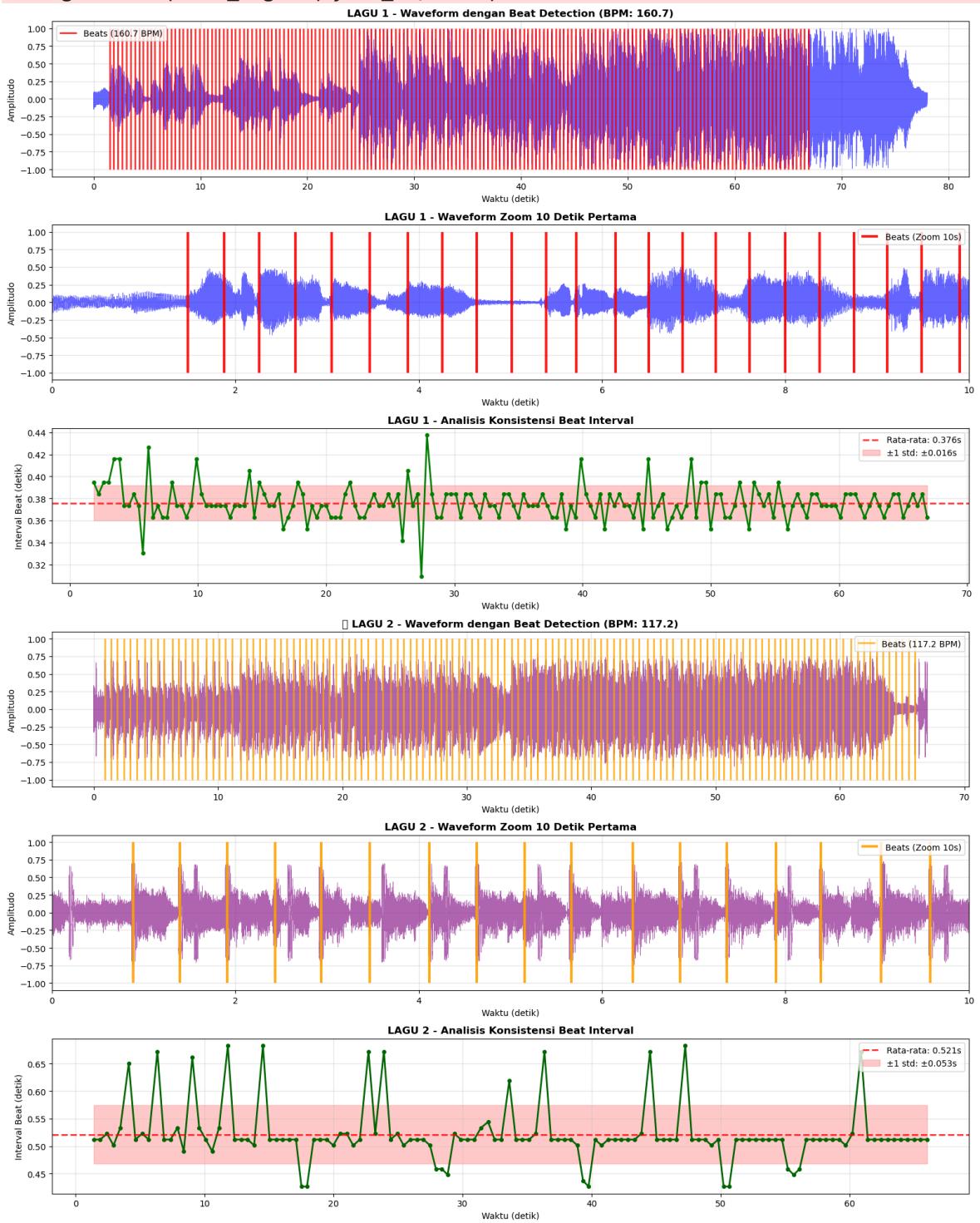
plt.tight_layout()
plt.show()

