Exercise - Audio Processing

> Baca dengen seksama dan teliti

Nama : Deva Ahmad NIM : 122140015

Link Kumpulan Audio: TEKAN AKU TEKAN AKU

Perhatian: Konversi dahulu audio ke format WAV, bisa menggunakan aplikasi seperti Format Factory, atau website Cloudconvert

Soal 1

- Rekamlah suara Anda sendiri selama 25 detik dimana anda membaca sebuah teks berita.
- Dalam 25 detik rekaman tersebut, anda harus merekam:
  - 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 ini).
- Visualisasikan waveform dan spektogram dari rekaman suara anda
- Sertakan penjelasan singkat mengenai hasil visualisasi tersebut.
- Lakukan resampling pada file audio anda kemudian bandingkan kualitas dan durasinya.

Install library yang diperlukan

Sebelum memulai, kita perlu menginstal beberapa library yang dibutuhkan, yaitu Numpy, Matplotlib, Librosa, Wave, IPython dan OS. Gunakan perintah *pip install librosa matplotlib numpy ipython*

Import Library yang diperlukan

Kita akan mengimport beberapa library yang berfungsi untuk membaca, memproses, dan memvisualisasikan suara.

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Audio
import librosa
import librosa.display
```

```
import wave
import os
```

Penjelasan:

- numpy: untuk operasi numerik, vektor waktu, dan transformasi data audio.
- librosa: untuk memuat file audio dan melakukan analisis sinyal (durasi, amplitudo, spektrogram, dll).
- matplotlib: untuk menampilkan waveform dan spektrogram.
- ipython: untuk menampilkan audio player di notebook (Audio()).
- wave: untuk membaca metadata file audio
- os: untuk mengecek keberadaan file audio

Membaca dan Memproses File Audio

Sebelum visualisasi, kita perlu membaca file WAV yang telah direkam untuk menampilkan informasi tentang file tersebut.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Audio
import librosa
import librosa.display
import wave
import os

# Memuat Audio (sama dengan Week 1)
PATH_AUDIO = os.path.join(os.getcwd(), 'audio', 'Soal 1.wav')

# Load audio dengan error handling
try:
    y, sr = librosa.load(PATH_AUDIO, sr=None)  # sr=None mempertahankan sample rate
    source_info = f"✅ Berhasil memuat: {PATH_AUDIO}"
except Exception as e:
    print(f"❌ Error loading audio: {e}")
    raise

# Konversi ke mono jika stereo
if y.ndim > 1:
    y = np.mean(y, axis=0)
    source_info += " - Dikonversi ke mono"

# Normalisasi dan konversi tipe data
y = y.astype(np.float32)

print("🎵 AUDIO LOADED")
print("=" * 40)
print(source_info)
print(f"📊 Shape: {y.shape}")
print(f"🎵 Sample rate: {sr:,} Hz")
print(f"⏱️ Durasi: {len(y)/sr:.2f} detik")
print(f"📈 Amplitude range: {y.min():.3f} to {y.max():.3f}")
```

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

```
🎵 AUDIO LOADED
========================================
✅ Berhasil memuat: e:\Semester 7\MULMED\Minggu 6\.venv\audio\Soal 1.wav
📊 Shape: (1261568,)
🎵 Sample rate: 48,000 Hz
⏱️ Durasi: 26.28 detik
📈 Amplitude range: -0.851 to 0.951
▶️ Audio Original
```

0:00 / 0:26

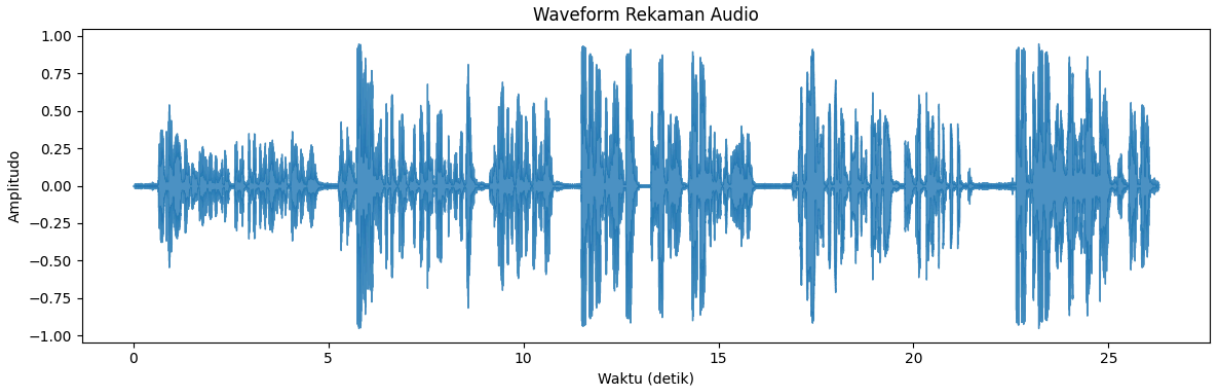
Visualisasi Audio

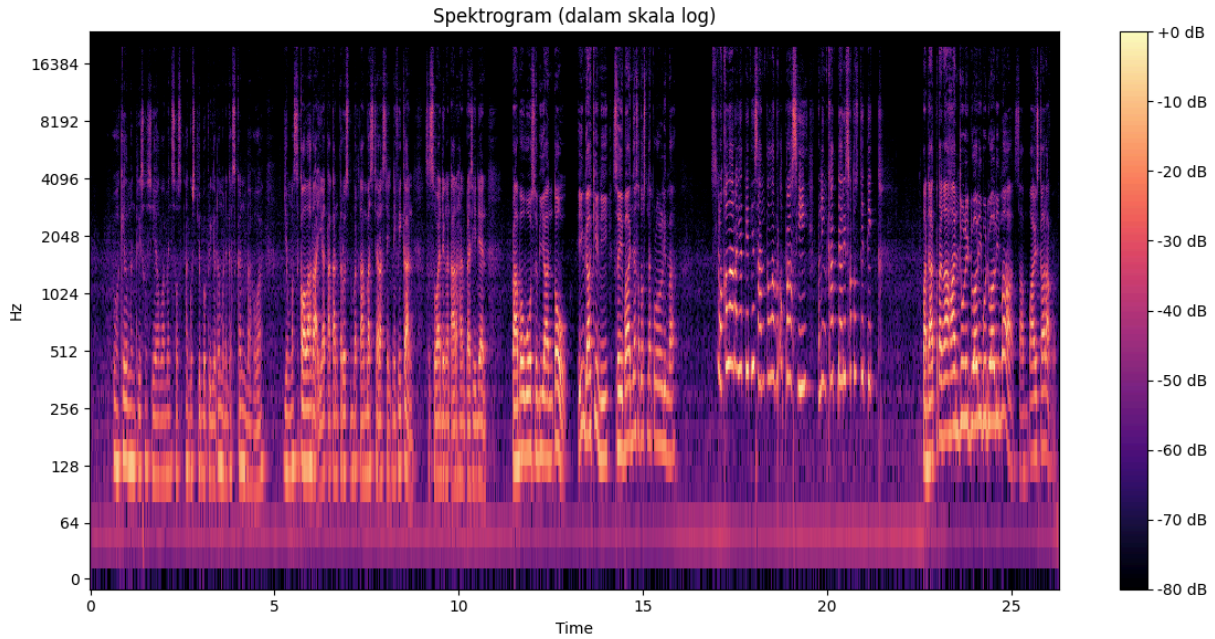
Sekarang tampilkan audio yang telah direkam dalam visualisasi Waveform dan Spectogram

In [3]:

```
# Hitung Nilai Peak (linear ke dBFS)
peak = np.max(np.abs(y))
peak_db = 20 * np.log10(peak)

# Hitung Nilai RMS (linear ke dBFS)
rms = np.sqrt(np.mean(y**2))
rms_db = 20 * np.log10(rms)

print("\n📏 Metrik Loudness:")
print(f"🔊 Peak: {peak:.3f} ({peak_db:.2f} dBFS)")
print(f"📈 RMS:  {rms:.3f} ({rms_db:.2f} dBFS)")

# === Visualisasi Waveform ===
plt.figure(figsize=(12, 4))
librosa.display.waveshow(y, sr=sr, alpha=0.8)
plt.title("Waveform Rekaman Audio")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.tight_layout()
plt.show()

# === Visualisasi Spektrogram ===
# Konversi ke dB
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)

plt.figure(figsize=(12, 6))
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram (dalam skala log)")
plt.tight_layout()
plt.show()
```

```
📏 Metrik Loudness:
🔊 Peak: 0.951 (-0.44 dBFS)
📈 RMS:  0.138 (-17.23 dBFS)
```





- Penjelasan Visualisasi Waveform Waveform di atas menunjukan rekaman audio sekitar 27 detik. Di sumbu X ada waktu, sedangkan sumbu Y itu amplitudo atau keras/lembutnya suara. Terlihat ada bagian yang rapat dan tinggi, contoh di sekitar detik ke-5, 11, 19, sama 24. Kemungkinan itu adalah bagian orang ngomong lebih keras atau ada suara yang dominan. Sebaliknya, saat bagian gelombangnya kecil dan jarang, berarti sedang hening atau suara lebih pelan.

