

Hans On Pemrosesan Audio

Nama : Rafki Haykhal Alif

NIM : 122140035

Soal 1: Rekaman dan Analisis Suara Multi-Level

1. Load rekaman suara dengan jenis suara yang berbeda beda dan menampilkan waveform dan spektrogramnya.

In [78]:

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

# Ganti dengan nama file rekaman kamu
file_path = "/data-rekaman/rekaman_soal_1.wav"

# Load audio tanpa mengubah sample rate aslinya
y, sr = librosa.load(file_path, sr=None)

print(f"Sample rate asli: {sr} Hz")
print(f"Durasi: {len(y)/sr:.2f} detik")

# Tampilkan waveform
plt.figure(figsize=(12, 3))
librosa.display.waveform(y, sr=sr)
plt.title("Waveform rekaman suara dengan 5 jenis suara berbeda")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.show()
```

```

# Tampilkan spektrogram
D = np.abs(librosa.stft(y))
DB = librosa.amplitude_to_db(D, ref=np.max)

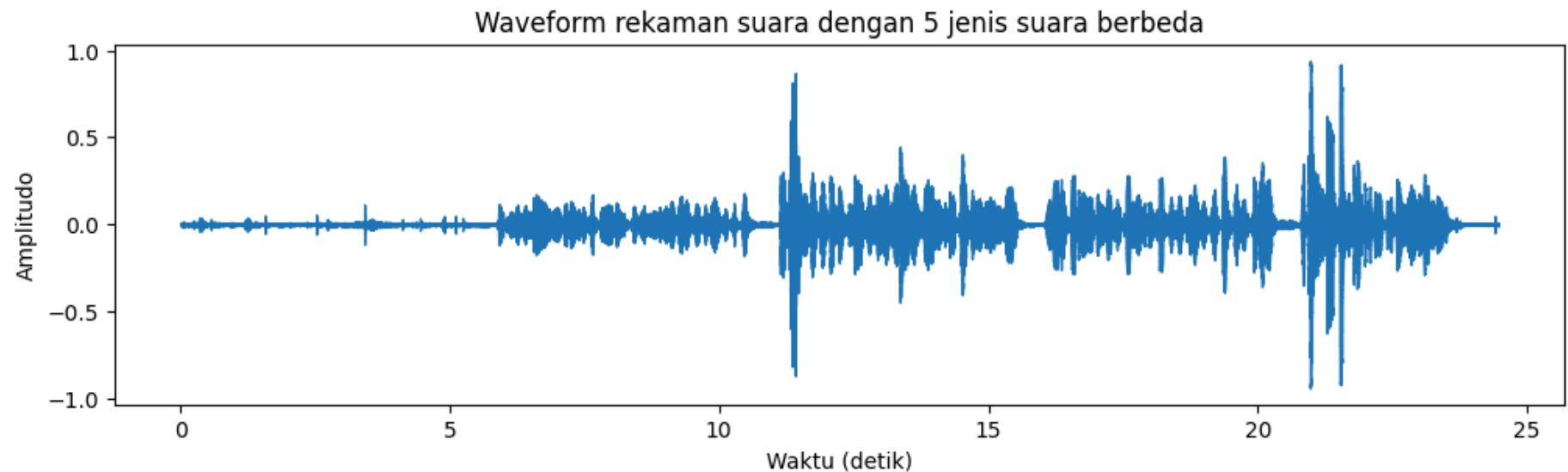
plt.figure(figsize=(12, 4))
librosa.display.specshow(DB, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format="%+2.0f dB")
plt.title("Spektrogram rekaman suara dengan 5 jenis suara berbeda")
plt.show()