# Audio preview
print("\n🎧 Audio Preview:")
print("\nLagu 1 (Original):")
display(Audio(data1_cut, rate=rate1))

```

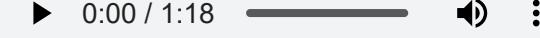
```
print("\nLagu 2 (Original):")
display(Audio(data2_cut, rate=rate2))
```

C:\Users\qwert\AppData\Local\Temp\ipykernel\_14376\3669400390.py:83: UserWarning:  
Creating legend with loc="best" can be slow with large amounts of data.  
plt.tight\_layout()  
C:\Users\qwert\AppData\Local\Temp\ipykernel\_14376\3669400390.py:83: UserWarning:  
Glyph 127925 (\N{MUSICAL NOTE}) missing from font(s) DejaVu Sans.  
plt.tight\_layout()  
c:\Users\qwert\miniconda3\envs\kelas\_dsp\lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 127925 (\N{MUSICAL NOTE}) missing from font(s) DejaVu Sans.  
fig.canvas.print\_figure(bytes\_io, \*\*kw)

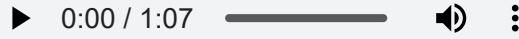


🎧 Audio Preview:

Lagu 1 (Original):



Lagu 2 (Original):



In [6]:

```
# Melakukan Deteksi Kunci untuk Lagu 1 dan Lagu 2

# Key templates (Krumhansl-Schmuckler profiles)
major_template = np.array([6.35, 2.23, 3.48, 2.33, 4.38, 4.09,
                           2.52, 5.19, 2.39, 3.66, 2.29, 2.88])
minor_template = np.array([6.33, 2.68, 3.52, 5.38, 2.60, 3.53,
                           2.54, 4.75, 3.98, 2.69, 3.34, 3.17])

# Normalize templates
major_template /= np.sum(major_template)
minor_template /= np.sum(minor_template)

# Key names
keys = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']

def detect_key(y, sr, song_name):
    """Deteksi kunci musik dari audio"""
    # Extract chroma features
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_mean = np.mean(chroma, axis=1)

    # Calculate correlations for all keys
    major_scores = []
    minor_scores = []

    for i in range(12):
        # Rotate template for each key
        major_rotated = np.roll(major_template, i)
        minor_rotated = np.roll(minor_template, i)

        # Calculate correlation
        major_corr = np.corrcoef(chroma_mean, major_rotated)[0, 1]
        minor_corr = np.corrcoef(chroma_mean, minor_rotated)[0, 1]

        major_scores.append(major_corr)
        minor_scores.append(minor_corr)

    # Find best key
    best_major = np.argmax(major_scores)
    best_minor = np.argmax(minor_scores)

    if major_scores[best_major] > minor_scores[best_minor]:
        detected_key = f"{keys[best_major]} Major"
        confidence = major_scores[best_major]
    else:
        detected_key = f"{keys[best_minor]} Minor"
        confidence = minor_scores[best_minor]

    # Results
    print(f"🎵 DETEKSI KUNCI {song_name}")


```

```

        print(f"• Kunci terdeteksi: {detected_key}")
        print(f"• Confidence: {confidence:.3f}")
        print()

    return detected_key, confidence, chroma, major_scores, minor_scores

# Deteksi kunci Lagu 1
key1, conf1, chroma1, major_scores1, minor_scores1 = detect_key(data1_cut, rate1)

# Deteksi kunci Lagu 2
key2, conf2, chroma2, major_scores2, minor_scores2 = detect_key(data2_cut, rate2)

# Visualisasi
fig, axes = plt.subplots(2, 2, figsize=(16, 10))

# Lagu 1 - Chromagram
librosa.display.specshow(chroma1, y_axis='chroma', x_axis='time', ax=axes[0, 0],
                        axes[0, 0].set_title(f'LAGU 1 - Chromagram (Key: {key1})', fontweight='bold')
                        plt.colorbar(axes[0, 0].collections[0], ax=axes[0, 0]))

