

Nama : Dina Rahma Dita

NIM : 122140184

Mata Kuliah : Sistem Teknologi Multimedia

Dosen Pengampu : Martin Clinton Tosima Manullang, Ph.D.

Repository Github : [kuliah-multimedia-25-26](#)

Alat Bantu : <https://chatgpt.com/share/68f027a5-eb9c-8008-b316-202a685f6191>

In [1]:

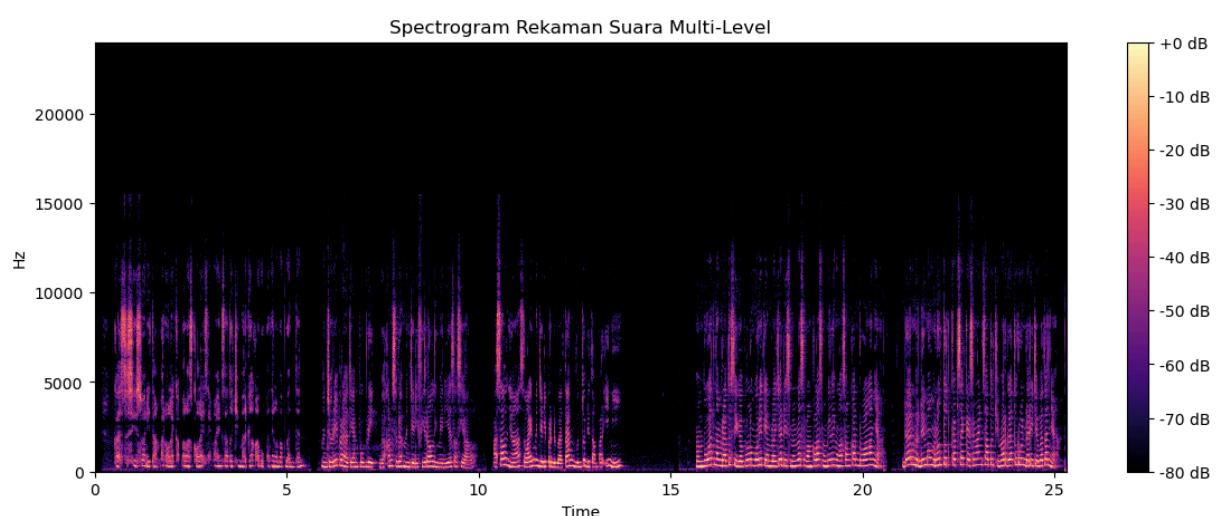
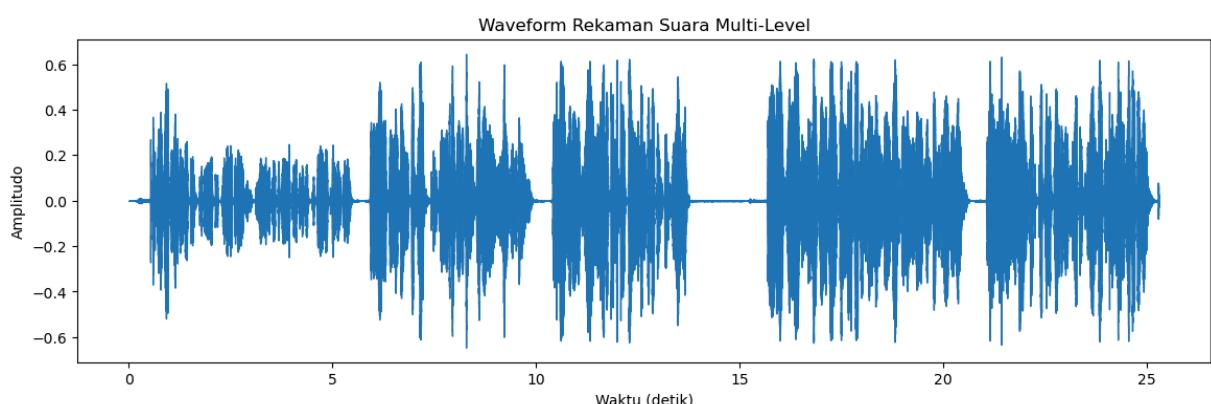
```
# =====
# Soal 1 - Rekaman dan Analisis Suara Multi-Level
# =====

import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

# 1. Load audio
audio_path = "audio.wav"
y, sr = librosa.load(audio_path, sr=None)

# 2. Visualisasi waveform
plt.figure(figsize=(14, 4))
librosa.display.waveform(y, sr=sr)
plt.title("Waveform Rekaman Suara Multi-Level")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.show()

# 3. Visualisasi spectrogram
plt.figure(figsize=(14, 5))
S = librosa.stft(y)
S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%.2f dB')
plt.title("Spectrogram Rekaman Suara Multi-Level")
plt.show()
```



Kesimpulan :

1. Berdasarkan waveform, amplitudo pada 5 detik awal tampak kecil karena suara diucapkan sangat pelan. Setelah detik ke-15, amplitudo meningkat tajam dan menjadi sangat rapat — menandakan volume suara yang tinggi (teriak). Ruang kosong di sekitar detik ke-15 menunjukkan jeda tanpa suara.
2. Berdasarkan spectrogram, terlihat bahwa energi paling kuat berada dalam rentang 0 - 10.000 Hz yang ditandai dengan tersebarnya warna cerah, terdapat area hitam yang menunjukkan tidak ada suara atau senyap pada detik tersebut

```
In [2]: from IPython.display import Audio, display

# Resampling ke 16 kHz
y_resampled = librosa.resample(y, orig_sr=sr, target_sr=16000)

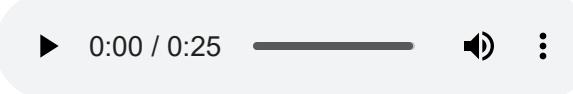
# Simpan hasil resampling (opsional)
import soundfile as sf
sf.write("audio_resampled.wav", y_resampled, 16000)

# Bandingkan durasi dan visualisasi
print("Sample rate awal:", sr)
print("Sample rate baru:", 16000)
print("Durasi awal:", len(y)/sr, "detik")
print("Durasi baru:", len(y_resampled)/16000, "detik")

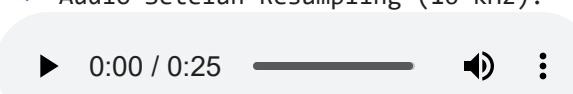
print("◆ Audio Asli:")
display(Audio(y, rate=sr))

print("◆ Audio Setelah Resampling (16 kHz):")
display(Audio(y_resampled, rate=16000))
```

Sample rate awal: 48000
 Sample rate baru: 16000
 Durasi awal: 25.30133333333332 detik
 Durasi baru: 25.301375 detik
 ◆ Audio Asli:



◆ Audio Setelah Resampling (16 kHz):



Setelah dilakukan resampling ke 16 KHz, kualitas suara terdengar sedikit turun tidak sejernih dari audio awal, namun suara tetap terdengar dengan jelas dan tidak terlalu mengganggu, sedangkan untuk durasi tidak mengalami perubahan signifikan dengan selisih kurang dari 0.001 detik

```
In [3]: # =====
# Soal 2 - Noise Reduction dengan Filtering
# =====

import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import butter, filtfilt
import soundfile as sf

# Load audio
audio_path = "audio2.wav"
y, sr = librosa.load(audio_path, sr=None)
print("◆ Audio Asli:")
display(Audio(y, rate=sr))

# Tampilkan waveform
plt.figure(figsize=(14, 4))
librosa.display.waveform(y, sr=sr)
```