# Putar audio
print("Suara Asli:")
display(Audio(y, rate=sr))

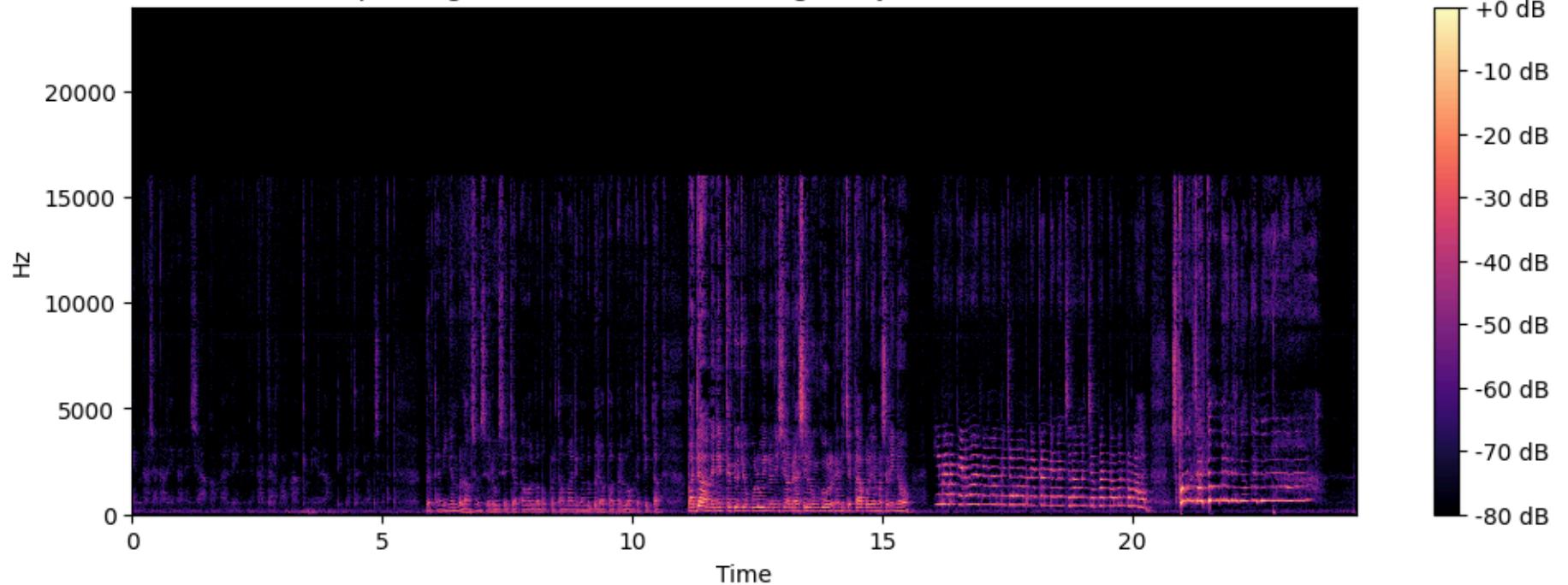
```

Sample rate asli: 48000 Hz

Durasi: 24.47 detik



Spektrogram rekaman suara dengan 5 jenis suara berbeda



Suara Asli:

▶ 0:00 / 0:24



:

2. Penjelasan singkat mengenai hasil visualisasi diatas.

Berdasarkan hasil visualisasi dari waveform dan histogram untuk rekaman suara dengan 5 jenis suara yang berbeda bisa dilihat bahwa setiap jenis suara tersebut memiliki karakteristik suara masing-masing dari waveform terlihat setiap 5 detik bentuk waveformnya berbeda beda satu sama lain yang menandakan setiap jenis suara tersebut memiliki amplitudo yang berbeda dan pada spektrogram juga kelihatan perbedaan frequensi dari tiap jenis suara tersebut.

3. Melakukan resampling pada rekaman suara

In [79]:

```
# Lakukan resampling ke 16 kHz
y_16k = librosa.resample(y, orig_sr=sr, target_sr=16000)

# Simpan hasil resampling
import soundfile as sf
sf.write("rekaman_rafki_16k.wav", y_16k, 16000)

print(f"Durasi asli: {len(y)/sr:.2f} detik")
print(f"Durasi setelah resampling: {len(y_16k)/16000:.2f} detik")

# Putar hasil resampling
print("Suara Setelah Resampling (16 kHz):")
display(Audio(y_16k, rate=16000))

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

# Spektrogram asli
plt.subplot(2, 1, 1)
D1 = np.abs(librosa.stft(y))
DB1 = librosa.amplitude_to_db(D1, ref=np.max)
librosa.display.specshow(DB1, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
plt.title(f"Spektrogram Asli ({sr} Hz)")
plt.colorbar(format="%+2.0f dB")

# Spektrogram hasil resampling
plt.subplot(2, 1, 2)
D2 = np.abs(librosa.stft(y_16k))
DB2 = librosa.amplitude_to_db(D2, ref=np.max)
librosa.display.specshow(DB2, sr=16000, x_axis='time', y_axis='hz', cmap='magma')
plt.title("Spektrogram Setelah Resampling (16 kHz)")
plt.colorbar(format="%+2.0f dB")