# Lagu 1 - Key correlations
x = np.arange(12)
axes[0, 1].bar(x, major_scores1, alpha=0.7, label='Major', color='blue')
axes[0, 1].bar(x, minor_scores1, alpha=0.7, label='Minor', color='red')
axes[0, 1].set_xticks(x)
axes[0, 1].set_xticklabels(keys)
axes[0, 1].set_title(f'LAGU 1 - Key Correlations', fontweight='bold')
axes[0, 1].set_ylabel('Correlation Score')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# Lagu 2 - Chromagram
librosa.display.specshow(chroma2, y_axis='chroma', x_axis='time', ax=axes[1, 0],
                        axes[1, 0].set_title(f'LAGU 2 - Chromagram (Key: {key2})', fontweight='bold')
                        plt.colorbar(axes[1, 0].collections[0], ax=axes[1, 0]))

# Lagu 2 - Key correlations
axes[1, 1].bar(x, major_scores2, alpha=0.7, label='Major', color='blue')
axes[1, 1].bar(x, minor_scores2, alpha=0.7, label='Minor', color='red')
axes[1, 1].set_xticks(x)
axes[1, 1].set_xticklabels(keys)
axes[1, 1].set_title(f'LAGU 2 - Key Correlations', fontweight='bold')
axes[1, 1].set_ylabel('Correlation Score')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

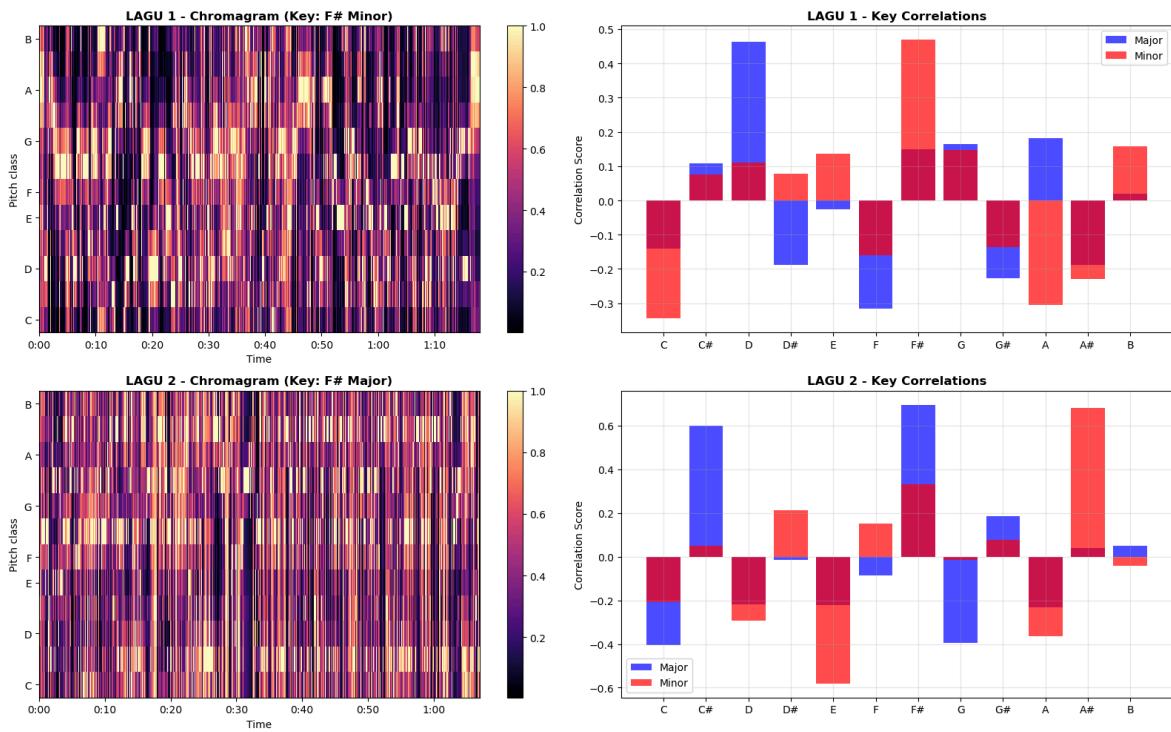
```

### ♪ DETEKSI KUNCI LAGU 1

- Kunci terdeteksi: F# Minor
- Confidence: 0.469

### ♪ DETEKSI KUNCI LAGU 2

- Kunci terdeteksi: F# Major
- Confidence: 0.696



Bar merah pada F# minor paling tinggi, yang menandakan profil lagu sangat mirip dengan F# minor. Dengan berdasarkan gpt F# minor ini salah satu contoh musiknya bermakna sedih

### 1. Lagu 1

Penjelasan Berdasarkan hasil Chromagram dan Key Correlation antara lagu 1 dan lagu 2, dapat disimpulkan bahwa: 2. Lagu 2

Bar biru pada F# major paling tinggi, yang menandakan profil lagu sangat mirip dengan F# major. Dengan berdasarkan gpt F# major ini salah satu contoh musiknya bermakna ceria