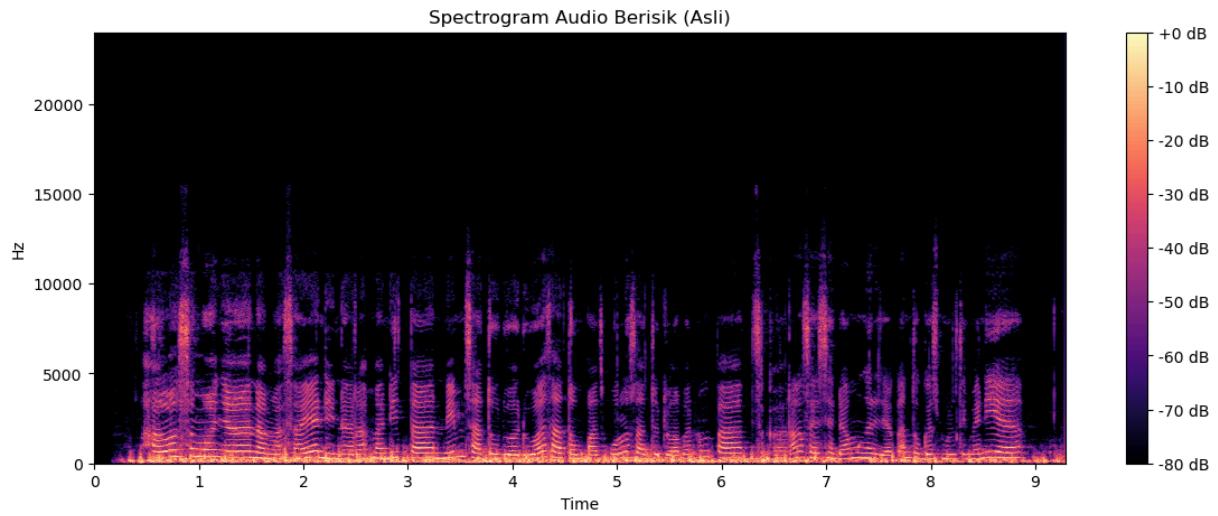
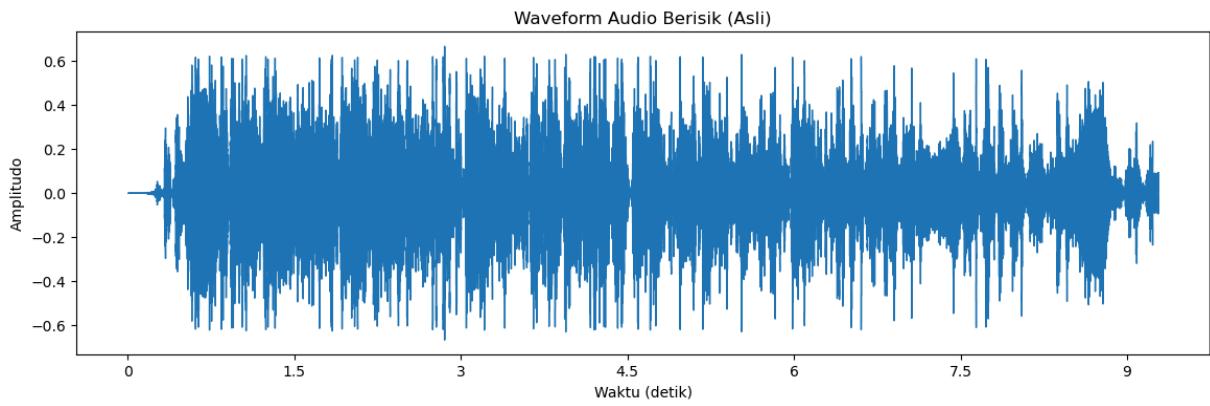
```

plt.title("Waveform Audio Berisik (Asli)")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.show()

# Tampilkan spectrogram
plt.figure(figsize=(14, 5))
S = librosa.stft(y)
S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%+2.0f dB')
plt.title("Spectrogram Audio Berisik (Asli)")
plt.show()

```

◆ Audio Asli:



In [7]:

```

# =====
# Pemutaran Audio Hasil Filtering
# =====
from IPython.display import Audio, display
from scipy.signal import butter, filtfilt
import numpy as np

# --- Fungsi bantu: normalisasi biar volume tidak terlalu kecil
def normalize_audio(data):
    return data / np.max(np.abs(data)) if np.max(np.abs(data)) != 0 else data

# --- Fungsi bantu: filter audio ---
def butter_filter(data, cutoff, fs, btype='low', order=6):
    nyq = 0.5 * fs
    if btype == 'band':
        normal_cutoff = [c / nyq for c in cutoff]
    else:
        normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype=btype, analog=False)
    y = filtfilt(b, a, data)
    return y

# --- Terapkan filter utama dulu (biar gak NameError) ---
low_500 = butter_filter(y, 500, sr, 'low', order=6)
high_1000 = butter_filter(y, 1000, sr, 'high', order=6)

```

```

band_500_2000 = butter_filter(y, [500, 2000], sr, 'band', order=6)

# --- Daftar hasil filter yang ingin diputar ---
versions = {
    "Asli": y,
    "Low-pass 500 Hz": low_500,
    "High-pass 1000 Hz": high_1000,
    "Band-pass 500-2000 Hz": band_500_2000,
}

# --- (Opsional) Tambahkan versi eksperimen lain ---
band_400_2500 = butter_filter(y, [400, 2500], sr, 'band', order=6)
band_600_2200 = butter_filter(y, [600, 2200], sr, 'band', order=6)

# Normalisasi dan tambahkan ke daftar
versions["Band-pass 400-2500 Hz"] = normalize_audio(band_400_2500)
versions["Band-pass 600-2200 Hz"] = normalize_audio(band_600_2200)

# --- Putar semua versi secara berurutan ---
for name, data in versions.items():
    print(f"🎧 {name}:")
    data_norm = normalize_audio(data)
    display(Audio(data_norm, rate=sr))

```

🎧 Asli:

▶ 0:00 / 0:09 🔊 ⋮

🎧 Low-pass 500 Hz:

▶ 0:00 / 0:09 🔊 ⋮

🎧 High-pass 1000 Hz:

▶ 0:00 / 0:09 🔊 ⋮

🎧 Band-pass 500-2000 Hz:

▶ 0:00 / 0:09 🔊 ⋮

🎧 Band-pass 400-2500 Hz:

▶ 0:00 / 0:09 🔊 ⋮

🎧 Band-pass 600-2200 Hz:

▶ 0:00 / 0:09 🔊 ⋮

```

In [8]: # =====
# Visualisasi hasil
# =====

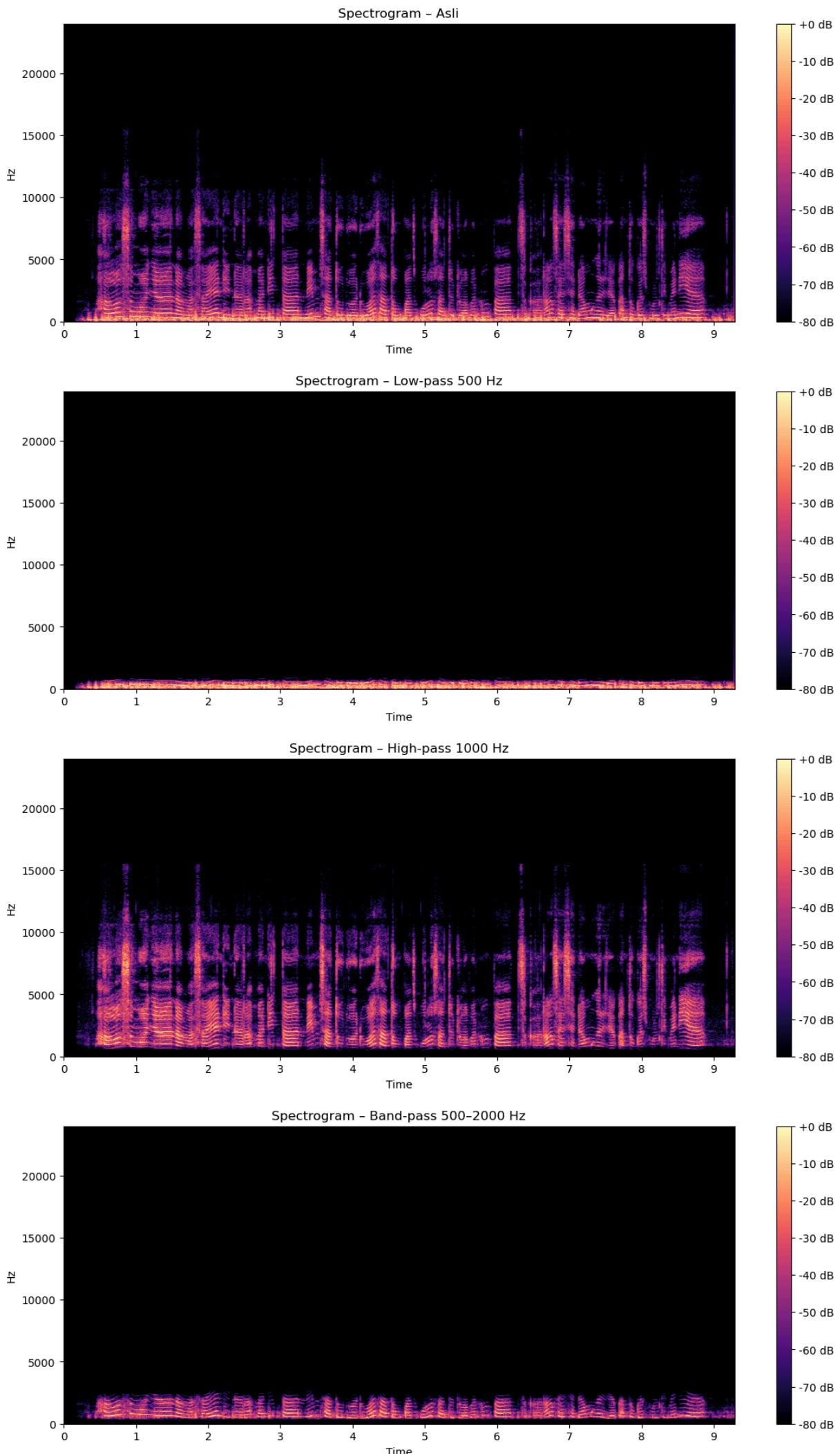
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

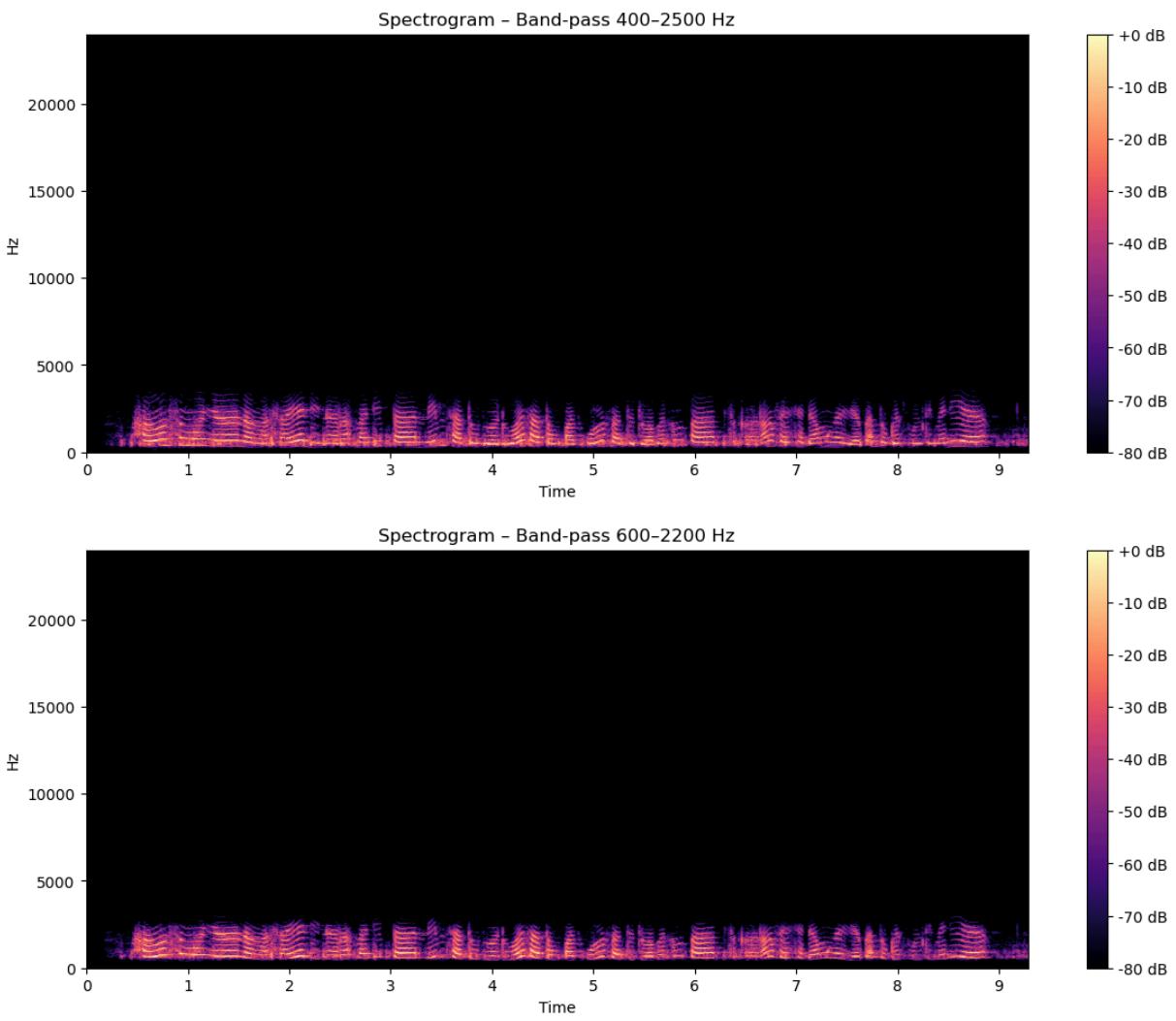
def plot_spec(y, sr, title):
    S = librosa.stft(y)
    S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
    plt.figure(figsize=(14, 5))
    librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    plt.colorbar(format='%.2f dB')
    plt.title(title)
    plt.show()

# --- Tampilkan semua versi ---

```

```
for name, data in versions.items():
    plot_spec(data, sr, f"Spectrogram - {name}")
```





Dari hasil perbandingan yang dilakukan, menurut saya filter high-pass paling efektif dalam mengurangi noise pada audio. Hal ini karena suara kipas yang menjadi sumber noise utama berada pada frekuensi rendah (<1000 Hz). Saat diterapkan high-pass filter dengan cutoff 1000 Hz, bagian frekuensi rendah tersebut berhasil ditekan dengan cukup baik.

Namun, penggunaan high-pass filter juga punya efek samping. Suara ucapan terdengar sedikit "cempreng" karena sebagian dari frekuensi rendah yang membuat suara terasa tebal ikut terpotong.

Sementara itu, low-pass filter justru membuat noise makin terasa karena membiarkan frekuensi rendah lewat, sehingga suara terdengar kurang jelas seperti "orang kumur-kumur." Sedangkan band-pass filter dengan rentang frekuensi 500–2000 Hz atau 600–2200 Hz memang bisa menjaga keseimbangan antara kejernihan dan kealamian suara, tapi masih menyisakan sedikit noise karena sebagian frekuensi gangguan tetap ikut lewat.