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

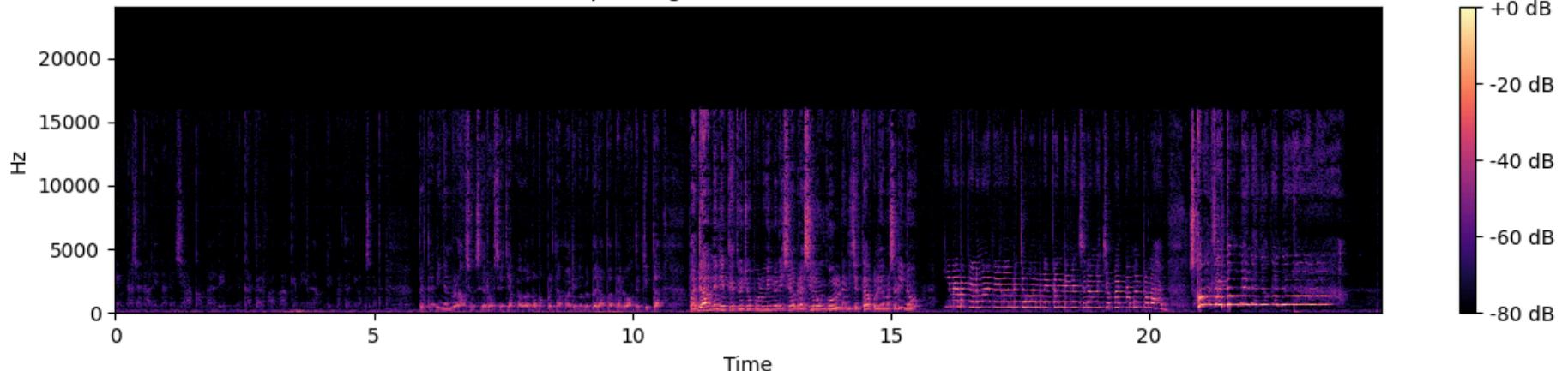
Durasi asli: 24.47 detik

Durasi setelah resampling: 24.47 detik

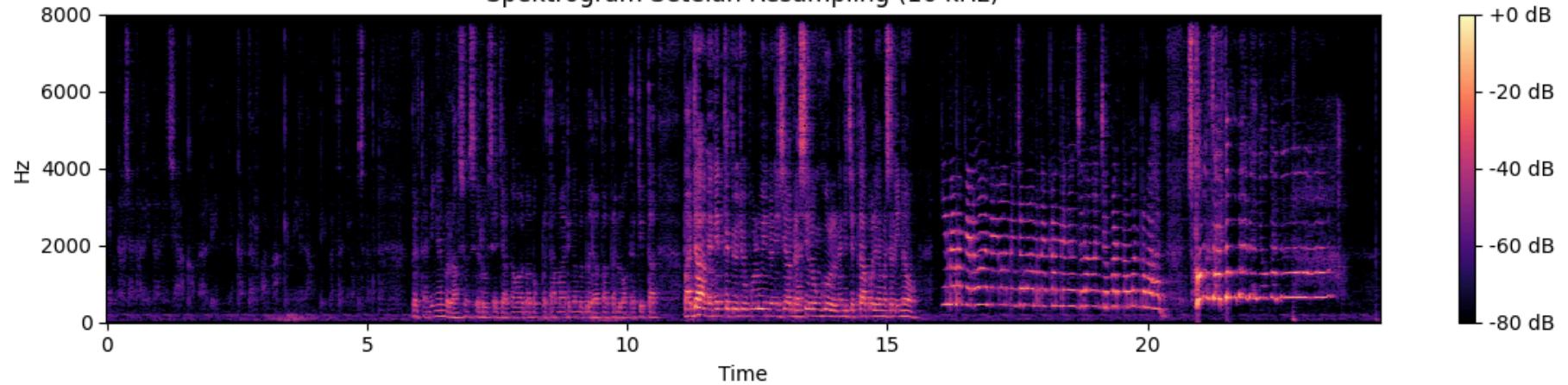
Suara Setelah Resampling (16 kHz):

▶ 0:00 / 0:24 ⏸ ⏴

Spektrogram Asli (48000 Hz)



Spektrogram Setelah Resampling (16 kHz)



Setelah melakukan resampling ke 16kHz didapatkan hasil spektrogram perbandingan dengan sebelum di resampling yang dapat dilihat diatas. Untuk kualitas audio yang dihasilkan setelah resampling adalah untuk suara yang memiliki frekuensi yang sangat tinggi seperti suara keras dan suara teriak akan hilang jika suara tersebut lebih tinggi dari 16 kHz tetapi itu hanya sedikit mempengaruhi dari suara aslinya. Durasi yang dihasilkan setelah di resampling tetap sama dengan durasi aslinya karena pada proses resampling ini hanya menghilangkan suara dengan frequensi yang tinggi

Soal 2: Noise Reduction dengan Filtering

1. Load rekaman suara dengan noise dan tampilan waveform dan spektrogram rekaman suara yang ada noise.

In [80]:

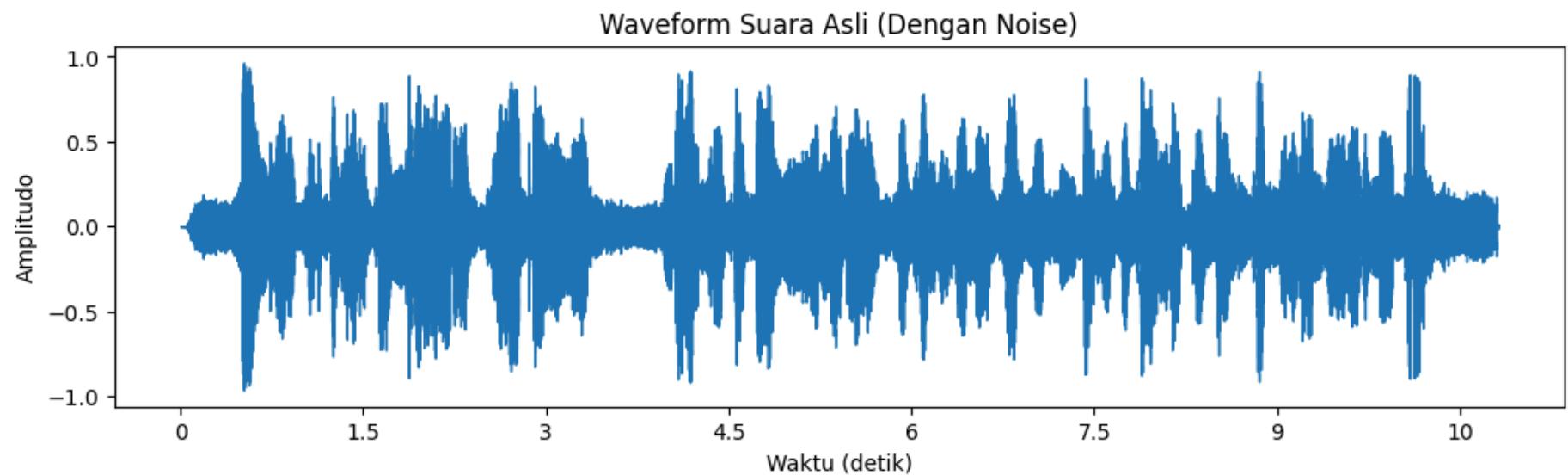
```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import Audio, display

# Load file rekaman
y, sr = librosa.load("/data-rekaman/rekaman_suara_noise.wav", sr=None)

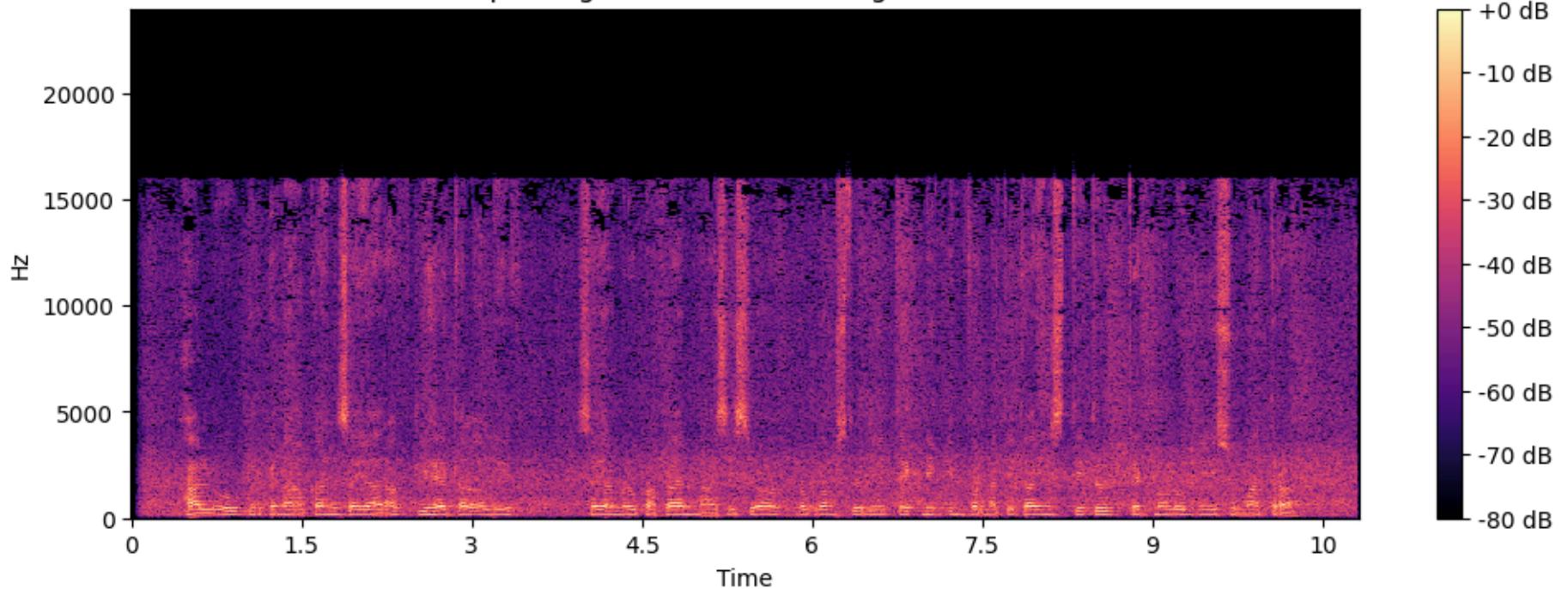
# Waveform
plt.figure(figsize=(12, 3))
librosa.display.waveform(y, sr=sr)
plt.title("Waveform Suara Asli (Dengan Noise)")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.show()

# Spektrogram
D = librosa.amplitude_to_db(abs(librosa.stft(y)), ref=np.max)
plt.figure(figsize=(12, 4))
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%.2f dB')
plt.title("Spektrogram Suara Asli (Dengan Noise)")
plt.show()
```

```
# Putar audio asli
print("Audio Asli (dengan noise):")
display(Audio(y, rate=sr))
```



Spektrogram Suara Asli (Dengan Noise)



Audio Asli (dengan noise):

▶ 0:00 / 0:10



:

2. Melakukan filtering pada audio yang memiliki noise dengan beberapa filter seperti lowpass, highpass, dan bandpass di frekuensi cutoff yang berbeda beda untuk mendapatkan hasil suara yang terbaik.

```
In [81]: def simple_filter(y, sr, ftype, cutoff_low=None, cutoff_high=None):
    """
    ftype: 'lowpass', 'highpass', 'bandpass'
    cutoff_low, cutoff_high: batas frekuensi (Hz)
    """
    Y = np.fft.rfft(y)
    freqs = np.fft.rfftfreq(len(y), 1/sr)
```

```

mask = np.ones_like(Y)
if ftype == 'lowpass':
    mask[freqs > cutoff_high] = 0
elif ftype == 'highpass':
    mask[freqs < cutoff_low] = 0
elif ftype == 'bandpass':
    mask[(freqs < cutoff_low) | (freqs > cutoff_high)] = 0

Y_filtered = Y * mask
y_filtered = np.fft.irfft(Y_filtered)
return y_filtered

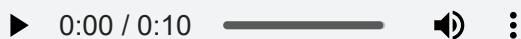
cutoffs = [500, 1000, 2000]
results = {}

for c in cutoffs:
    results[f'Low-pass {c}Hz'] = simple_filter(y, sr, 'lowpass', cutoff_high=c)
    results[f'High-pass {c}Hz'] = simple_filter(y, sr, 'highpass', cutoff_low=c)
    results[f'Band-pass 500-{c}Hz'] = simple_filter(y, sr, 'bandpass', cutoff_low=500, cutoff_high=c)

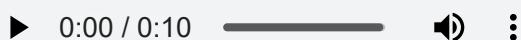
for name, data in results.items():
    print(f"\n{name}")
    display(Audio(data, rate=sr))