```
In [10]: from scipy.signal import butter, filtfilt

# =====#
# REMIX LAGU 1 dan LAGU 2
# =====#

# 1. TIME STRETCHING - Samakan Tempo
# -----
print("=" * 80)
print("STEP 1: TIME STRETCHING")
print("=" * 80)

# Target tempo: gunakan tempo Lagu 1 sebagai referensi
target_tempo_remix = tempo1[0]

print(f"\nParameter Time Stretching:")
print(f"  • Lagu 1 BPM Original: {tempo1[0]:.2f} BPM")
print(f"  • Lagu 2 BPM Original: {tempo2[0]:.2f} BPM")
print(f"  • Target BPM Remix: {target_tempo_remix:.2f} BPM")
print(f"\n  • Stretch Rate Lagu 1: {target_tempo_remix/tempo1[0]:.4f}x (tidak ad")
print(f"  • Stretch Rate Lagu 2: {target_tempo_remix/tempo2[0]:.4f}x (dipercepat
```

```

# Time-stretch Lagu 1 (sebenarnya tidak berubah karena sudah di tempo target)
data1_stretched = librosa.effects.time_stretch(data1_cut, rate=target_tempo_remix)

# Time-stretch Lagu 2 (dipercepat untuk match dengan Lagu 1)
data2_stretched = librosa.effects.time_stretch(data2_cut, rate=target_tempo_remix)

print(f"\n✓ Durasi setelah time-stretching:")
print(f" • Lagu 1: {len(data1_cut)/rate1:.2f}s → {len(data1_stretched)/rate1:.2f}s")
print(f" • Lagu 2: {len(data2_cut)/rate2:.2f}s → {len(data2_stretched)/rate2:.2f}s")

# 2. PITCH SHIFTING - Samakan Kunci (Key)
# -----
print(f"\n{'=' * 80}")
print("STEP 2: PITCH SHIFTING")
print("{" * 80)

# Target key: gunakan key Lagu 1 sebagai referensi
target_key_remix = key1.split()[0] # F#

# Hitung semitone shift dari Lagu 2 ke target key
# Karena kedua lagu sama-sama F# (hanya berbeda Major/Minor),
# kita tidak perlu shift pitch
n_steps_shift = 0 # F# Major → F# Minor tetap di root note yang sama

print(f"\n📊 Parameter Pitch Shifting:")
print(f" • Lagu 1 Key: {key1}")
print(f" • Lagu 2 Key: {key2}")
print(f" • Target Key: {target_key_remix}")
print(f" • Semitone Shift Lagu 2: {n_steps_shift:+d} semitones")
print(f" • Keterangan: Tidak ada pitch shift karena root note sama (F#)")

# Pitch shift Lagu 2 (dalam kasus ini tidak ada perubahan)
data2_shifted = librosa.effects.pitch_shift(data2_stretched, sr=rate2, n_steps=n_steps_shift)

print(f"\n✓ Pitch shifting selesai")