Jadi, bisa disimpulkan bahwa cutoff terbaik ada pada high-pass 1000 Hz, karena hasilnya menghasilkan suara yang tetap jelas dan noise paling sedikit dibandingkan filter lainnya, meskipun warna suaranya sedikit berubah.

```
In [9]: # =====#
# Soal 3 - Pitch Shifting dan Audio Manipulation
# =====#

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

# Load audio asli (hasil Soal 1)
y, sr = librosa.load("audio.wav", sr=None)
print(f"Sample rate: {sr} Hz | Durasi: {len(y)/sr:.2f} detik")

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 hasil
sf.write("audio_pitch_+7.wav", y_pitch7, sr)
sf.write("audio_pitch_+12.wav", y_pitch12, sr)

# 3. Gabungkan kedua hasil pitch shift menjadi satu audio
combined_audio = np.concatenate((y_pitch7, y_pitch12))
sf.write("audio_pitch_combined.wav", combined_audio, sr)

# 4. Dengarkan hasil
print("🎧 Asli:")
display(Audio(y, rate=sr))

print("🎵 Pitch +7 semitone:")
display(Audio(y_pitch7, rate=sr))

print("🎵 Pitch +12 semitone (chipmunk):")
display(Audio(y_pitch12, rate=sr))

print("🎶 Gabungan (+7 diikuti +12):")
display(Audio(combined_audio, rate=sr))

```

Sample rate: 48000 Hz | Durasi: 25.30 detik

🎧 Asli:

▶ 0:00 / 0:25 ━━━━ 🔊 ⋮

🎵 Pitch +7 semitone:

▶ 0:00 / 0:25 ━━━━ 🔊 ⋮

🎵 Pitch +12 semitone (chipmunk):

▶ 0:00 / 0:25 ━━━━ 🔊 ⋮

🎶 Gabungan (+7 diikuti +12):

▶ 0:00 / 0:50 ━━━━ 🔊 ⋮

In [10]: # =====

```

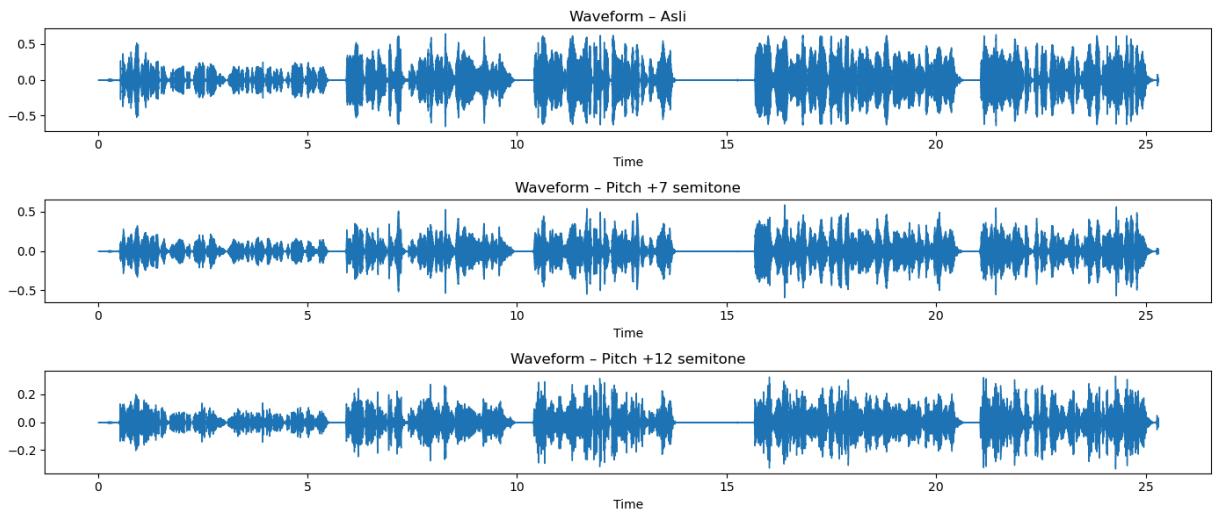
# Visualisasi dengan Waveform
# =====

plt.figure(figsize=(14, 6))
plt.subplot(3,1,1)
librosa.display.waveform(y, sr=sr)
plt.title("Waveform - Asli")

plt.subplot(3,1,2)
librosa.display.waveform(y_pitch7, sr=sr)
plt.title("Waveform - Pitch +7 semitone")

plt.subplot(3,1,3)
librosa.display.waveform(y_pitch12, sr=sr)
plt.title("Waveform - Pitch +12 semitone")
plt.tight_layout()
plt.show()

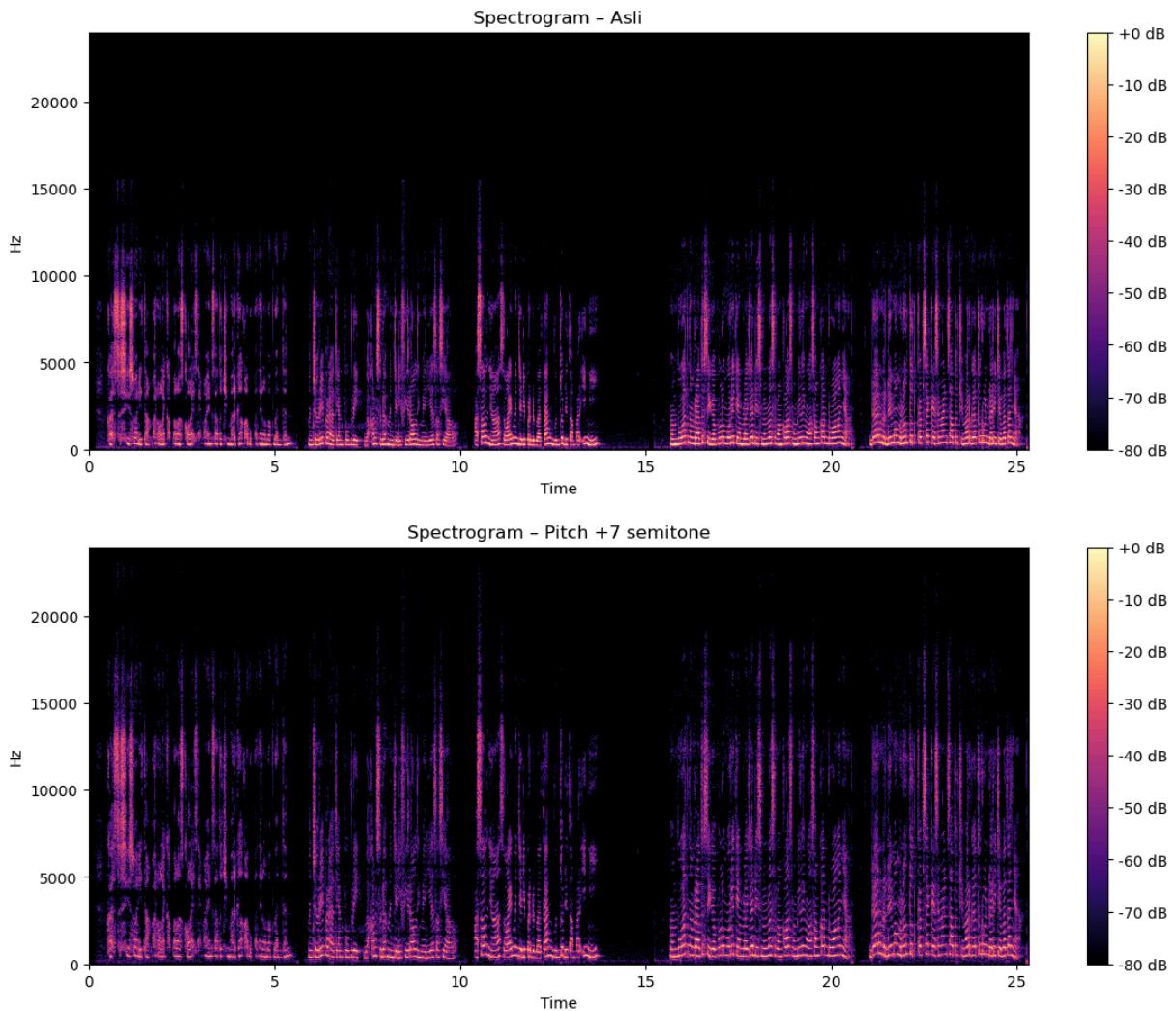
```