- Penjelasan Visualisasi Spectogram Spectogram menunjukkan audio yang sama, tapi bukan hanya keras/pelan seperti waveform, melainkan juga frekuensi apa saja yang aktif sepanjang waktu. Sama seperti waveform juga, bagian disekitar detik 5, 11, 19 dan 24 juga frekuensi menengah dan tinggi lebih terang, menandakan bahwa kepadatan suara tinggi.

Resampling Audio

Kita akan melakukan down resampling file audio dari 48000 Hz ke 16000 Hz

In [5]:
```
# Target sample rate baru
target_sr = 16000
```

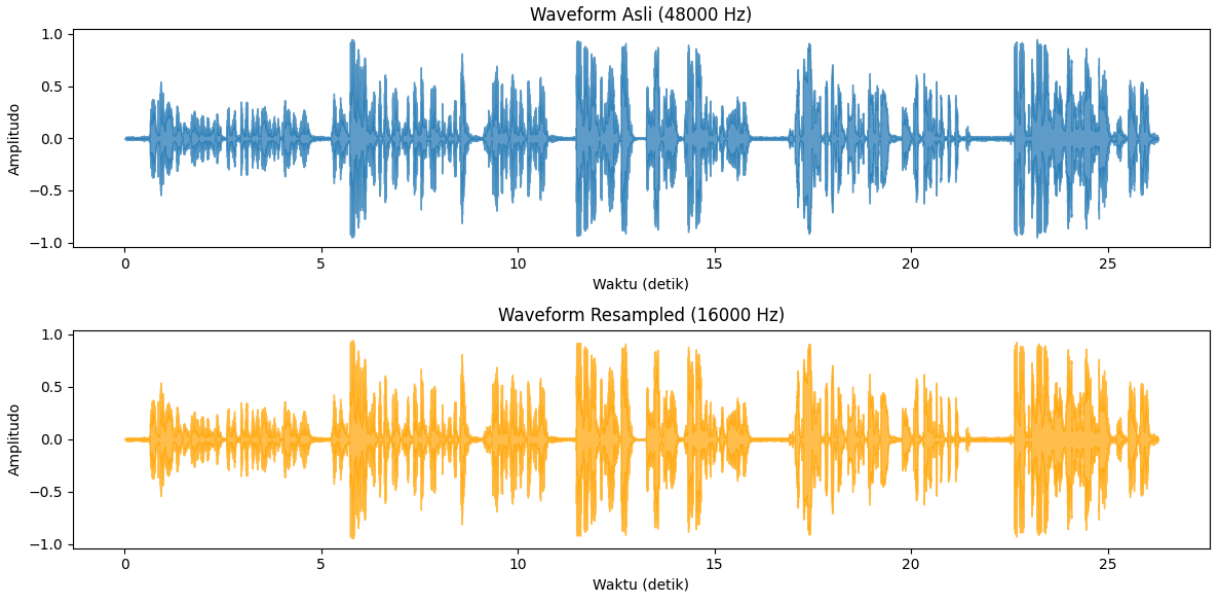
```python
# Resampling audio
y_resampled = librosa.resample(y, orig_sr=sr, target_sr=target_sr)

# --- Informasi perbandingan ---
print("=== Perbandingan Audio Asli vs Resampled ===")
print(f"Sample rate asli    : {sr} Hz")
print(f"Sample rate baru    : {target_sr} Hz")
print(f"Durasi asli         : {len(y)/sr:.2f} detik")
print(f"Durasi resampled    : {len(y_resampled)/target_sr:.2f} detik")
print(f"Jumlah sampel asli  : {len(y)}")
print(f"Jumlah sampel baru  : {len(y_resampled)}")

# Putar audio original
print("▶️ Audio Original")
display(Audio(y, rate=sr))

# Putar audio hasil resampling
print("▶️ Audio Resampled")
display(Audio(y_resampled, rate=target_sr))

# --- Visualisasi Perbandingan Waveform ---
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
librosa.display.waveshow(y, sr=sr, alpha=0.7)
plt.title(f"Waveform Asli ({sr} Hz)")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")

plt.subplot(2, 1, 2)
librosa.display.waveshow(y_resampled, sr=target_sr, color='orange', alpha=0.7)
plt.title(f"Waveform Resampled ({target_sr} Hz)")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")

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

```
=== Perbandingan Audio Asli vs Resampled ===
Sample rate asli    : 48000 Hz
Sample rate baru    : 16000 Hz
Durasi asli         : 26.28 detik
Durasi resampled    : 26.28 detik
Jumlah sampel asli  : 1261568
Jumlah sampel baru  : 420523
▶️ Audio Original
```

0:00 / 0:26

```
▶️ Audio Resampled
```

0:00 / 0:26



- Penjelasan Analisis Original vs Resampling Pada waveform original kelihatan lebih rapat dan halus, karena jumlah sampel per detik lebih banyak. Sedangkan pada resampling, detail frekuensi tinggi hilang, jadi waveform kelihatan sedikit lebih kasar dan tidak sehalus yang 48 kHz.

Soal 2

- Rekam suara Anda berbicara di sekitar objek yang berisik (seperti kipas angin, AC, atau mesin).
  - Rekaman tersebut harus berdurasi kurang lebih 10 detik.
  - Rekam dalam format WAV (atau konversikan ke WAV sebelum dimuat ke notebook ini).
- 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

Membaca dan Memproses File Audio

Sebelum visualisasi, kita perlu membaca file WAV yang telah direkam untuk menampilkan informasi tentang file tersebut.

In [10]:
```
# Memuat Audio lain (misalnya Soal 2.wav)
PATH_AUDIO2 = os.path.join(os.getcwd(), 'audio', 'Knalpot.wav')

try:
    y2, sr2 = librosa.load(PATH_AUDIO2, sr=None)
    source_info2 = f"✅ Berhasil memuat: {PATH_AUDIO2}"
except Exception as e:
    print(f"❌ Error loading audio: {e}")
    raise

if y2.ndim > 1:
    y2 = np.mean(y2, axis=0)
    source_info2 += " - Dikonversi ke mono"

y2 = y2.astype(np.float32)

print("🎵 AUDIO LOADED")
print("=" * 40)
print(source_info2)
print(f"📊 Shape: {y2.shape}")
print(f"🎵 Sample rate: {sr2:,} Hz")
print(f"⏱️ Durasi: {len(y2)/sr2:.2f} detik")
print(f"📈 Amplitude range: {y2.min():.3f} to {y2.max():.3f}")