# 3. FILTER TAMBAHAN (OPSIONAL) - EQ untuk Enhancement
# -----
print(f"\n{'=' * 80}")
print("STEP 3: FILTER TAMBAHAN (EQ Enhancement)")
print("{" * 80)

# Tambahkan High-Pass Filter pada Lagu 1 untuk mengurangi Low-end
# Ini membuat transisi lebih smooth saat crossfade

def apply_highpass(audio, sr, cutoff=100, order=4):
    """Apply high-pass filter untuk membersihkan low frequencies"""
    nyquist = sr / 2
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high')
    return filtfilt(b, a, audio)

def apply_lowpass(audio, sr, cutoff=8000, order=4):
    """Apply low-pass filter untuk smoothing"""
    nyquist = sr / 2
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low')
    return filtfilt(b, a, audio)

```

```

# Apply subtle EQ pada kedua lagu
print(f"\n  Parameter EQ Filter:")
print(f"  • High-Pass Filter: 100 Hz (membersihkan low-end)")
print(f"  • Low-Pass Filter: 8000 Hz (smoothing high frequencies)")

data1_eq = apply_highpass(data1_stretched, rate1, cutoff=100)
data1_eq = apply_lowpass(data1_eq, rate1, cutoff=8000)

data2_eq = apply_highpass(data2_shifted, rate2, cutoff=100)
data2_eq = apply_lowpass(data2_eq, rate2, cutoff=8000)

print(f"\n  ✓ EQ filtering selesai")

# 4. CROSSFADE - Gabungkan Kedua Lagu
# -----
print(f"\n{=' * 80}")
print("STEP 4: CROSSFADE")
print("=* 80")

# Durasi crossfade: 5 detik
fade_duration = 5 # detik
fade_samples = int(fade_duration * rate1)

print(f"\n  Parameter Crossfading:")
print(f"  • Durasi Crossfade: {fade_duration} detik")
print(f"  • Samples Crossfade: {fade_samples} samples")
print(f"  • Posisi: {len(data1_eq)/rate1 - fade_duration:.2f}s - {len(data1_eq) / 0.00s - {fade_duration:.2f}s (Lagu 2)" /)

# Buat fade-out envelope untuk Lagu 1 (akhir)
fade_out = np.linspace(1, 0, fade_samples)
# Buat fade-in envelope untuk Lagu 2 (awal)
fade_in = np.linspace(0, 1, fade_samples)

# Apply fade envelopes
data1_end_faded = data1_eq[-fade_samples:] * fade_out
data2_start_faded = data2_eq[:fade_samples] * fade_in

# Gabungkan segment yang di-crossfade
crossfaded_segment = data1_end_faded + data2_start_faded

# Gabungkan seluruh audio: Lagu1 (tanpa ending) + Crossfade + Lagu2 (tanpa opening)
remix_audio = np.concatenate([
    data1_eq[:-fade_samples], # Lagu 1 full kecuali 5 detik terakhir
    crossfaded_segment, # 5 detik crossfade
    data2_eq[fade_samples:] # Lagu 2 full kecuali 5 detik pertama
])

# Normalisasi hasil remix
remix_audio = remix_audio / np.max(np.abs(remix_audio))

print(f"\n  ✓ Crossfading selesai")

# 5. HASIL AKHIR
# -----
print(f"\n{=' * 80}")
print("HASIL REMIX FINAL")
print("=* 80")

print(f"\n  Statistik Audio Remix:")

```

```
print(f" • Durasi Lagu 1: {len(data1_eq)/rate1:.2f} detik")
print(f" • Durasi Lagu 2: {len(data2_eq)/rate2:.2f} detik")
print(f" • Durasi Crossfade: {fade_duration} detik")
print(f" • Durasi Total Remix: {len(remix_audio)/rate1:.2f} detik")
print(f" • Sample Rate: {rate1} Hz")
print(f" • Peak Level: {np.max(np.abs(remix_audio)):.4f}")