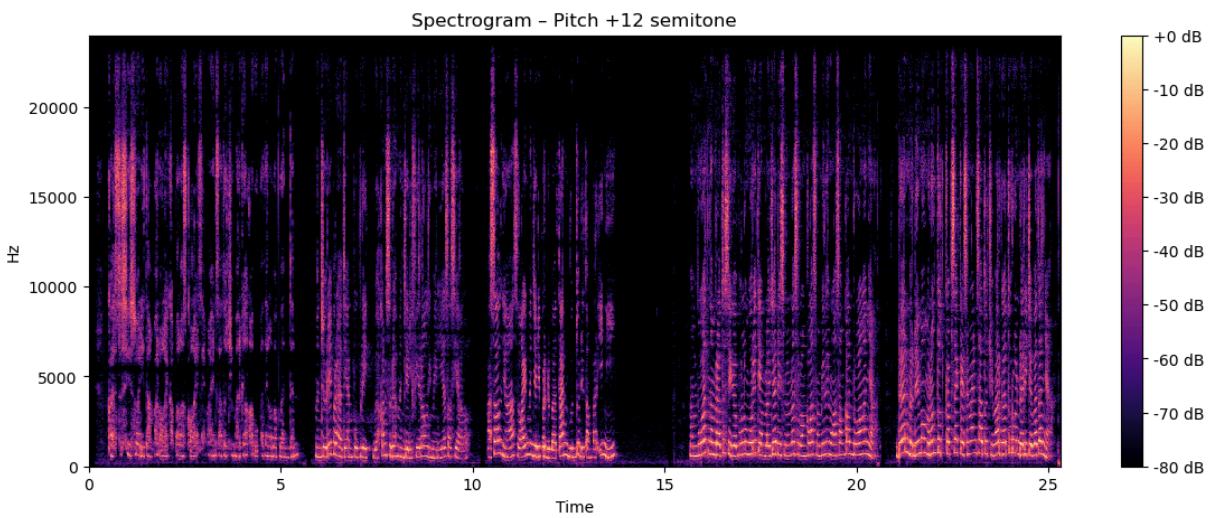


```
In [11]: # -----
# Visualisasi dengan Spectrogram
# -----
```

```
def plot_spec(y, sr, title):
    S = librosa.stft(y)
    S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
    plt.figure(figsize=(14, 5))
    librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    plt.colorbar(format='%.+2.0f dB')
    plt.title(title)
    plt.show()

plot_spec(y, sr, "Spectrogram - Asli")
plot_spec(y_pitch7, sr, "Spectrogram - Pitch +7 semitone")
plot_spec(y_pitch12, sr, "Spectrogram - Pitch +12 semitone")
```





Proses pitch shifting dilakukan dengan menggunakan fungsi `librosa.effects.pitch_shift()` dan mengatur parameter `n_steps` sebesar `+7` dan `+12` semitone. Nilai `+7` berarti menaikkan nada sekitar setengah oktaf, sedangkan `+12` menaikkan satu oktaf penuh sehingga menghasilkan efek suara seperti "chipmunk".

Beberapa parameter penting yang digunakan antara lain:

- `n_steps`: jumlah kenaikan atau penurunan nada dalam satuan semitone.
- `sr`: sample rate dari audio asli.

Librosa secara otomatis menjaga durasi audio tetap sama, karena proses penyesuaian waktu (time-stretch) dilakukan secara otomatis di dalam fungsi tersebut.

Analisis waveform

Berdasarkan hasil visualisasi waveform, terlihat bahwa gelombang suara setelah proses pitch shifting punya amplitudo yang sedikit lebih rendah dibandingkan suara aslinya. Puncak gelombangnya tidak setinggi sebelumnya, sehingga bisa dikatakan ada sedikit penurunan energi pada sinyal suara.

Penurunan ini kemungkinan terjadi karena proses interpolasi dan penyesuaian waktu otomatis (time-stretch kompensatif) yang dilakukan oleh fungsi `librosa.effects.pitch_shift()`. Walaupun durasi audio tetap sama, perubahan frekuensi membuat sebagian energi sinyal berpindah, jadi amplitudonya terlihat sedikit menurun.

Selain itu, gelombang juga tampak lebih rapat, menandakan adanya peningkatan frekuensi — ciri khas ketika nada atau pitch dinaikkan.

Analisis Spectrogram

Dari hasil spectrogram, terlihat perbedaan yang cukup jelas antara suara asli dan hasil pitch shifting. Pada suara asli, energi paling dominan berada di rentang frekuensi sekitar 0–15.000 Hz. Setelah dilakukan peningkatan pitch sebesar `+7` semitone, jangkauan frekuensinya meluas hingga kurang lebih 20.000 Hz, yang menunjukkan bahwa seluruh nada dan harmoniknya naik sekitar setengah oktaf.

Ketika pitch dinaikkan `+12` semitone (satu oktaf penuh), hampir seluruh energi berpindah ke frekuensi di atas 20.000 Hz, sehingga bagian atas spectrogram tampak lebih "penuh".

Warna-warna cerah yang menunjukkan area berenergi tinggi juga bergeser ke frekuensi yang lebih tinggi, dengan kata lain, semakin besar kenaikan pitch, semakin tinggi pula posisi nada dan suara yang terlihat pada spectrogram.

```
In [12]: # =====#
# Soal 4 - Audio Processing Chain (Equalizer, Gain, Normalization, dll.)
# =====#
```

```

import librosa
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
import librosa.display
from IPython.display import Audio, display
from scipy.signal import butter, filtfilt

# -----
# 1. LOAD AUDIO HASIL PITCH SHIFTING
# -----
# File yang digunakan bisa ganti ke hasil Soal 3 (misal: audio_pitch_+7.wav)
y, sr = librosa.load("audio_pitch_+7.wav", sr=None)
print(f"Sample rate: {sr}, Durasi: {len(y)/sr:.2f} detik")

# -----
# 2. EQUALIZER (EQ)
# -----
# Tujuan EQ: menyeimbangkan spektrum suara dengan memotong atau menaikkan
# frekuensi tertentu. Di sini kita hilangkan frekuensi rendah (high-pass),
# batasi frekuensi tinggi (low-pass), dan boost mid frequency agar suara lebih jelas
def apply_eq(data, sr, low_cut=100, mid_boost=1.2, high_cut=8000):
    # high-pass filter: hapus frekuensi di bawah Low_cut (biasanya noise rendah)
    b, a = butter(6, low_cut / (0.5 * sr), btype='high')
    y_eq = filtfilt(b, a, data)

    # Low-pass filter: hapus frekuensi di atas high_cut (suara mendesis)
    b, a = butter(6, high_cut / (0.5 * sr), btype='low')
    y_eq = filtfilt(b, a, y_eq)

    # naikkan gain di area mid-frequency (simulasi mid-boost)
    y_eq = y_eq * mid_boost
    return y_eq

y_eq = apply_eq(y, sr)

# -----
# 3. GAIN / FADE
# -----
# Gain digunakan untuk menaikkan volume keseluruhan dalam satuan desibel (dB).
# disini gain yang diterapkan yaitu 2 dB
def apply_gain(data, gain_db=2):
    factor = 10 ** (gain_db / 20)
    return data * factor

y_gain = apply_gain(y_eq, gain_db=3)

# -----
# 4. NORMALIZATION
# -----
# Normalisasi mengatur level rata-rata (RMS) audio agar berada di target tertentu.
# Target -16 dBFS berarti loudness sedang (tidak terlalu keras, tidak terlalu pelan)
def normalize_to_peak(data, target_dBFS=-16):
    rms = np.sqrt(np.mean(np.square(data)))           # RMS (root mean square)
    current_dBFS = 20 * np.log10(rms)                 # ubah ke dBFS
    change_dB = target_dBFS - current_dBFS          # selisih yang perlu disesuaikan
    factor = 10 ** (change_dB / 20)                  # ubah ke faktor pengali
    return data * factor

y_norm = normalize_to_peak(y_gain, -16)

# -----
# 5. COMPRESSION
# -----
# Kompresor digunakan untuk menekan dinamika suara (mengurangi perbedaan antara
# bagian pelan dan keras). Threshold = level mulai dikompresi, ratio = seberapa besar
# penekanan dilakukan.
def simple_compressor(data, threshold=-20, ratio=4):
    data_DB = 20 * np.log10(np.abs(data) + 1e-6)      # ubah amplitudo ke dB
    over_threshold = data_DB > threshold              # cari bagian yang lebih keras
    gain_reduction = np.where(over_threshold, (data_DB - threshold) / ratio, 0)
    gain_factor = 10 ** (-gain_reduction / 20)         # ubah kembali ke faktor linier
    return data * gain_factor