print("▶️ Audio: ")
display(Audio(y2, rate=sr))
```

```
🎵 AUDIO LOADED
========================================
✅ Berhasil memuat: e:\Semester 7\MULMED\Minggu 6\.venv\audio\Knalpot.wav
📊 Shape: (467968,)
🎵 Sample rate: 48,000 Hz
⏱️ Durasi: 9.75 detik
📈 Amplitude range: -0.953 to 0.967
▶️ Audio:
```

0:00 / 0:09

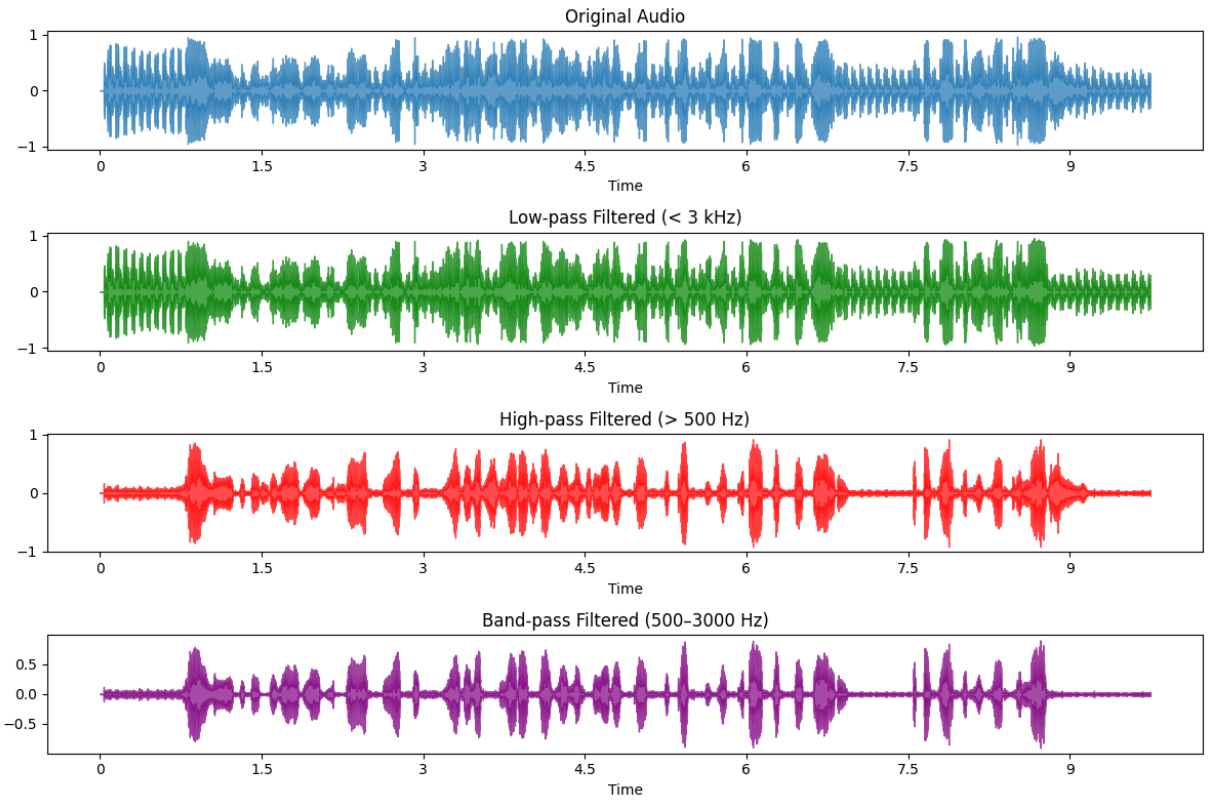
Filter Equalisasi (Low Pass, High Pass, Band Pass)

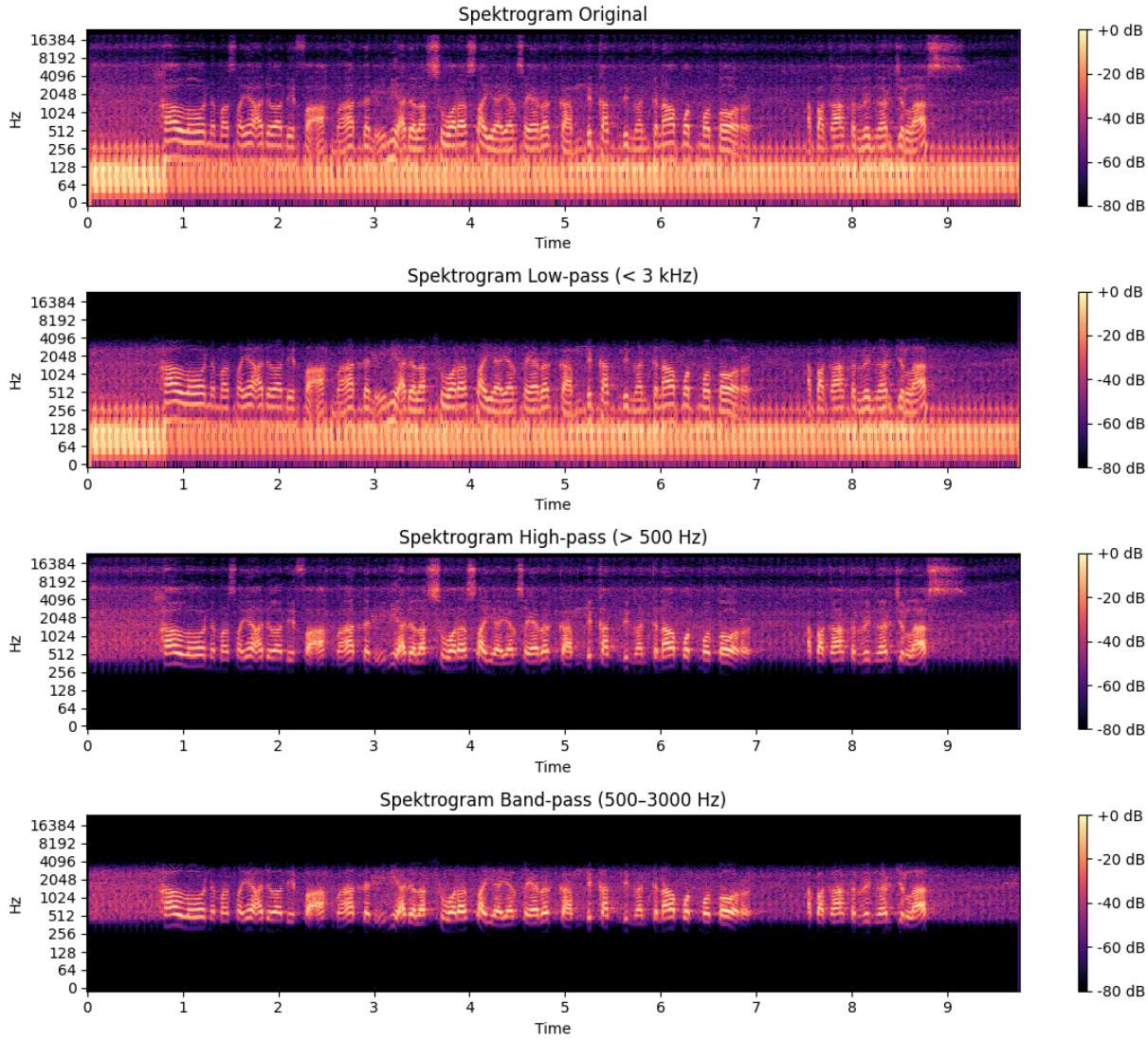
Sekarang kita akan melakukan filter equalisasi untuk menghilangkan noise pada rekaman ini.

In [23]:
```
import scipy.signal as signal

# --- Fungsi Helper untuk bikin filter ---
def butter_filter(data, cutoff, sr, btype='low', order=5):
    nyquist = 0.5 * sr
    normal_cutoff = np.array(cutoff) / nyquist
    b, a = signal.butter(order, normal_cutoff, btype=btype, analog=False)
    filtered_data = signal.filtfilt(b, a, data)
    return filtered_data

# --- Terapkan filter ---
```

```
# Low-pass filter (contoh: hanya biarkan frekuensi < 3000 Hz)
low_pass = butter_filter(y2, cutoff=3000, sr=sr2, btype='low')

# High-pass filter (contoh: hanya biarkan frekuensi > 500 Hz)
high_pass = butter_filter(y2, cutoff=500, sr=sr2, btype='high')

# Band-pass filter (contoh: hanya biarkan 500-3000 Hz)
band_pass = butter_filter(y2, cutoff=[500, 3000], sr=sr2, btype='band')

# --- Visualisasi hasil waveform ---
plt.figure(figsize=(12, 8))

plt.subplot(4, 1, 1)
librosa.display.waveshow(y2, sr=sr2, alpha=0.7)
plt.title("Original Audio")

plt.subplot(4, 1, 2)
librosa.display.waveshow(low_pass, sr=sr2, color='green', alpha=0.7)
plt.title("Low-pass Filtered (< 3 kHz)")

plt.subplot(4, 1, 3)
librosa.display.waveshow(high_pass, sr=sr2, color='red', alpha=0.7)
plt.title("High-pass Filtered (> 500 Hz)")

plt.subplot(4, 1, 4)
librosa.display.waveshow(band_pass, sr=sr2, color='purple', alpha=0.7)
plt.title("Band-pass Filtered (500-3000 Hz)")

plt.tight_layout()
plt.show()

# === Visualisasi Spektrogram Perbandingan ===
plt.figure(figsize=(12, 10))

# Spektrogram Original
D_orig = librosa.amplitude_to_db(np.abs(librosa.stft(y2)), ref=np.max)
plt.subplot(4, 1, 1)
librosa.display.specshow(D_orig, sr=sr2, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Original")

# Spektrogram Low-pass
D_low = librosa.amplitude_to_db(np.abs(librosa.stft(low_pass)), ref=np.max)
plt.subplot(4, 1, 2)
librosa.display.specshow(D_low, sr=sr2, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Low-pass (< 3 kHz)")

# Spektrogram High-pass
D_high = librosa.amplitude_to_db(np.abs(librosa.stft(high_pass)), ref=np.max)
plt.subplot(4, 1, 3)
librosa.display.specshow(D_high, sr=sr2, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram High-pass (> 500 Hz)")

# Spektrogram Band-pass
```

```
D_band = librosa.amplitude_to_db(np.abs(librosa.stft(band_pass)), ref=np.max)
plt.subplot(4, 1, 4)
librosa.display.specshow(D_band, sr=sr2, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Band-pass (500–3000 Hz)")

plt.tight_layout()
plt.show()