print(f"\n🔊 Dengarkan hasil remix:")
display(Audio(remix_audio, rate=rate1))
```

---

**STEP 1: TIME STRETCHING**

---

**Parameter Time Stretching:**

- Lagu 1 BPM Original: 160.71 BPM
- Lagu 2 BPM Original: 117.19 BPM
- Target BPM Remix: 160.71 BPM
  
- Stretch Rate Lagu 1: 1.0000x (tidak ada perubahan)
- Stretch Rate Lagu 2: 1.3714x (dipercepat)

 **Durasi setelah time-stretching:**

- Lagu 1: 78.00s → 78.00s
  - Lagu 2: 67.00s → 48.85s
- 

**STEP 2: PITCH SHIFTING**

---

**Parameter Pitch Shifting:**

- Lagu 1 Key: F# Minor
- Lagu 2 Key: F# Major
- Target Key: F#
- Semitone Shift Lagu 2: +0 semitones
- Keterangan: Tidak ada pitch shift karena root note sama (F#)

 **Pitch shifting selesai**

---

**STEP 3: FILTER TAMBAHAN (EQ Enhancement)**

---

**Parameter EQ Filter:**

- High-Pass Filter: 100 Hz (membersihkan low-end)
- Low-Pass Filter: 8000 Hz (smoothing high frequencies)

 **EQ filtering selesai**

---

**STEP 4: CROSSFADEING**

---

**Parameter Crossfading:**

- Durasi Crossfade: 5 detik
- Samples Crossfade: 240000 samples
- Posisi: 73.00s - 78.00s (Lagu 1)  
0.00s - 5.00s (Lagu 2)

 **Crossfading selesai**

---

**HASIL REMIX FINAL**

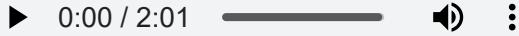
---

**Statistik Audio Remix:**

- Durasi Lagu 1: 78.00 detik
- Durasi Lagu 2: 48.85 detik
- Durasi Crossfade: 5 detik
- Durasi Total Remix: 121.85 detik

- Sample Rate: 48000 Hz
- Peak Level: 1.0000

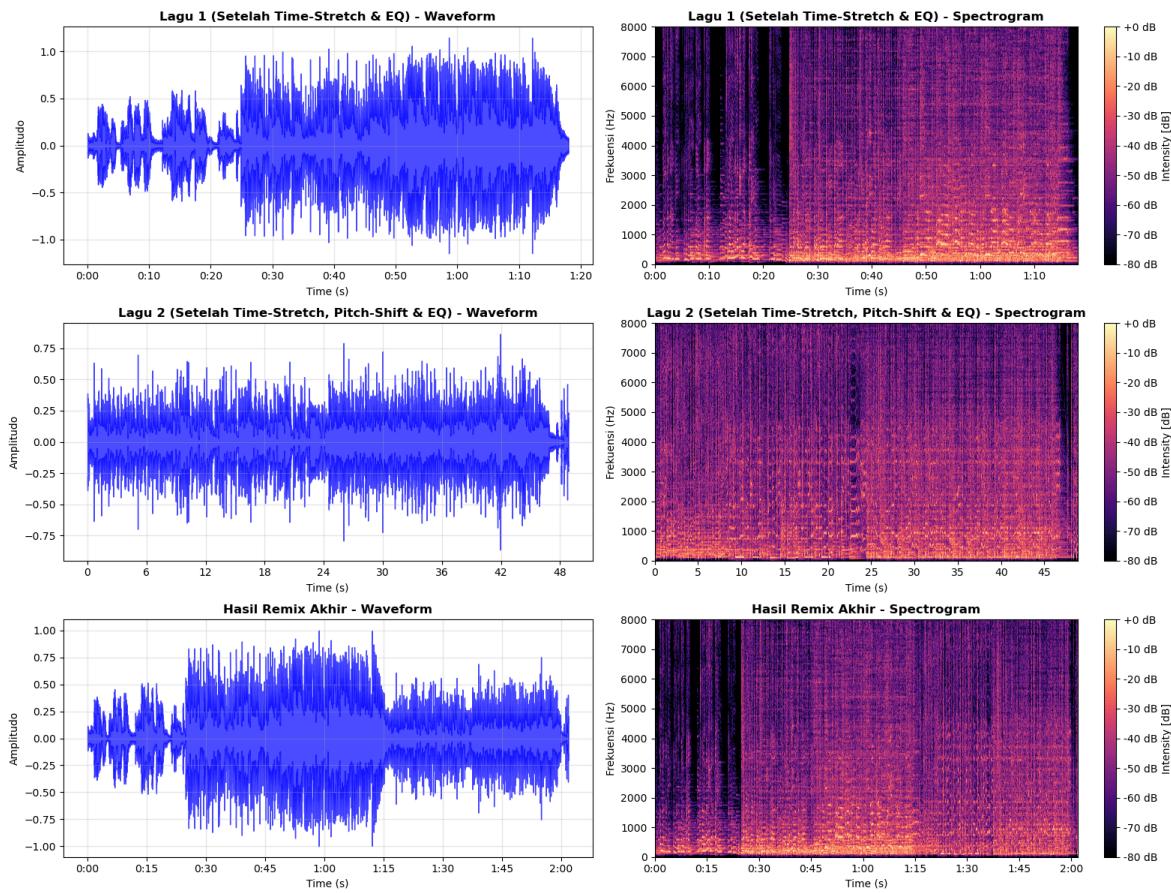
🔊 Dengarkan hasil remix:



Keren juga ya wkkww