```

```

y_comp = simple_compressor(y_norm)

# -----
# 6. NOISE GATE
#
# Noise gate menonaktifkan (membisukan) bagian yang sangat pelan agar noise latar
# (misal desis, kipas) hilang ketika tidak ada ucapan.
def noise_gate(data, threshold=0.02):
    gated = np.where(np.abs(data) < threshold, 0, data)
    return gated

y_gate = noise_gate(y_comp)

# -----
# 7. SILENCE TRIMMING
#
# Fungsi ini menghapus bagian hening di awal atau akhir rekaman agar hasil
# lebih ringkas dan efisien.
y_trimmed, _ = librosa.effects.trim(y_gate, top_db=30)

# Simpan hasil akhir
sf.write("audio_processed_chain.wav", y_trimmed, sr)

# -----
# 8. PERBANDINGAN HASIL AUDIO
#
print("🎧 Audio Sebelum Processing:")
display(Audio(y, rate=sr))

print("🎵 Audio Setelah Processing Chain:")
display(Audio(y_trimmed, rate=sr))

# -----
# 9. VISUALISASI PERBANDINGAN WAVEFORM
#
plt.figure(figsize=(14, 5))
plt.subplot(2,1,1)
librosa.display.waveshow(y, sr=sr)
plt.title("Waveform – Sebelum Processing")

plt.subplot(2,1,2)
librosa.display.waveshow(y_trimmed, sr=sr)
plt.title("Waveform – Setelah Processing Chain")
plt.tight_layout()
plt.show()

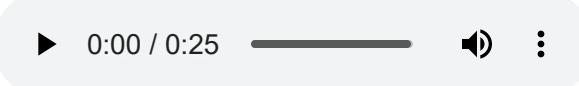
# -----
# 10. VISUALISASI PERBANDINGAN SPECTROGRAM
#
def plot_spec(y, sr, title):
    # Hitung Short-Time Fourier Transform (STFT)
    S = librosa.stft(y)
    # Konversi amplitudo ke skala dB
    S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
    # Plot spektrogram
    plt.figure(figsize=(14, 5))
    librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    plt.colorbar(format='%.2f dB')
    plt.title(title)
    plt.tight_layout()
    plt.show()

# Panggil fungsi di atas untuk audio sebelum dan sesudah processing
plot_spec(y, sr, "Spectrogram – Sebelum Processing Chain")
plot_spec(y_trimmed, sr, "Spectrogram – Setelah Processing Chain")

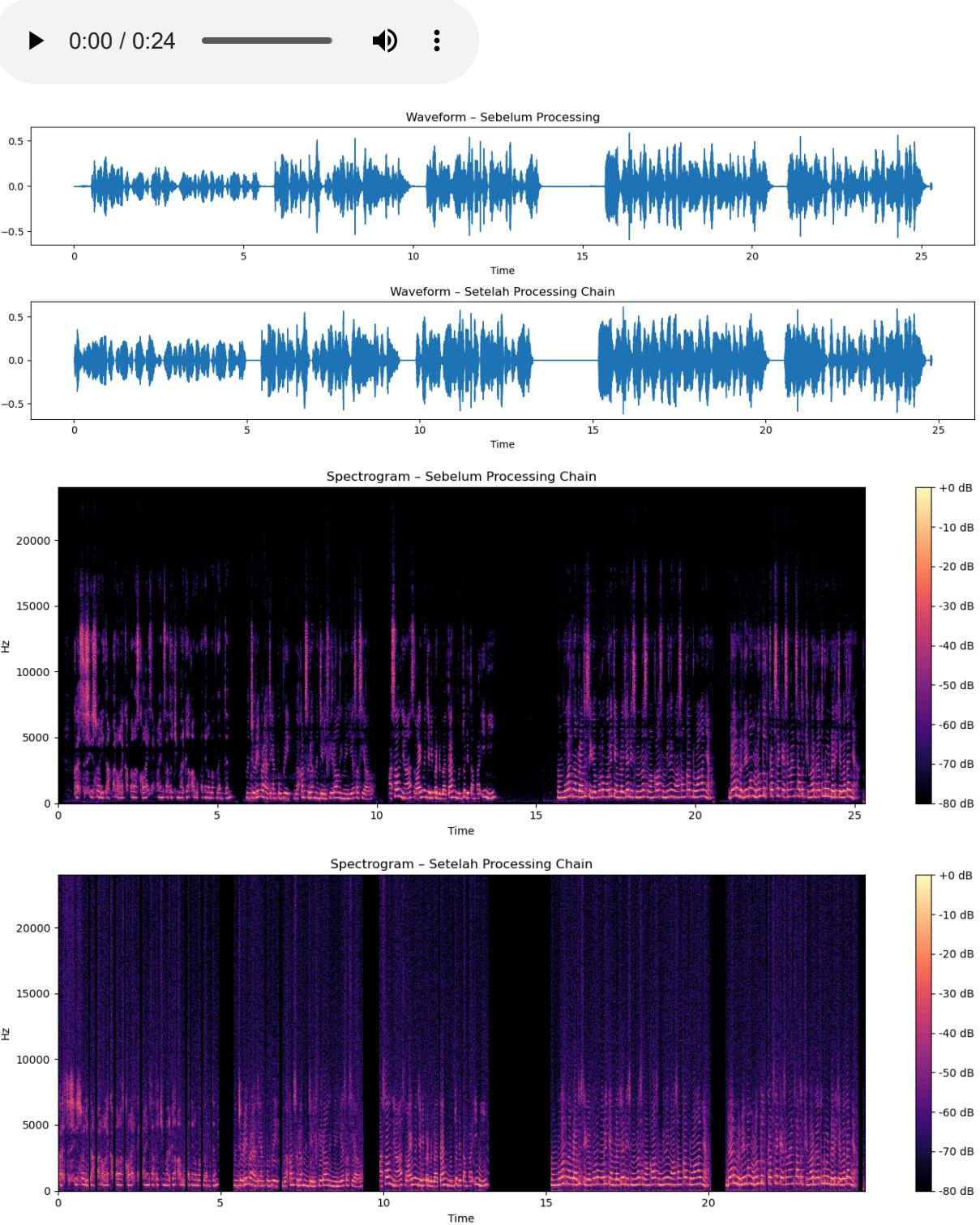
```

Sample rate: 48000 Durasi: 25-30 detik

Sample Rate: 48000, Duration: 2.000000



🎵 Audio Setelah Processing Chain:



1. Setelah melalui proses processing chain (equalizer, gain, normalisasi, kompresi, dan noise gate), terlihat bahwa gelombang suara menjadi lebih padat dan merata dibandingkan sebelum diproses. Bagian yang tadinya terlalu pelan kini terdengar lebih jelas, sedangkan bagian yang terlalu keras sedikit ditekan. Artinya, rentang dinamika suara berkurang — perbedaan antara bagian pelan dan keras tidak sejauh sebelumnya. Hasilnya, suara terdengar lebih stabil dan konsisten volumenya.
2. Normalisasi peak dan normalisasi LUFS sebenarnya punya tujuan yang mirip, yaitu menyeimbangkan volume suara, tapi caranya beda. Pada normalisasi peak, penyesuaian dilakukan berdasarkan puncak tertinggi dari sinyal audio. Jadi sistem hanya mengatur supaya bagian paling keras tidak melebihi batas tertentu. Tapi, cara ini tidak selalu membuat keseluruhan suara jadi terasa seimbang, karena hanya fokus ke titik puncaknya saja.