# --- Pemutar audio untuk membandingkan ---
print("▶️ Original Audio")
display(Audio(y2, rate=sr2))

print("▶️ Low-pass Filtered")
display(Audio(low_pass, rate=sr2))

print("▶️ High-pass Filtered")
display(Audio(high_pass, rate=sr2))

print("▶️ Band-pass Filtered")
display(Audio(band_pass, rate=sr2))
```





Original Audio

0:00 / 0:09

Low-pass Filtered

0:00 / 0:09

High-pass Filtered

0:00 / 0:09

Band-pass Filtered

0:00 / 0:09

Penjelasan dan Analisis

Dari keempat Spectogram (Original, Low-pass, High-pass, dan Band-pass) diatas dan hasil audio yang telah diproses, berikut adalah analisis:

- Pada Waveform original, Semua frekuensi rendah/sedang/tinggi masih ada sehingga bentuknya terlihat kompleks, banyak detail kecil dan rapat. Pada Low-pass filter bentuk waveform jadi lebih halus, tapi informasi utama seperti suara manusia masih terdengar jelas. Pada High-pass filter, bentuk waveform kelihatan lebih ramping dengan banyak detail kecil. Sedangkan pada waveform filter Band-pass, bentuknya gabungan low-pass dan high-pass, karena hanya mengambil frekuensi di rentang 500-3000 Hz.

- Pada spectrogram original, seluruh frekuensi dari rendah hingga tinggi masih terlihat sehingga audio terdengar penuh namun kompleks. Low-pass filter (<3 kHz) memotong frekuensi tinggi sehingga suara terdengar lebih redup, sedangkan high-pass filter (>500 Hz) menghilangkan bass dan membuat suara manusia lebih jelas. Band-pass filter (500–3000 Hz) menghasilkan visual paling fokus pada rentang inti ucapan, sehingga suara manusia terdengar paling jernih tanpa gangguan frekuensi rendah maupun tinggi.

- Pada audio low-pass suara terdengar redup dan terpendam, tidak ada suara frekuensi tinggi karena di cut-off di atas 3000Hz, suara bass knalpot motor masih terdengar jelas. Pada audio high-pass suara manusia terdengar jelas dan suara knalpot tidak terdengar bass-nya karena dicut-off dibawah 500Hz.

- Jenis noise yang muncul pada rekaman adalah suara knalpot yang cenderung ngebass dan bulat. Filter yang menurut saya efektif mengurangi noise tersebut adalah High-pass filter. Nilai cut-off yang efektif adalah 500 Hz untuk mengcut off suara-suara noise yang bass dan frekuensi rendah. Kualitas suara manusia setelah proses filter High-pass bass vokalnya sedikit redup tetapi masih terdengar jelas.

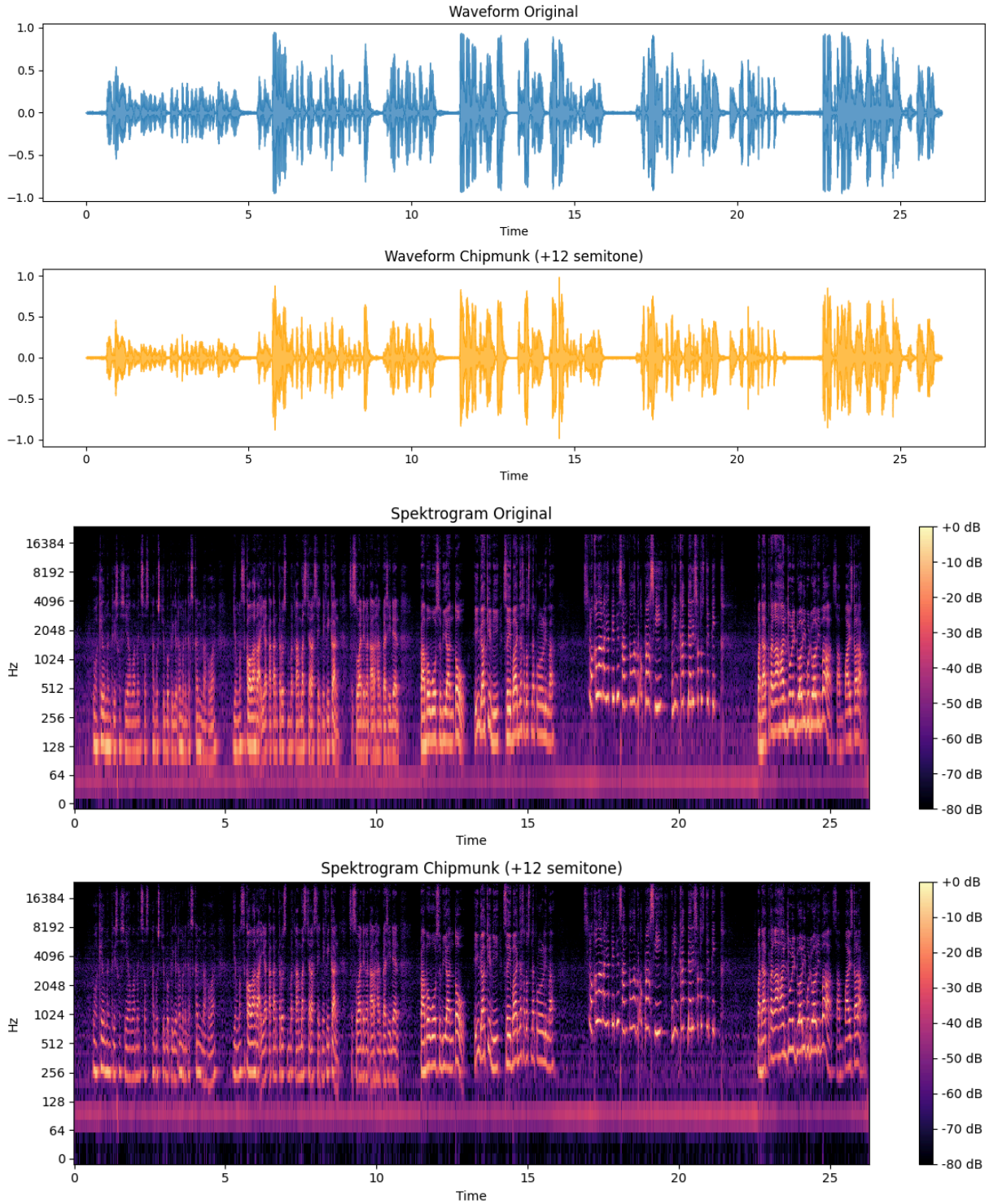
Soal 3

- 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 buat pitch tinggi, misalnya pitch +7 dan pitch +12.
- Gabungkan kedua rekaman yang telah di-pitch shift ke dalam satu file audio. (Gunakan ChatGPT / AI untuk membantu Anda dalam proses ini)

Proses Pitch Shifting

Kita akan melakukan pitch shifting pada rekaman suara Soal 1 untuk membuat suara terdengar seperti chipmunk

In [21]:
```
import librosa.effects

# Pitch shifting +12 semitone (1 oktaf naik → suara chipmunk)
y_chipmunk = librosa.effects.pitch_shift(y, sr=sr, n_steps=12)

# === Waveform perbandingan ===
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
librosa.display.waveshow(y, sr=sr, alpha=0.7)
plt.title("Waveform Original")

plt.subplot(2, 1, 2)
librosa.display.waveshow(y_chipmunk, sr=sr, color='orange', alpha=0.7)
plt.title("Waveform Chipmunk (+12 semitone)")

plt.tight_layout()
plt.show()

# === Spektrogram perbandingan ===
D_orig = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
D_chip = librosa.amplitude_to_db(np.abs(librosa.stft(y_chipmunk)), ref=np.max)

plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
librosa.display.specshow(D_orig, sr=sr, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Original")

plt.subplot(2, 1, 2)
librosa.display.specshow(D_chip, sr=sr, x_axis='time', y_axis='log', cmap='magma')
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Chipmunk (+12 semitone)")

plt.tight_layout()
plt.show()

# === Pemutar audio ===
print("▶️ Original Audio")
display(Audio(y, rate=sr))

print("▶️ Chipmunk Audio")
display(Audio(y_chipmunk, rate=sr))
```



Original Audio

0:00 / 0:26

Chipmunk Audio

0:00 / 0:26

Penjelasan dan Analisis

- Parameter yang digunakan: n_steps = 12, jumlah semitone yang digeser, tandanya naik 1 oktaf (menjadi suara chipmunk)
- Perbedaan dalam representasi visual antara suara asli dan suara yang telah dimodifikasi Secara bentuk, waveform chipmunk masih mirip dengan aslinya, tapi pola osilasinya jadi lebih rapat karena frekuensi naik. Jadi terlihat gelombangnya lebih padat walaupun pola besar tetap sama. Sedangkan pada spektogram, suara original, energi suara tersebar di frekuensi normal, pada chipmunk seluruh pola tergeser ke atas. Jadi bentuk dan harmonik suara naik ke frekuensi lebih tinggi, bikin suara kedengaran lebih melengking.
- Bagaimana perubahan pitch memengaruhi kualitas dan kejelasan suara Pada pitch naik, suara jadi lebih melengking, karakter asli berubah drastis. Semakin tinggi pergeseran pitch, semakin terasa buatan dan bisa mengurangi naturalitas suara.

Pitching 2 Audio +7 dan +12 dan gabungkan keduanya

In [22]:
```
import soundfile as sf
import librosa.effects

# --- Pitch shifting ---
y_pitch7 = librosa.effects.pitch_shift(y, sr=sr, n_steps=7)
y_pitch12 = librosa.effects.pitch_shift(y, sr=sr, n_steps=12)

# --- Simpan masing-masing file ---
sf.write("audio_pitch_plus7.wav", y_pitch7, sr)
sf.write("audio_pitch_plus12.wav", y_pitch12, sr)

print("✅ File audio hasil pitch shift disimpan: audio_pitch_plus7.wav & audio_pitc

# --- Gabungkan kedua rekaman ---
y_concat = np.concatenate([y_pitch7, y_pitch12])

# --- Simpan gabungan ---
sf.write("audio_pitch_combined.wav", y_concat, sr)
print("✅ File gabungan disimpan: audio_pitch_combined.wav")