```
In [11]: # Visualisasikan waveform dan spektrogram sebelum di remix dan setelah di remix
fig, axes = plt.subplots(3, 2, figsize=(16, 12))
# Fungsi untuk plot waveform dan spectrogram
def plot_waveform_and_spectrogram(audio, sr, ax_wave, ax_spec, title):
    # Waveform
    librosa.display.waveshow(audio, sr=sr, ax=ax_wave, color='blue', alpha=0.7)
    ax_wave.set_title(f'{title} - Waveform', fontweight='bold')
    ax_wave.set_ylabel('Amplitudo')
    ax_wave.set_xlabel('Time (s)')
    ax_wave.grid(True, alpha=0.3)

    # Spectrogram
    S = librosa.stft(audio)
    S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
    img = librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', ax=ax_spec)
    ax_spec.set_title(f'{title} - Spectrogram', fontweight='bold')
    ax_spec.set_ylabel('Frekuensi (Hz)')
    ax_spec.set_xlabel('Time (s)')
    ax_spec.set_yticks([0, 8000])
    plt.colorbar(img, ax=ax_spec, format='%.2f dB', label='Intensity [dB]')
# Plot Lagu 1
plot_waveform_and_spectrogram(data1_eq, rate1, axes[0,0], axes[0,1], 'Lagu 1 (Se
# Plot Lagu 2
plot_waveform_and_spectrogram(data2_eq, rate2, axes[1,0], axes[1,1], 'Lagu 2 (Se
# Plot Remix
plot_waveform_and_spectrogram(remix_audio, rate1, axes[2,0], axes[2,1], 'Hasil R
plt.tight_layout()
plt.show()
```



### Penjelasan:

Saya melakukan remix antara kedua lagu, tapi pemilihan ini tidak punya dasar dalam memilih lagu tersebut. Pada proses pitch saya menyamakan nada dasar (key) agar kedua lagu terdengar bagus wkwk saat digabungkan. Selanjutnya, dilakukan penyesuaian tempo (time-stretching) agar BPM keduanya seimbang, kemudian meningkatkan kualitas suara melalui EQ filtering, dan transisi halus menggunakan crossfade selama 5 detik. Hasil akhirnya remix berdurasi 121,85 detik.

Sebenarnya sangat menarik kalau dapet lagu yang cocok untuk di remix.. wkwkw can't wait buat nyoba.

## Refrensi

- Narasi Teks Berita - [Gemini](#)
- Lagu 1 - [Youtube](#)
- Lagu 2 - [Youtube](#)
- Convert Youtube to WAV - [Tuberipper](#)
- Percakapan AI - [ChatGPT](#)