```

Low-pass 500Hz



High-pass 500Hz



Band-pass 500-500Hz

```

/usr/local/lib/python3.12/dist-packages/IPython/lib/display.py:174: RuntimeWarning: invalid value encountered in divide
  scaled = data / normalization_factor * 32767
/usr/local/lib/python3.12/dist-packages/IPython/lib/display.py:175: RuntimeWarning: invalid value encountered in cast
  return scaled.astype("<h").tobytes(), nchan

```

▶ 0:00 / 0:10 🔊 ⋮

Low-pass 1000Hz

▶ 0:00 / 0:10 🔊 ⋮

High-pass 1000Hz

▶ 0:00 / 0:10 🔊 ⋮

Band-pass 500-1000Hz

▶ 0:00 / 0:10 🔊 ⋮

Low-pass 2000Hz

▶ 0:00 / 0:10 🔊 ⋮

High-pass 2000Hz

▶ 0:00 / 0:10 🔊 ⋮

Band-pass 500-2000Hz

▶ 0:00 / 0:10 🔊 ⋮

```
In [82]: #simpan hasil high-pass 500Hz
import soundfile as sf
sf.write("rekaman_rafki_highpass_500Hz.wav", results["High-pass 500Hz"], sr)
```

3. Menampilkan visualisasi spektrogram dari hasil suara yang sudah dilakukan filter dan membandingkannya dengan spektrogram suara original.

```
In [83]: plt.figure(figsize=(12, 12))

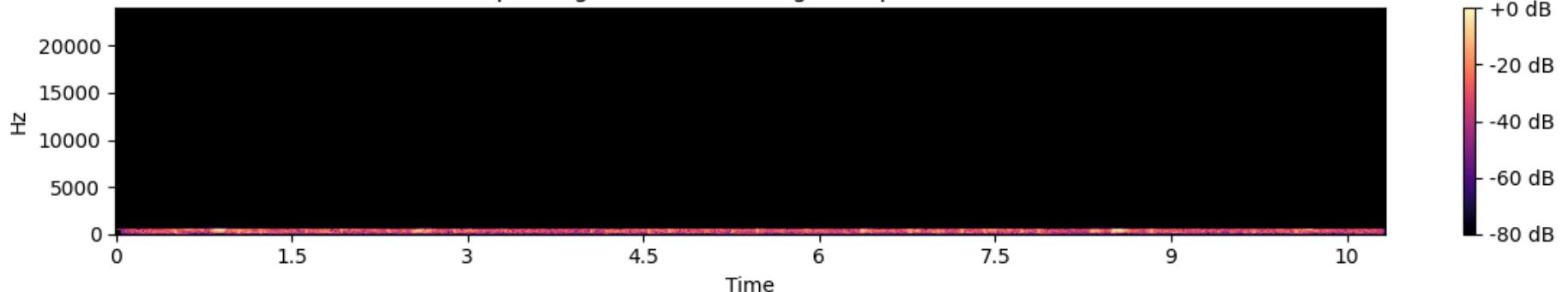
filters_to_show = [
    "Low-pass 500Hz",
    "High-pass 500Hz",
    "High-pass 1000Hz",
    "Band-pass 500-2000Hz"
]

for i, name in enumerate(filters_to_show, 1):
    plt.subplot(len(filters_to_show)+1, 1, i)
    D_filtered = librosa.amplitude_to_db(np.abs(librosa.stft(results[name])), ref=np.max)
    librosa.display.specshow(D_filtered, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    plt.title(f"Spektrogram Hasil Filtering: {name}")
    plt.colorbar(format="%+2.0f dB")

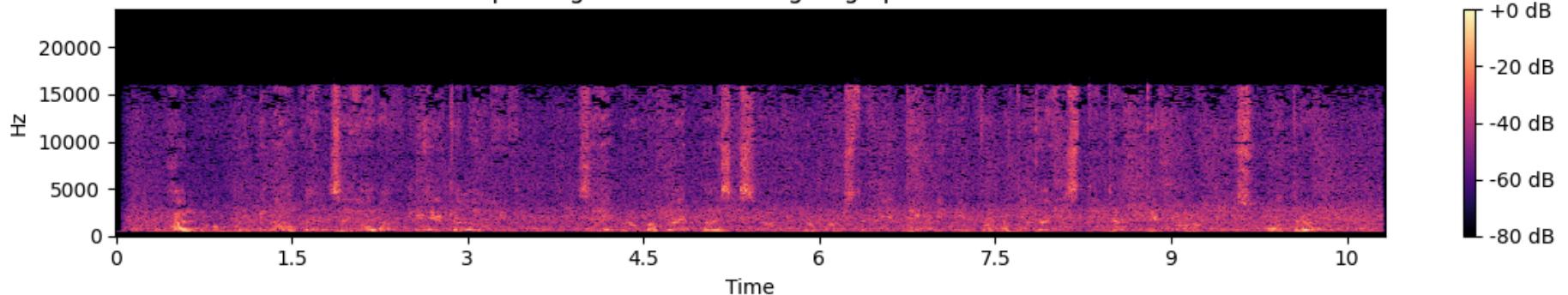
# Tambahkan spektrogram asli di bawahnya untuk perbandingan
plt.subplot(len(filters_to_show)+1, 1, len(filters_to_show)+1)