# --- Pemutar audio ---
print("▶️ Pitch +7")
display(Audio(y_pitch7, rate=sr))

print("▶️ Pitch +12 (Chipmunk)")
display(Audio(y_pitch12, rate=sr))

print("▶️ Gabungan Pitch +7 lalu +12")
display(Audio(y_concat, rate=sr))
```

```
✅ File audio hasil pitch shift disimpan: audio_pitch_plus7.wav & audio_pitch_plus1
2.wav
✅ File gabungan disimpan: audio_pitch_combined.wav
▶️ Pitch +7
```

0:00 / 0:26

▶️ Pitch +12 (Chipmunk)

0:00 / 0:26

▶️ Gabungan Pitch +7 lalu +12

0:00 / 0:52

Soal 4

- Lakukan processing pada rekaman yang sudah di-pitch shift pada Soal 3 dengan tahapan:
  - Equalizer
  - gain/fade
  - Normalization
  - Compression
  - Noise Gate
  - Silence trimming
- Atur nilai target loudness ke -16 LUFS.
- 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

Tahap Processing

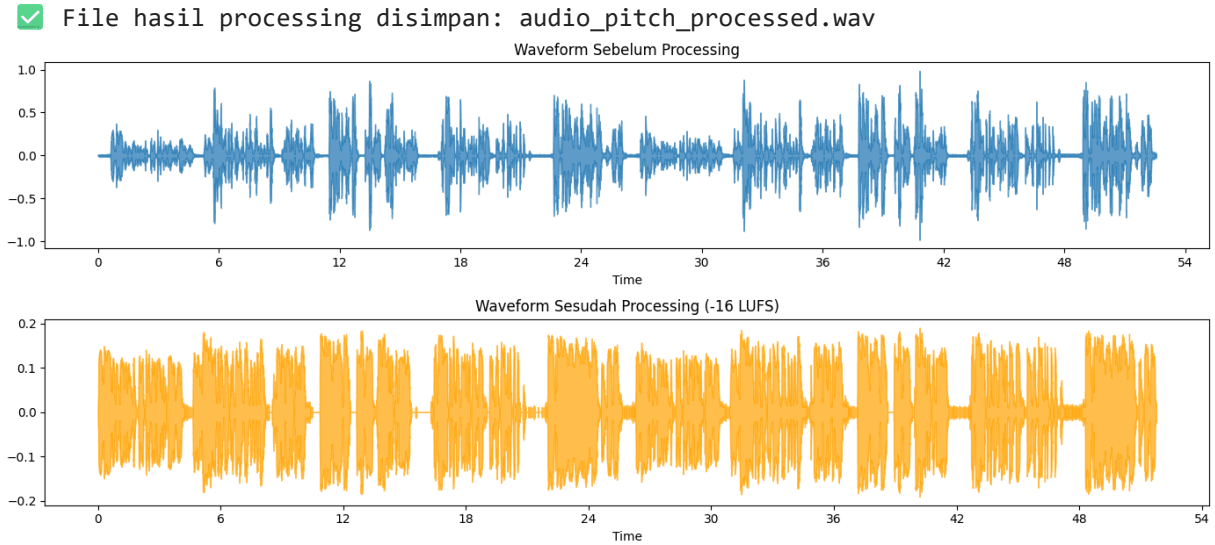
Pada tahap ini, rekaman yang sudah di pitch shift pada soal 3 akan dilakukan processing. Sebelum itu kita perlu menginstall library pylourdnom dengan perintah: pip install pyloudnorm

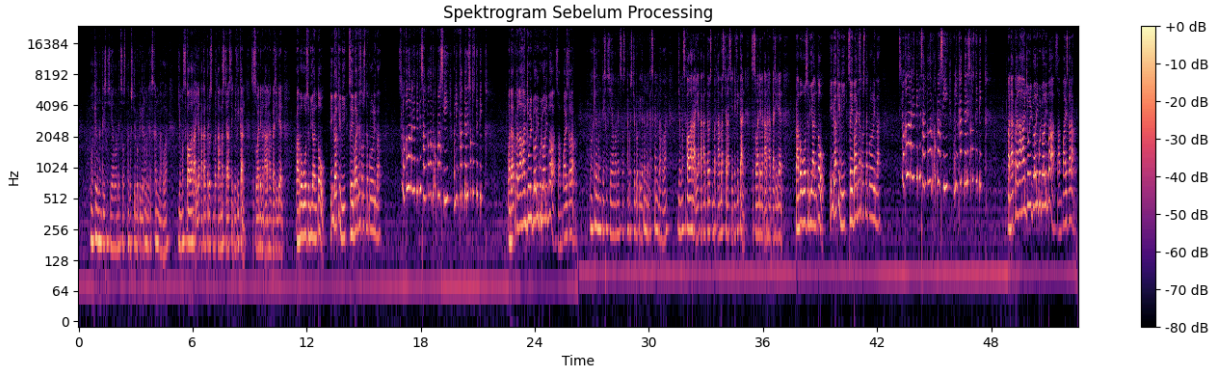
In [25]:
```
import librosa.display
import matplotlib.pyplot as plt
import pyloudnorm as pyln
import scipy.signal as signal

# === Load hasil gabungan dari Soal 3 ===
y_proc, sr_proc = librosa.load("audio_pitch_combined.wav", sr=None)

# --- 1. Equalizer (contoh band-pass 100–8000 Hz) ---
def butter_bandpass_filter(data, lowcut, highcut, sr, order=5):
    nyquist = 0.5 * sr
    low = lowcut / nyquist
```

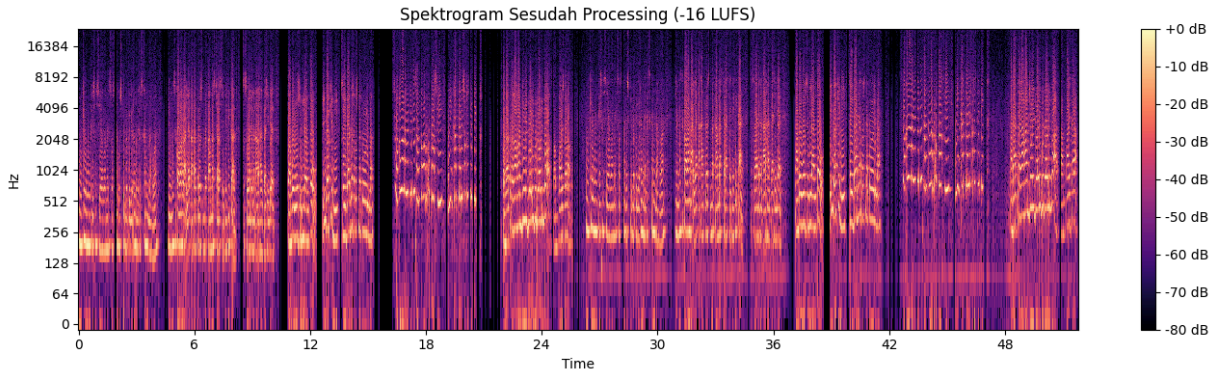
```
    high = highcut / nyquist
    b, a = signal.butter(order, [low, high], btype="band")
    return signal.filtfilt(b, a, data)

y_eq = butter_bandpass_filter(y_proc, 100, 8000, sr_proc)

# --- 2. Gain / Fade in & Fade out ---
fade_len = int(0.5 * sr_proc)  # 0.5 detik fade
fade_in = np.linspace(0., 1., fade_len)
fade_out = np.linspace(1., 0., fade_len)
y_gain = y_eq.copy()
y_gain[:fade_len] *= fade_in
y_gain[-fade_len:] *= fade_out

# --- 3. Normalization ke target LUFS (-16) ---
meter = pyln.Meter(sr_proc)  # Loudness meter
loudness = meter.integrated_loudness(y_gain)
y_norm = pyln.normalize.loudness(y_gain, loudness, -16.0)

# --- 4. Compression sederhana ---
def compressor(data, threshold=-20.0, ratio=4.0):
    # ubah ke dB
    data_db = 20 * np.log10(np.maximum(np.abs(data), 1e-10))
    over_threshold = data_db > threshold
    gain_reduction = (1 - 1/ratio) * (data_db - threshold)
    data_db[over_threshold] -= gain_reduction[over_threshold]
    # balik ke linear
    return np.sign(data) * (10**(data_db/20))

y_comp = compressor(y_norm)

# --- 5. Noise Gate sederhana ---
def noise_gate(data, threshold_db=-40.0):
    threshold = 10**(threshold_db/20)
    gated = np.where(np.abs(data) < threshold, 0, data)
    return gated

y_gate = noise_gate(y_comp)

# --- 6. Silence trimming ---
y_trim, _ = librosa.effects.trim(y_gate, top_db=30)

# === Simpan hasil processing ===
sf.write("audio_pitch_processed.wav", y_trim, sr_proc)
print("✅ File hasil processing disimpan: audio_pitch_processed.wav")