Sementara itu, normalisasi LUFS (Loudness Units Full Scale) lebih mempertimbangkan kenyaringan yang dirasakan oleh telinga manusia. Jadi bukan hanya melihat angka amplitudo, tapi menghitung seberapa “keras” suara itu terdengar secara rata-rata. Metode ini biasanya dipakai di produksi audio profesional karena hasilnya lebih konsisten kalau diputar di berbagai perangkat.

Secara singkat bisa dibilang, normalisasi peak lebih bersifat teknis, sedangkan normalisasi LUFS lebih menekankan pada kenyamanan pendengar.

3. Setelah melalui proses EQ, normalisasi, kompresi, dan noise gate, kualitas suara menunjukkan beberapa perubahan yang cukup jelas.

Pertama, noise pada frekuensi rendah berhasil dikurangi. Hal ini terlihat pada bagian bawah spectrogram (sekitar 0–500 Hz) yang tampak lebih gelap setelah proses dilakukan.

Perubahan ini disebabkan oleh penggunaan filter high-pass pada tahap EQ yang berfungsi menyaring suara rendah yang tidak diinginkan.

Kedua, tingkat loudness meningkat dan menjadi lebih stabil, yang bisa diamati dari warna spectrogram yang lebih cerah dan merata sepanjang waktu. Kejelasan ucapan juga meningkat, karena frekuensi menengah (sekitar 1–5 kHz) yang berperan penting dalam artikulasi diperkuat selama proses EQ.

Selain itu, bagian hening terdengar lebih bersih, ditandai dengan munculnya garis vertikal hitam pada spectrogram hasil akhir. Efek ini muncul akibat penerapan noise gate dan silence trimming yang memotong suara latar di bagian tanpa ucapan.

Namun, proses ini juga memiliki efek samping. Dinamika suara alami sedikit berkurang dan timbre terdengar lebih datar, karena perbedaan antara bagian keras dan pelan telah dikompresi.

4. Kelebihan:

- Volume terdengar lebih konsisten dari awal sampai akhir.
- Mengurangi risiko suara terlalu pelan saat diputar di perangkat berbeda.

Kekurangan:

- Jika dilakukan berlebihan, suara bisa terdengar kurang natural atau terasa "tertekan".
- Proses ini juga bisa menambah distorsi kecil jika gain terlalu tinggi.

In [13]:

```
# -----
# Soal 5 - Music Analysis dan Remix
# -----



import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
from IPython.display import Audio, display

# -----
# 1. LOAD LAGU
# -----
lagu1, sr1 = librosa.load("lagu_potongan_sedih.wav", sr=None)
lagu2, sr2 = librosa.load("lagu_potongan_senang.wav", sr=None)

print(f'Lagu sedih - Sample rate: {sr1}, Durasi: {len(lagu1)/sr1:.2f} detik')
print(f'Lagu senang - Sample rate: {sr2}, Durasi: {len(lagu2)/sr2:.2f} detik')

# -----
# 2. DETEKSI TEMPO (BPM) DAN KEY (KUNCI)
# -----
# Tempo (beats per minute)
tempo1, _ = librosa.beat.beat_track(y=lagu1, sr=sr1)
tempo2, _ = librosa.beat.beat_track(y=lagu2, sr=sr2)

# Estimasi kunci dasar dengan chroma feature
chroma1 = librosa.feature.chroma_cqt(y=lagu1, sr=sr1)
chroma2 = librosa.feature.chroma_cqt(y=lagu2, sr=sr2)
key1 = np.argmax(np.mean(chroma1, axis=1))
key2 = np.argmax(np.mean(chroma2, axis=1))
```

```

notasi = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
print(f"Lagu sedih: Tempo = {tempo1[0]:.2f} BPM, Key = {notasi[key1]}")
print(f"Lagu senang: Tempo = {tempo2[0]:.2f} BPM, Key = {notasi[key2]}")

# -----
# 3. TIME-STRETCH: Samakan Tempo
# -----
# Misal disamakan ke rata-rata tempo kedua Lagu
target_tempo = (tempo1[0] + tempo2[0]) / 2
rate1 = target_tempo / tempo1[0]
rate2 = target_tempo / tempo2[0]

lagu1_stretch = librosa.effects.time_stretch(lagu1, rate=rate1)
lagu2_stretch = librosa.effects.time_stretch(lagu2, rate=rate2)

# -----
# 4. PITCH SHIFT: Samakan Key
# -----
# Jika key Lagu2 lebih tinggi dari lagu1, turunkan sesuai selisih semitone
shift = key1 - key2
lagu2_pitch = librosa.effects.pitch_shift(lagu2_stretch, sr=sr2, n_steps=shift)

# -----
# 5. CROSSFADE DAN REMIX
# -----
# Potong durasi agar seimbang (misal 60 detik per lagu)
min_len = min(len(lagu1_stretch), len(lagu2_pitch))
lagu1_stretch = lagu1_stretch[:min_len]
lagu2_pitch = lagu2_pitch[:min_len]

# Tentukan panjang crossfade (misal 5 detik)
fade_duration = int(sr1 * 5)

# Buat envelope crossfade
fade_out = np.linspace(1, 0, fade_duration)
fade_in = np.linspace(0, 1, fade_duration)

# Terapkan crossfade
mix = np.concatenate([
    lagu1_stretch[:-fade_duration] * 1.0,
    lagu1_stretch[-fade_duration:] * fade_out + lagu2_pitch[:fade_duration] * fade_in,
    lagu2_pitch[fade_duration:]
])

# Simpan hasil remix
sf.write("lagu_remix.wav", mix, sr1)

# -----
# 6. FILTER TAMBAHAN (opsional, misal efek low-pass halus)
from scipy.signal import butter, filtfilt
b, a = butter(4, 8000 / (0.5 * sr1), btype='low')
mix_filtered = filtfilt(b, a, mix)
sf.write("lagu_remix_filtered.wav", mix_filtered, sr1)

# -----
# 7. VISUALISASI WAVEFORM & SPECTROGRAM
# -----
plt.figure(figsize=(14, 5))
librosa.display.waveshow(mix_filtered, sr=sr1)
plt.title("Waveform - Hasil Remix Lagu 1 & Lagu 2")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()

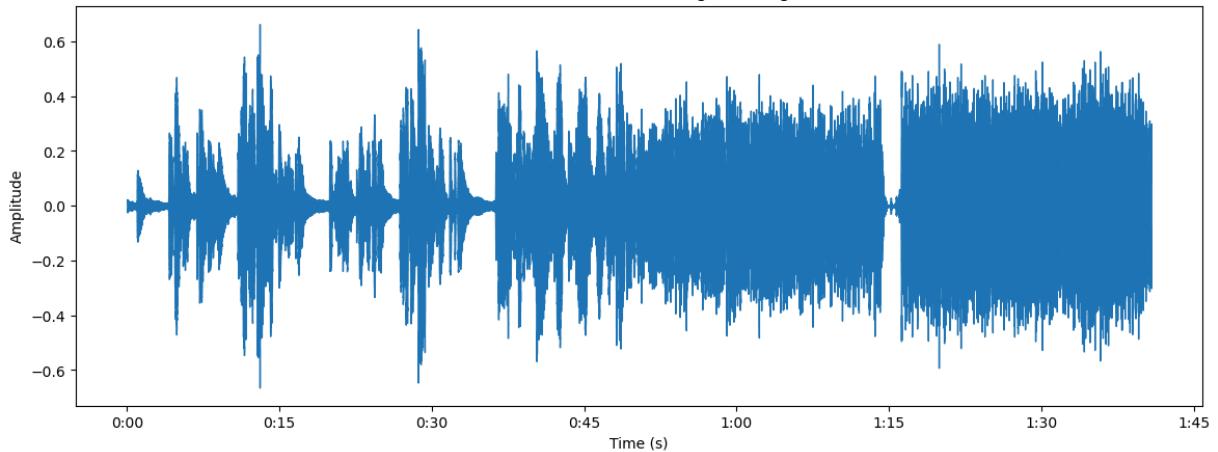
# Spectrogram
S = librosa.stft(mix_filtered)
S_db = librosa.amplitude_to_db(np.abs(S), ref=np.max)
plt.figure(figsize=(14, 6))
librosa.display.specshow(S_db, sr=sr1, x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%.2f dB')
plt.title("Spectrogram - Hasil Remix Lagu 1 & Lagu 2")
plt.show()