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
plt.title("Spektrogram Asli (Tanpa Filter)")
plt.colorbar(format="%+2.0f dB")

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

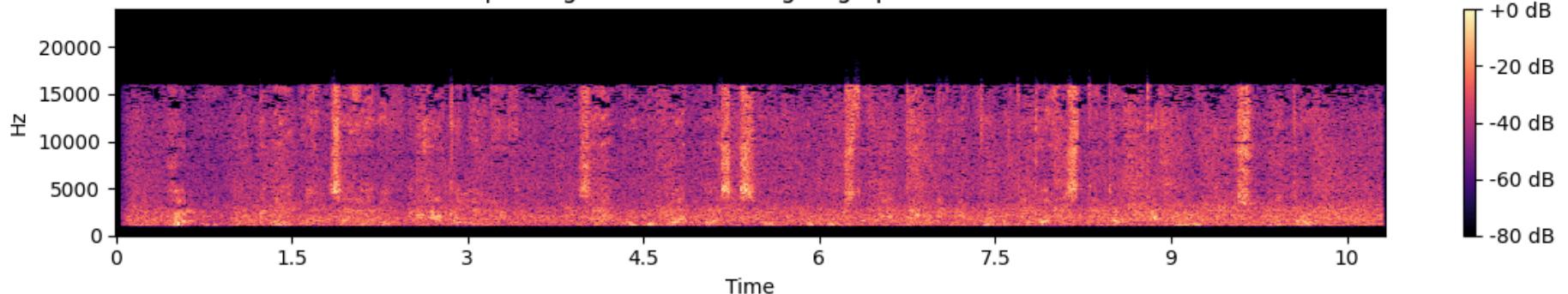
Spektrogram Hasil Filtering: Low-pass 500Hz



Spektrogram Hasil Filtering: High-pass 500Hz

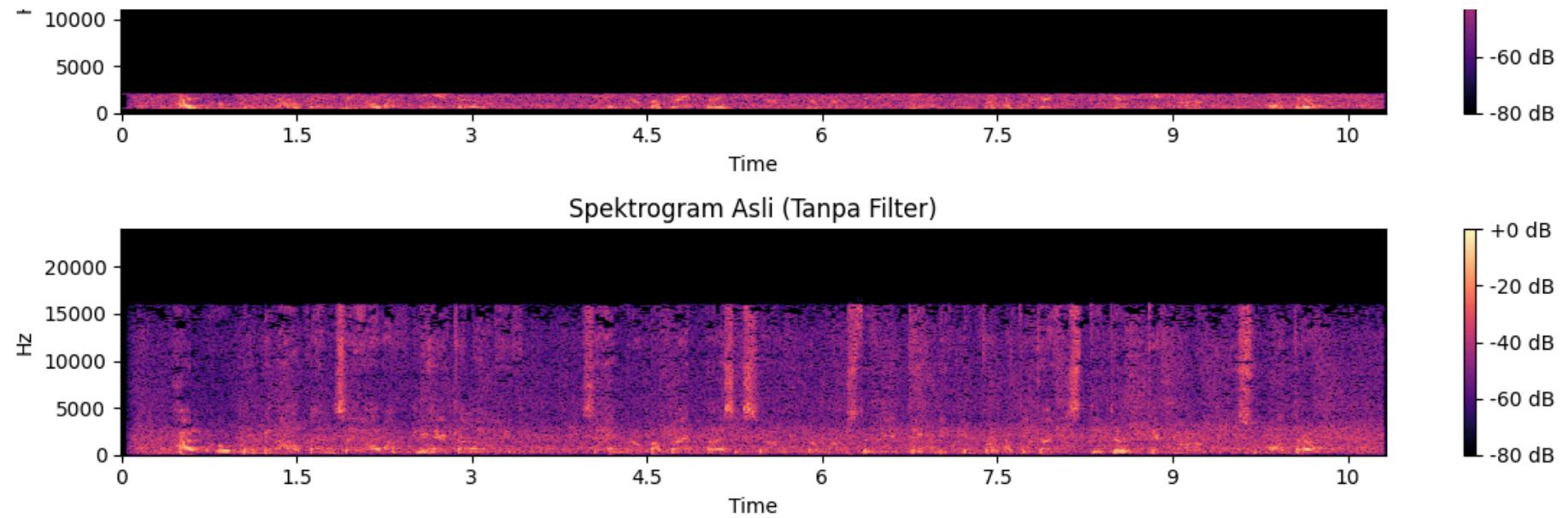


Spektrogram Hasil Filtering: High-pass 1000Hz



Spektrogram Hasil Filtering: Band-pass 500-2000Hz





4. Analisis :

A. Jenis noise yang muncul pada rekaman Anda

Noise yang terdapat pada rekaman suara saya tersebut merupakan noise yang berasal dari suara hujan.

B. Filter mana yang paling efektif untuk mengurangi noise tersebut

Berdasarkan hasil eksperimen saya dalam melakukan filtering pada rekaman suara yang terdapat noise tersebut, dari semua filter yang digunakan hasil suara yang paling optimal yang dihasilkan yaitu suara dengan menggunakan filter Highpass karena noise yang ada pada frequensi yang cukup tinggi.

C. Nilai cutoff yang memberikan hasil terbaik

Nilai cutoff yang memberikan hasil terbaik yaitu 500Hz karena pada proses filter highpass dengan cutoff 500Hz menghasilkan suara yang paling optimal dibandingkan dengan yang lain

D. Bagaimana kualitas suara (kejelasan ucapan) setelah proses filtering

Kualitas suara yang dihasilkan setelah proses filtering yaitu cukup jelas walaupun masih terdapat sedikit suara noise pada hasil suaranya karena noise yang ada cukup besar dan lumayan sulit untuk dihilangkan dan akan mempengaruhi suara rekamannya.

Soal 3: Pitch Shifting dan Audio Manipulation

1. Load rekaman suara yang akan dilakukan proses pitch shifting yaitu dengan menggunakan rekaman suara pada soal nomor 1.