# --- Visualisasi waveform & spektrogram sebelum & sesudah normalisasi ---
plt.figure(figsize=(14, 6))

plt.subplot(2, 1, 1)
librosa.display.waveshow(y_proc, sr=sr_proc, alpha=0.7)
plt.title("Waveform Sebelum Processing")

plt.subplot(2, 1, 2)
librosa.display.waveshow(y_trim, sr=sr_proc, alpha=0.7, color="orange")
plt.title("Waveform Sesudah Processing (-16 LUFS)")
```

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

# Spektrogram
plt.figure(figsize=(14, 8))

D_before = librosa.amplitude_to_db(np.abs(librosa.stft(y_proc)), ref=np.max)
D_after = librosa.amplitude_to_db(np.abs(librosa.stft(y_trim)), ref=np.max)

plt.subplot(2, 1, 1)
librosa.display.specshow(D_before, sr=sr_proc, x_axis="time", y_axis="log", cmap="m
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Sebelum Processing")

plt.subplot(2, 1, 2)
librosa.display.specshow(D_after, sr=sr_proc, x_axis="time", y_axis="log", cmap="ma
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram Sesudah Processing (-16 LUFS)")

plt.tight_layout()
plt.show()

# --- Pemutar Audio ---
print("▶️ Audio Sebelum Processing")
display(Audio(y_proc, rate=sr_proc))

print("▶️ Audio Sesudah Processing")
display(Audio(y_trim, rate=sr_proc))
```

```
e:\Semester 7\MULMED\Minggu 6\.venv\Lib\site-packages\pyloudnorm\normalize.py:62: Us
erWarning: Possible clipped samples in output.
  warnings.warn("Possible clipped samples in output.")
✅ File hasil processing disimpan: audio_pitch_processed.wav
```







Audio Sebelum Processing

0:00 / 0:52

Audio Sesudah Processing

0:00 / 0:51

Penjelasan dan Analisis

- Perubahan dinamuka suara yang terjadi: Audio1 terdengar mentah: masih ada noise, level volume kurang konsisten, dan ada bagian hening yang panjang. Audio2 terdengar lebih bagus: jernih, fokus pada frekuensi penting, volume rata, transisi halus, noise berkurang, dan tanpa hening berlebihan. Pada waveform sebelum processing, amplitudo terlihat bervariasi dengan perbedaan jelas antara bagian pelan dan keras, serta masih ada jeda hening yang cukup panjang. Setelah diproses dengan normalisasi ke -16 LUFS, waveform menjadi lebih padat dan rata, menunjukkan bahwa volume sudah diseimbangkan sehingga bagian pelan terdengar lebih jelas tanpa terlalu jauh tertinggal dari bagian keras. Sementara pada spectrogram sebelum processing, distribusi energi frekuensi masih tidak konsisten, bagian rendah (bass/noise) cukup dominan dan terdapat perbedaan intensitas antar segmen. Setelah diproses, spectrogram sesudah processing menunjukkan distribusi energi yang lebih merata di seluruh durasi audio, frekuensi penting lebih menonjol, noise berkurang, dan jeda hening dipangkas sehingga suara terdengar lebih stabil, bersih, dan konsisten.

- Perbedaan antara normalisasi peak dan normalisasi LUFS adalah: Normalisasi Peak: Menyetel sinyal audio supaya puncak tertinggi (peak amplitude) mencapai nilai target tertentu (misalnya 0 dBFS). Ciri-cirinya adalah tidak peduli rata-rata keras pelannya suara. Kalau audio banyak bagian pelan, tetap saja hasil akhirnya terdengar pelan walaupun puncaknya sudah menyentuh target. Normalisasi LUFS: Menyesuaikan level audio berdasarkan persepsi loudness manusia. Lebih konsisten didengar telinga, karena memperhitungkan frekuensi dan durasi (tidak hanya puncak).
- Bagaimana kualitas suara berubah setelah proses normalisasi dan loudness optimization? Sebelum normalisasi/loudness optimization: Level audio tidak konsisten, ada bagian yang terlalu pelan dan ada bagian yang terlalu keras. Puncak amplitudo mungkin sudah tinggi, tapi rata-rata loudness masih rendah. Setelah Normalisasi Peak saja: Suara jadi lebih besar karena puncak tertinggi diset ke batas aman (misalnya 0 dBFS).Tapi bagian yang pelan tetap saja masih pelan, jadi tidak selalu terasa lebih jelas. Setelah Normalisasi LUFS: Rata-rata loudness sudah disesuaikan. Semua bagian terdengar lebih seimbang, bagian pelan naik sedikit, bagian keras terasa lebih terkontrol.
- Kelebihan dan kekurangan dari pengoptimalan loudness dalam konteks rekaman suara Kelebihan dari pengoptimalan loudness adalah mempertahankan konsistensi volume, meningkatkan kenyamanan pendengar, dan mengurangi perbedaan antar perangkat. Sedangkan untuk kekurangan adalah hilangnya dinamika alami karena terlalu banyak normalisasi/kompresi, resiko distorsi, dan karakter asli suara bisa berubah.

Soal 5

- Pilih 2 buah (potongan) lagu yang memiliki vokal (penyanyi) dan berdurasi sekitar 1 menit:
  - Lagu 1: Nuansa sedih, lambat
  - Lagu 2: Nuansa ceria, cepat
- Konversikan ke format WAV sebelum dimuat ke notebook ini.
- 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

Tampilkan informasi kedua audio

Sebelum melakukan langkah selanjutnya, kita perlu memotong durasi kedua audio menjadi 1 menit. Setelah itu menampilkan informasi audio seperti durasi, sample rate, dan shape.

In [ ]:
```
import os
import numpy as np
import librosa
from IPython.display import Audio, display

# Memuat Audio 1
PATH_AUDIO = os.path.join(os.getcwd(), 'audio', 'Ashes of Dreams.wav')

try:
    y, sr = librosa.load(PATH_AUDIO, sr=None)  # sr=None mempertahankan sample rate
    source_info = f"✅ Berhasil memuat: {PATH_AUDIO}"
except Exception as e:
    print(f"❌ Error loading audio: {e}")
    raise

if y.ndim > 1:
    y = np.mean(y, axis=0)
    source_info += " - Dikonversi ke mono"

y = y.astype(np.float32)

# Potong hanya 1 menit (60 detik)
max_len = 60 * sr
y = y[:max_len]

# Memuat Audio 2
PATH_AUDIO2 = os.path.join(os.getcwd(), 'audio', 'Hills of Radiant Wind.wav')

try:
    y2, sr2 = librosa.load(PATH_AUDIO2, sr=None)
    source_info2 = f"✅ Berhasil memuat: {PATH_AUDIO2}"
except Exception as e:
    print(f"❌ Error loading audio: {e}")
    raise

if y2.ndim > 1:
    y2 = np.mean(y2, axis=0)
    source_info2 += " - Dikonversi ke mono"

y2 = y2.astype(np.float32)

# Potong hanya 1 menit (60 detik)
max_len2 = 60 * sr2
y2 = y2[:max_len2]

# === Tampilkan Informasi ===
print("🎵 AUDIO LOADED 1 - Lagu Sedih: Ashes of Dreams")
print("=" * 40)
print(source_info)
print(f"📊 Shape: {y.shape}")
print(f"🎵 Sample rate: {sr:,} Hz")
print(f"⏱️ Durasi: {len(y)/sr:.2f} detik (setelah dipotong)")
```

```
print(f"📈 Amplitude range: {y.min():.3f} to {y.max():.3f}")

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

print("🎵 AUDIO LOADED 2 - Lagu Ceria: Hills of Radiant Wind")
print("=" * 40)
print(source_info2)
print(f"📊 Shape: {y2.shape}")
print(f"🎵 Sample rate: {sr2:,} Hz")
print(f"⏱️ Durasi: {len(y2)/sr2:.2f} detik (setelah dipotong)")
print(f"📈 Amplitude range: {y2.min():.3f} to {y2.max():.3f}")

print("▶️ Audio: ")
display(Audio(y2, rate=sr2))
```

```
🎵 AUDIO LOADED 1
========================================
✅ Berhasil memuat: e:\Semester 7\MULMED\Minggu 6\.venv\audio\Ashes of Dreams.wav
📊 Shape: (2646000,)
🎵 Sample rate: 44,100 Hz
⏱️ Durasi: 60.00 detik (setelah dipotong)
📈 Amplitude range: -0.668 to 0.609
▶️ Audio:
```

0:00 / 1:00

```
🎵 AUDIO LOADED 2
========================================
✅ Berhasil memuat: e:\Semester 7\MULMED\Minggu 6\.venv\audio\Hills of Radiant Win
d.wav
📊 Shape: (2646000,)
🎵 Sample rate: 44,100 Hz
⏱️ Durasi: 60.00 detik (setelah dipotong)
📈 Amplitude range: -0.903 to 0.922
▶️ Audio:
```

0:00 / 1:00

Deteksi Tempo dan Estimasi Kunci

Setelah menampilkan informasi audio, kita perlu mendeteksi tempo audio dan estimasi kunci dari kedua audio tersebut.