```

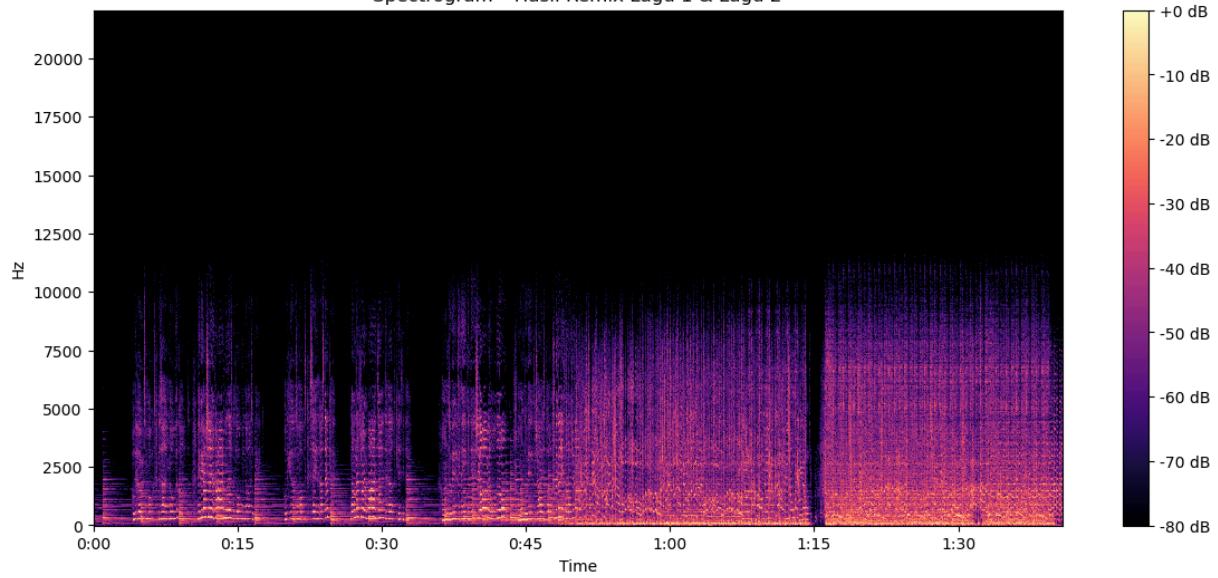
```
# Dengarkan hasilnya
print("🎧 Hasil Remix:")
display(Audio(mix_filtered, rate=sr1))
```

Lagu sedih - Sample rate: 44100, Durasi: 60.00 detik
 Lagu senang - Sample rate: 44100, Durasi: 60.00 detik
 Lagu sedih: Tempo = 126.05 BPM, Key = D
 Lagu senang: Tempo = 99.38 BPM, Key = D

Waveform – Hasil Remix Lagu 1 & Lagu 2



Spectrogram – Hasil Remix Lagu 1 & Lagu 2



🎧 Hasil Remix:



Langkah 1 – Pemilihan Lagu

Pada tahap awal, dipilih dua lagu dengan karakter yang berbeda untuk menghasilkan kontras yang jelas.

- Lagu 1 memiliki nuansa sedih dengan tempo lambat
- Lagu 2 bernuansa ceria dan memiliki tempo cepat.

Kedua lagu kemudian dikonversi ke format .wav agar dapat diolah menggunakan librosa di dalam notebook Python.

Langkah 2 – Analisis Tempo dan Kunci (Key Detection)

Tahap berikutnya adalah menganalisis tempo dan kunci nada dari masing-masing lagu.

- Proses ini menggunakan fungsi librosa.beat.beat_track() untuk mendeteksi tempo (BPM), dan librosa.feature.chroma_cqt() untuk mengestimasi key (nada dasar) lagu.
- Tempo menunjukkan seberapa cepat lagu berjalan, sedangkan key menggambarkan pusat nada (misalnya C, D, G, atau A#).

Hal ini dilakukan karena dua lagu dengan tempo dan key yang berbeda akan terdengar tidak selaras (clashing) jika langsung digabung tanpa penyesuaian.

Langkah 3 – Time Stretching (Penyamaan Tempo)

Setelah mengetahui tempo masing-masing lagu, dilakukan proses penyamaan tempo agar kedua lagu memiliki kecepatan yang serupa.

- Proses ini menggunakan fungsi librosa.effects.time_stretch().
- Rumus yang digunakan:
 - $rate = target_tempo / tempo_asli$
 - $lagu_stretch = librosa.effects.time_stretch(lagu, rate)$

Dengan cara ini, tempo dapat diubah tanpa memengaruhi tinggi nada. Hasilnya, kedua lagu kini berdurasi dan berkecepatan relatif sama, sehingga transisinya terdengar lebih halus.

Langkah 4 – Pitch Shifting (Penyamaan Nada Dasar)

Tahap ini dilakukan untuk menyamakan nada dasar (key) dari kedua lagu agar terdengar harmonis saat digabungkan.

- Penyesuaian dilakukan menggunakan fungsi librosa.effects.pitch_shift():
- $lagu_pitch = librosa.effects.pitch_shift(lagu, sr, n_steps=selisih_semitone)$
- Parameter n_steps menunjukkan jumlah semitone yang digeser (positif untuk menaikkan nada, negatif untuk menurunkan).

Tujuannya agar harmoni antar lagu terdengar natural ketika transisi terjadi.

Langkah 5 – Crossfading (Transisi Halus)

Untuk menggabungkan dua lagu secara halus tanpa jeda, digunakan teknik crossfade, yaitu kombinasi fade-out pada lagu pertama dan fade-in pada lagu kedua selama ± 5 detik.

- Proses ini menggunakan pendekatan berikut:
- $fade_out = np.linspace(1, 0, durasi_fade)$
- $fade_in = np.linspace(0, 1, durasi_fade)$

Dengan cara ini, pergantian antara dua lagu terasa mulus tanpa hentakan mendadak.

Langkah 6 – Filter Tambahan

Diterapkan low-pass filter dengan cutoff sekitar 8000 Hz menggunakan fungsi Butterworth Filter:

```
b, a = butter(4, 8000/(0.5*sr), btype='low')
mix_filtered = filtfilt(b, a, mix)
```

Filter ini berfungsi melembutkan suara dengan mengurangi frekuensi tinggi yang terlalu tajam, sehingga hasil perpindahan antar lagu terasa lebih lembut dan nyaman didengar.

Berdasarkan waveform dapat disimpulkan:

Pada bagian awal, amplitudo lagu tampak masih rendah dengan ritme yang terasa longgar, menggambarkan suasana lagu yang lambat dan melankolis. Ketika memasuki bagian pertengahan hingga akhir, amplitudo mulai meningkat dan gelombang terlihat lebih rapat, menandakan perubahan tempo menjadi lebih cepat serta nuansa lagu yang lebih ceria. Di bagian tengah grafik tampak adanya area transisi lembut (crossfade), terlihat dari penurunan amplitudo secara perlahan sebelum kembali meningkat.

Sementara itu, spectrogram memperlihatkan perubahan distribusi energi frekuensi dari awal hingga akhir lagu: Bagian awal yang bernuansa sedih didominasi warna merah-ungu di

bawah 7500 Hz, menunjukkan kehadiran instrumen lembut dan vokal pelan. Setelah melewati bagian crossfade, energi frekuensi meluas hingga kisaran 10 kHz dengan warna yang lebih cerah, menandakan masuknya lagu kedua yang lebih dinamis dan kaya frekuensi. Transisi antarbagian terlihat halus tanpa adanya lonjakan energi mendadak, yang menunjukkan bahwa proses time stretch dan pitch shift berhasil menjaga kesinambungan antara kedua lagu tersebut.