```
In [84]: import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import Audio, display

# Muat audio asli
y, sr = librosa.load("/data-rekaman/rekaman_soal_1.wav", sr=None)

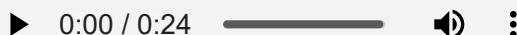
print(f"Sample rate: {sr} Hz")
print(f"Durasi: {len(y)/sr:.2f} detik")

# Putar audio asli
print("Suara Asli:")
display(Audio(y, rate=sr))
```

Sample rate: 48000 Hz

Durasi: 24.47 detik

Suara Asli:



2. Melakukan proses pitch shifting dengan pitch +7 dan +12 yang membuat suaranya seperti suara chipmunk.

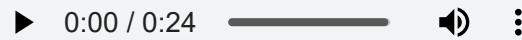
```
In [85]: # Pitch shifting
y_pitch7 = librosa.effects.pitch_shift(y, sr=sr, n_steps=7)
y_pitch12 = librosa.effects.pitch_shift(y, sr=sr, n_steps=12)

# Putar hasilnya
print("Pitch +7 Semitone:")
display(Audio(y_pitch7, rate=sr))

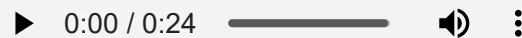
print("Pitch +12 Semitone:")
display(Audio(y_pitch12, rate=sr))

#Simpan hasil pitch shifting
sf.write("rekaman_rafki_pitch7.wav", y_pitch7, sr)
sf.write("rekaman_rafki_pitch12.wav", y_pitch12, sr)
```

Pitch +7 Semitone:



Pitch +12 Semitone:



3. Visualisasi perbandingan waveform dan spektrogram dari suara asli dan suara setelah dilakukan pitch shifting.

```
In [86]: # Waveform perbandingan
plt.figure(figsize=(12, 6))

plt.subplot(3, 1, 1)
librosa.display.waveshow(y, sr=sr)
plt.title("Waveform Asli")

plt.subplot(3, 1, 2)
librosa.display.waveshow(y_pitch7, sr=sr)
plt.title("Waveform Pitch +7 Semitone")
```

```
plt.subplot(3, 1, 3)
librosa.display.waveshow(y_pitch12, sr=sr)
plt.title("Waveform Pitch +12 Semitone")

plt.tight_layout()
plt.show()