In [15]:

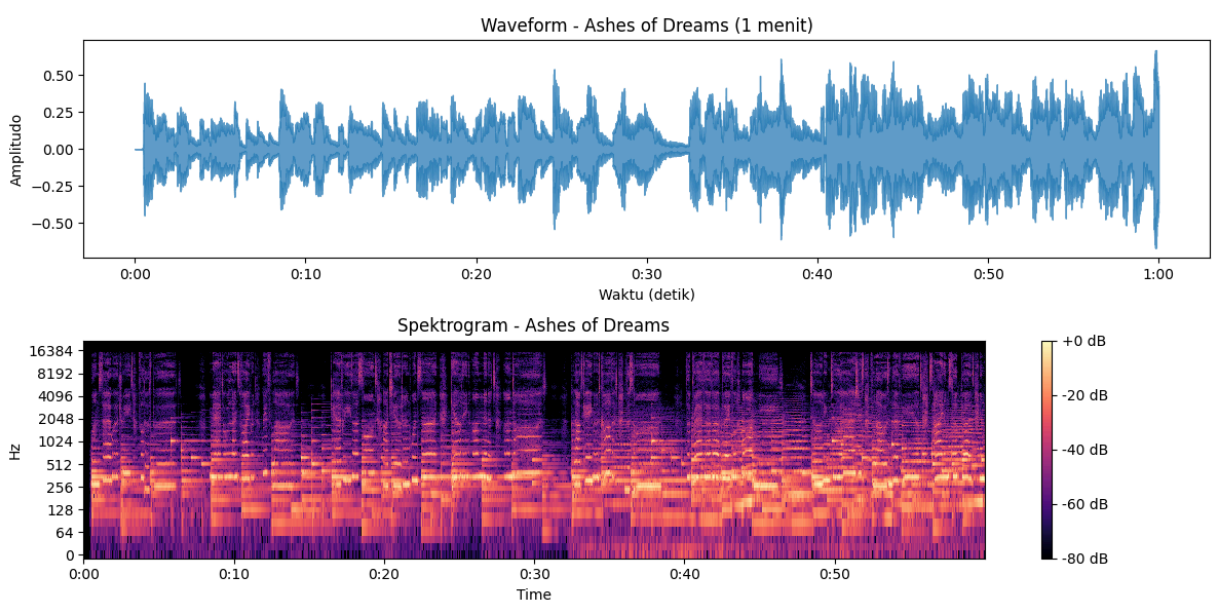
```
import os
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
from IPython.display import Audio, display

# === Fungsi Estimasi Key (mayor/minor sederhana) ===
```

```python
def estimate_key(y, sr):
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_mean = chroma.mean(axis=1)

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

    # Pitch class dominan
    key_index = chroma_mean.argmax()
    root = notes[key_index]

    # Periksa apakah lebih cenderung mayor/minor
    major_third = (key_index + 4) % 12
    minor_third = (key_index + 3) % 12
    if chroma_mean[minor_third] > chroma_mean[major_third]:
        return f"{root} Minor"
    else:
        return f"{root} Major"

# === Load Audio 1 ===
PATH_AUDIO = os.path.join(os.getcwd(), 'audio', 'Ashes of Dreams.wav')
y, sr = librosa.load(PATH_AUDIO, sr=None, mono=True, duration=60)  # potong 1 menit

# === Load Audio 2 ===
PATH_AUDIO2 = os.path.join(os.getcwd(), 'audio', 'Hills of Radiant Wind.wav')
y2, sr2 = librosa.load(PATH_AUDIO2, sr=None, mono=True, duration=60)  # potong 1 me

# === Analisis Lagu 1 ===
tempo1, _ = librosa.beat.beat_track(y=y, sr=sr)
tempo1 = float(np.atleast_1d(tempo1)[0])  # ambil nilai scalar aman
key1 = estimate_key(y, sr)

print("=== Analisis Lagu 1: Ashes of Dreams ===")
print(f"Tempo (BPM): {tempo1:.2f}")
print(f"Estimasi Key: {key1}")
display(Audio(y, rate=sr))

# Visualisasi waveform & spektrogram lagu 1
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
librosa.display.waveshow(y, sr=sr, alpha=0.7)
plt.title("Waveform - Ashes of Dreams (1 menit)")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")

plt.subplot(2, 1, 2)
D1 = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
librosa.display.specshow(D1, sr=sr, x_axis="time", y_axis="log", cmap="magma")
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram - Ashes of Dreams")

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

```python
# === Analisis Lagu 2 ===
tempo2, _ = librosa.beat.beat_track(y=y2, sr=sr2)
tempo2 = float(np.atleast_1d(tempo2)[0])
key2 = estimate_key(y2, sr2)

print("\n=== Analisis Lagu 2: Hills of Radiant Wind ===")
print(f"Tempo (BPM): {tempo2:.2f}")
print(f"Estimasi Key: {key2}")
display(Audio(y2, rate=sr2))

# Visualisasi waveform & spektrogram lagu 2
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
librosa.display.waveshow(y2, sr=sr2, color='orange', alpha=0.7)
plt.title("Waveform - Hills of Radiant Wind (1 menit)")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")

plt.subplot(2, 1, 2)
D2 = librosa.amplitude_to_db(np.abs(librosa.stft(y2)), ref=np.max)
librosa.display.specshow(D2, sr=sr2, x_axis="time", y_axis="log", cmap="magma")
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram - Hills of Radiant Wind")

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

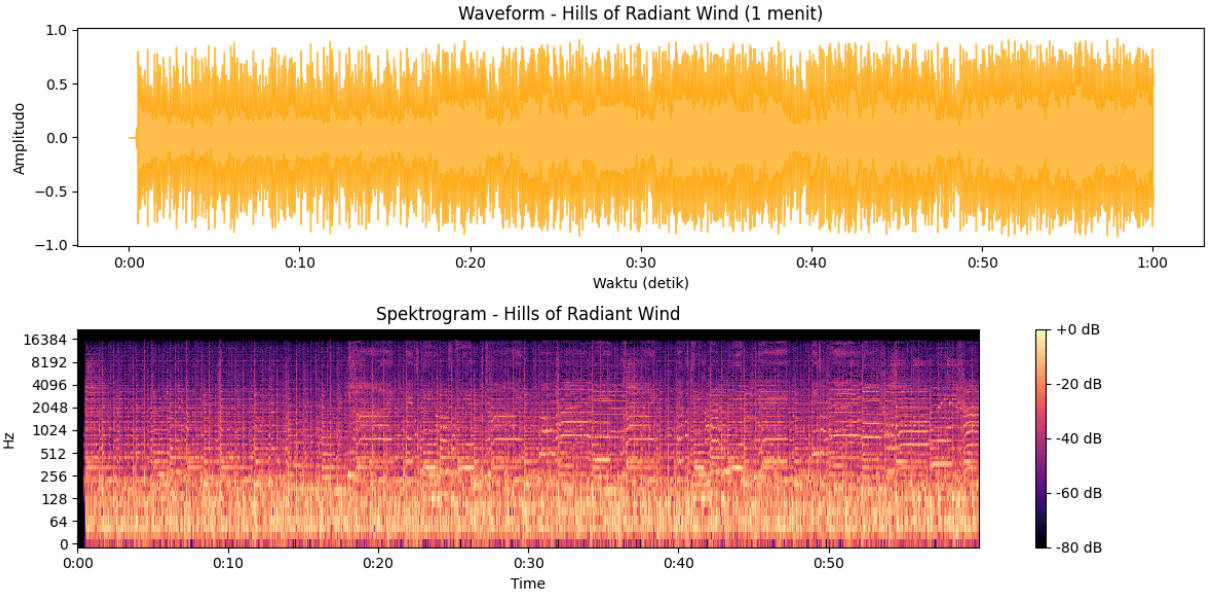
```
=== Analisis Lagu 1: Ashes of Dreams ===
Tempo (BPM): 114.84
Estimasi Key: F Minor
```

0:00 / 1:00




```
=== Analisis Lagu 2: Hills of Radiant Wind ===
Tempo (BPM): 161.50
Estimasi Key: G Major
```

0:00 / 1:00



Penjelasan dan Analisis

- Lagu 1: Ashes of Dreams Lagu ini memiliki tempo 114,84 BPM, termasuk dalam tempo sedang, sesuai dengan nuansa lagu yang lebih emosional dan dramatis. Lagu ini dimulai dari F Minor, cocok dengan lagu yang melankolis dan emosional dalam suasana sendu namun tragis. Visualisasi waveform lagu ini menampilkan variasi amplitudo yang dinamis, menampilkan bagian keras dan pelan secara bergantian Visualisasi spectogram lagu ini menampilkan frekuensi rendah dan menengah lebih dominan. Hal ini sesuai dengan vokal yang memberikan kedalaman emosional.
- Lagu 2: Hills of Radiant Wind Lagu ini memiliki tempo 161.50 BPM, tergolong cepat, memberikan energi yang lebih bersemangat dengan suasana protagonis hero era middleage. Lagu ini dimulai dari G Mayor, cocok dengan nuansa yang cerah, petualang, optimis, seolah olah berada di hamparan padang rumput yang luas. Visualisasi waveform lagu ini menampilkan amplitudo yang terlihat padat dan konstan hampir sepanjang durasi, menandakan intensitas musik yang stabil dan terus mengalir. Visualisasi spectogram lagu ini menampilkan persebaran energi yang merata hingga frekuensi tinggi, memperlihatkan instrumen yang kaya dan tekstur suara yang lebih penuh.

Proses Remix terhadap kedua lagu

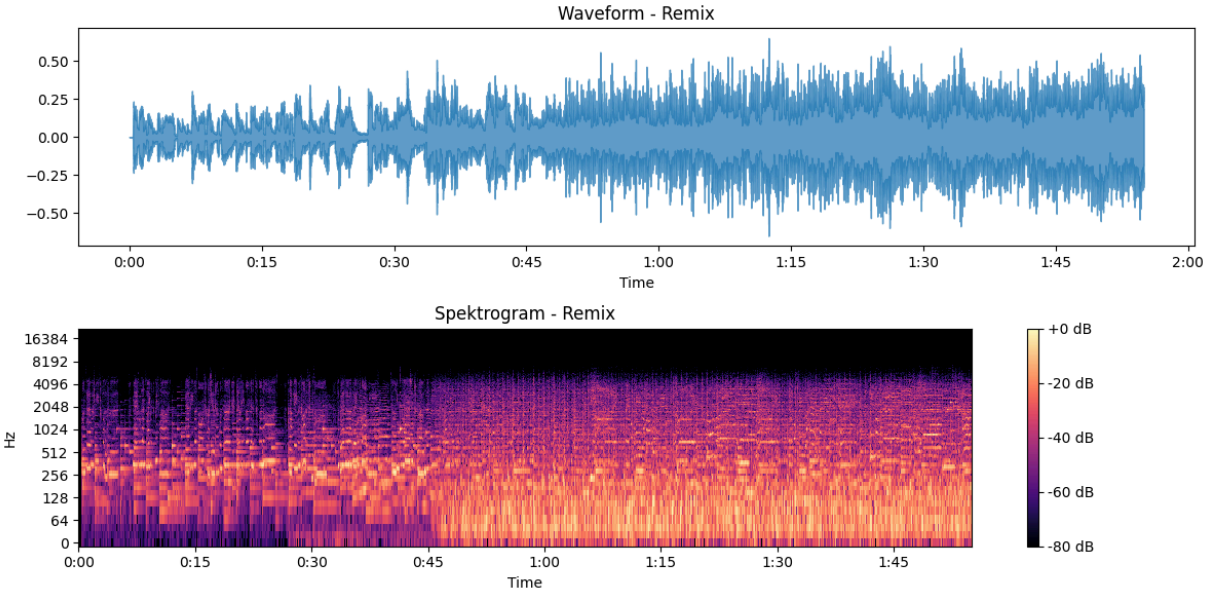
- Time Strech: Sesuaikan tempo kedua lagu supaya sama, yaitu tempo baru = rata-rata keduanya
- Pitch Shift: Samakan kunci, kunci yang dipilih adalah lagu pertama (F Minor). Lagu kedua akan dipitch shift agar selaras dengan lagu pertama

- Crossfading: Kita gabungkan kedua lagu dengan lagu pertama diakhirnya fade-out dan lagu kedua diawalnya fade-in supaya transisi lebih mulus.
- Filter Tambahan (Low Pass): untuk intro sebagai efek transisi

In [17]:
```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
from IPython.display import Audio, display

# === Time-stretch ke tempo target ===
target_bpm = (tempo1 + tempo2) / 2
rate1 = target_bpm / tempo1
rate2 = target_bpm / tempo2

y1_stretch = librosa.effects.time_stretch(y, rate=rate1)
y2_stretch = librosa.effects.time_stretch(y2, rate=rate2)

print(f"Target Tempo: {target_bpm:.2f} BPM")
print(f"Lagu 1 di-stretch dengan rate {rate1:.2f}")
print(f"Lagu 2 di-stretch dengan rate {rate2:.2f}")

# === Pitch shift ke key yang sama (misal F Minor dari lagu 1) ===
# Key mapping sederhana ke indeks nada
notes = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
def note_to_index(note):
    return notes.index(note.split()[0])

root1 = note_to_index(key1)
root2 = note_to_index(key2)

shift_steps = root1 - root2  # geser lagu 2 ke kunci lagu 1
y2_shift = librosa.effects.pitch_shift(y2_stretch, sr=sr2, n_steps=shift_steps)


print(f"Lagu 2 digeser {shift_steps:+} semitone agar sama key dengan Lagu 1 ({key1}

# === Crossfade kedua lagu ===
fade_len = int(5 * sr)  # 5 detik crossfade
len1 = len(y1_stretch)
len2 = len(y2_shift)

# Siapkan buffer gabungan
combined_len = len1 + len2 - fade_len
y_mix = np.zeros(combined_len)

# Masukkan lagu 1
y_mix[:len1] += y1_stretch

# Masukkan lagu 2 dengan crossfade
fade_in = np.linspace(0, 1, fade_len)
fade_out = np.linspace(1, 0, fade_len)

y_mix[len1-fade_len:len1] *= fade_out
```

```
y_mix[len1-fade_len:len1] += y2_shift[:fade_len] * fade_in
y_mix[len1:] += y2_shift[fade_len:]

# === Filter tambahan: Low-pass di transisi ===
import scipy.signal as signal
b, a = signal.butter(4, 0.2, btype='low')
y_mix = signal.filtfilt(b, a, y_mix)

# Simpan hasil remix
sf.write("remix_output.wav", y_mix, sr)
print("✅ Hasil remix disimpan: remix_output.wav")

# === Visualisasi ===
plt.figure(figsize=(12, 6))

plt.subplot(2,1,1)
librosa.display.waveshow(y_mix, sr=sr, alpha=0.7)
plt.title("Waveform - Remix")

plt.subplot(2,1,2)
D_mix = librosa.amplitude_to_db(np.abs(librosa.stft(y_mix)), ref=np.max)
librosa.display.specshow(D_mix, sr=sr, x_axis="time", y_axis="log", cmap="magma")
plt.colorbar(format="%+2.f dB")
plt.title("Spektrogram - Remix")

plt.tight_layout()
plt.show()

# Pemutar audio
print("▶️ Remix Result")
display(Audio(y_mix, rate=sr))
```

```
Target Tempo: 138.17 BPM
Lagu 1 di-stretch dengan rate 1.20
Lagu 2 di-stretch dengan rate 0.86
Lagu 2 digeser -2 semitone agar sama key dengan Lagu 1 (F Minor)
✅ Hasil remix disimpan: remix_output.wav
```




```
▶️ Remix Result
```

0:00 / 1:55

Penjelasan dan Analisis

- Tempo: Kedua lagu sekarang memiliki tempo yang sama yaitu di 160 BPM
- Key: Lagu kedua dipitch shift agar sesuai dengan lagu pertama
- Crossfade: Transisi antara lagu pertama ke kedua lebih halus dengan 5 detik overlap
- Visualisasi Waveform: Setelah time stretch, kedua lagu disesuaikan ke tempo rata-rata, sehingga bentuk alurnya sinkron. Pada bagian akhir lagu 1 terjadi proses fadeout dan lagu 2 proses fade in. Amplitudo keseluruhan lebih rata, menandakan gabungan kedua lagu sudah menyatu.
- Visualisasi Spectogram: Pada bagian transisi crossfade, ada overlap spektrum dari kedua lagu. Frekuensi menengah dan tinggi bercampur tapi tetap halus karena ada peredaman secara gradual (bertahap). Efek low pass filter di awal remix membuat bagian intro lebih gelap, sehingga transisi terdengar lebih lembut sebelum energi penuh muncul.

Informasi Lagu setelah diremix

In [18]:
```
import soundfile as sf
import librosa
import numpy as np

# === Load file remix ===
filename = "remix_output.wav"
y_remix, sr_remix = librosa.load(filename, sr=None, mono=True)

# Info dasar
num_samples = len(y_remix)
duration = num_samples / sr_remix
channels = 1 if y_remix.ndim == 1 else y_remix.shape[1]

print("🎵 INFO LAGU REMIX")
print("=" * 40)
print(f"Nama file     : {filename}")
print(f"Sample rate   : {sr_remix} Hz")
print(f"Durasi        : {duration:.2f} detik")
print(f"Jumlah sample : {num_samples}")
print(f"Jumlah kanal  : {channels} ({'Mono' if channels==1 else 'Stereo'})")
print(f"Amplitude range: {y_remix.min():.3f} to {y_remix.max():.3f}")

# === Estimasi Tempo & Key ===
tempo_remix, _ = librosa.beat.beat_track(y=y_remix, sr=sr_remix)
tempo_remix = float(np.atleast_1d(tempo_remix)[0])

# Key sederhana (pakai fungsi sebelumnya)
def estimate_key(y, sr):
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_mean = chroma.mean(axis=1)
    notes = ['C', 'C#', 'D', 'D#', 'E', 'F',
```

```
             'F#', 'G', 'G#', 'A', 'A#', 'B']
    key_index = chroma_mean.argmax()
    root = notes[key_index]
    major_third = (key_index + 4) % 12
    minor_third = (key_index + 3) % 12
    if chroma_mean[minor_third] > chroma_mean[major_third]:
        return f"{root} Minor"
    else:
        return f"{root} Major"

key_remix = estimate_key(y_remix, sr_remix)

print(f"Estimasi Tempo: {tempo_remix:.2f} BPM")
print(f"Estimasi Key  : {key_remix}")
```

```
🎵 INFO LAGU REMIX
========================================
Nama file     : remix_output.wav
Sample rate   : 44100 Hz
Durasi        : 115.00 detik
Jumlah sample : 5071500
Jumlah kanal  : 1 (Mono)
Amplitude range: -0.569 to 0.649
Estimasi Tempo: 136.00 BPM
Estimasi Key  : F Minor
```

SUMBER:

Link Konversasi dengan AI: LINK