# Spektrogram perbandingan
plt.figure(figsize=(12, 9))

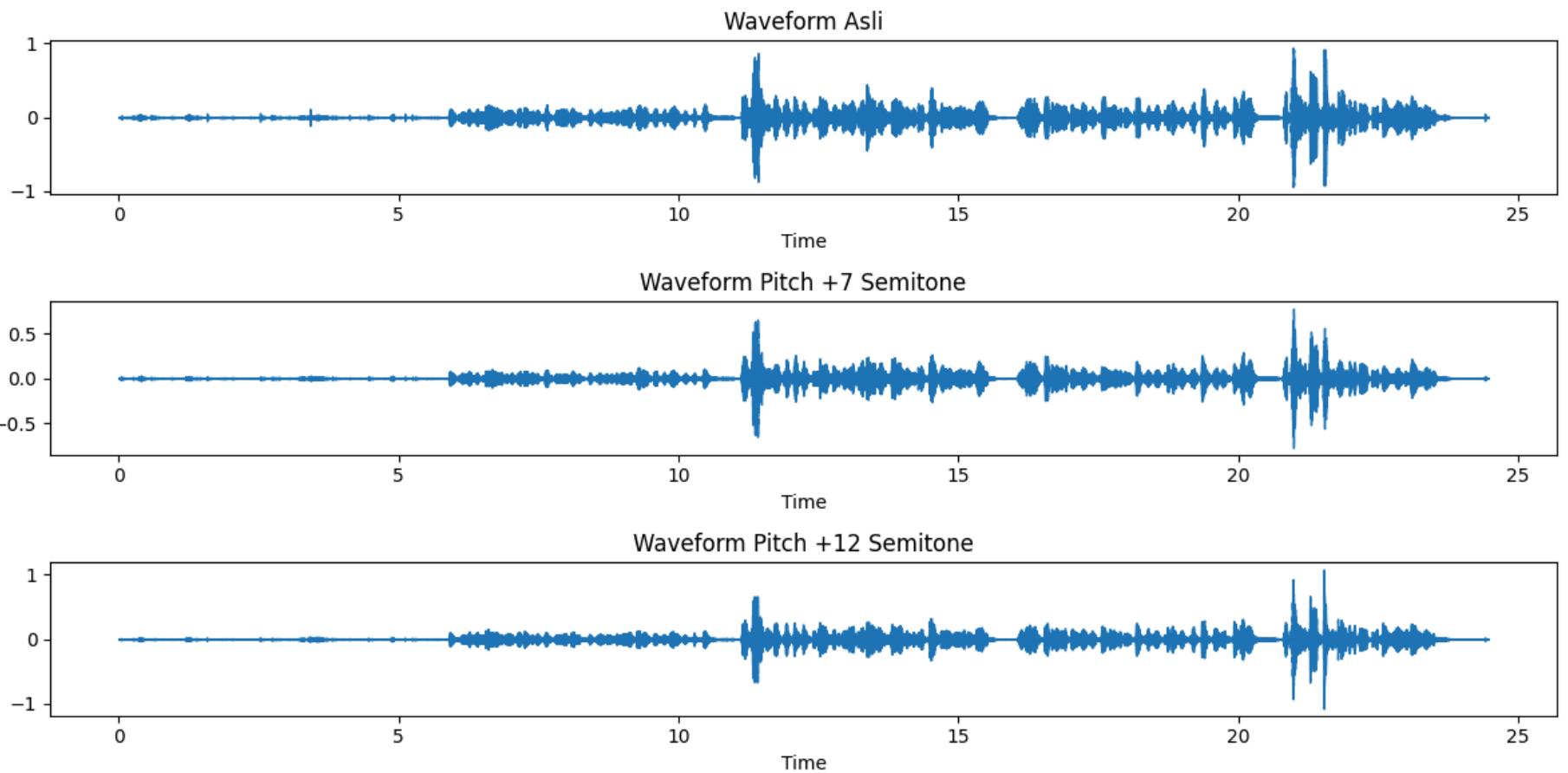
def plot_spec(y, sr, title):
    D = np.abs(librosa.stft(y))
    DB = librosa.amplitude_to_db(D, ref=np.max)
    librosa.display.specshow(DB, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    plt.title(title)
    plt.colorbar(format="%+2.0f dB")

plt.subplot(3, 1, 1)
plot_spec(y, sr, "Spektrogram Asli")

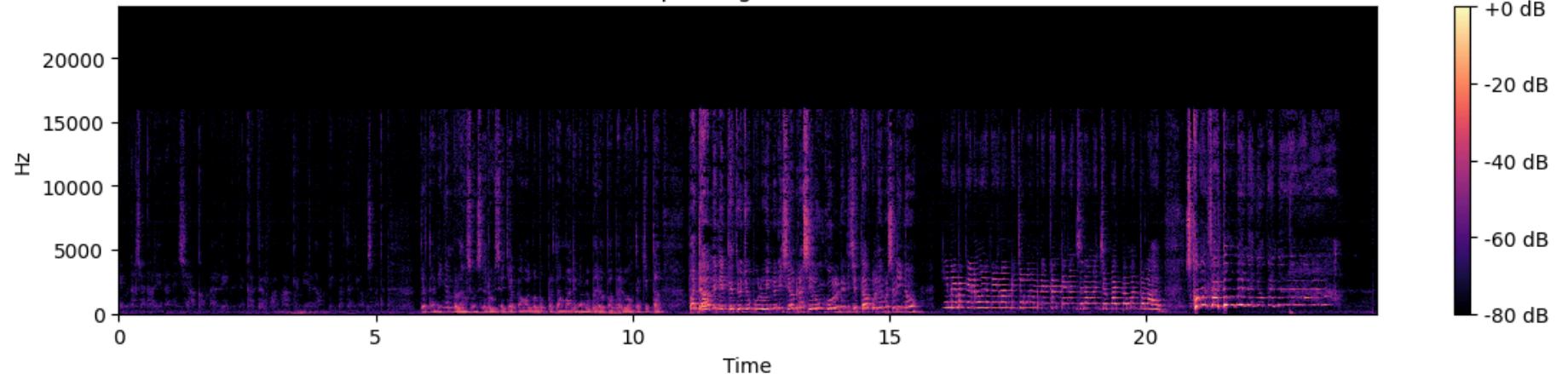
plt.subplot(3, 1, 2)
plot_spec(y_pitch7, sr, "Spektrogram Pitch +7 Semitone")

plt.subplot(3, 1, 3)
plot_spec(y_pitch12, sr, "Spektrogram Pitch +12 Semitone")

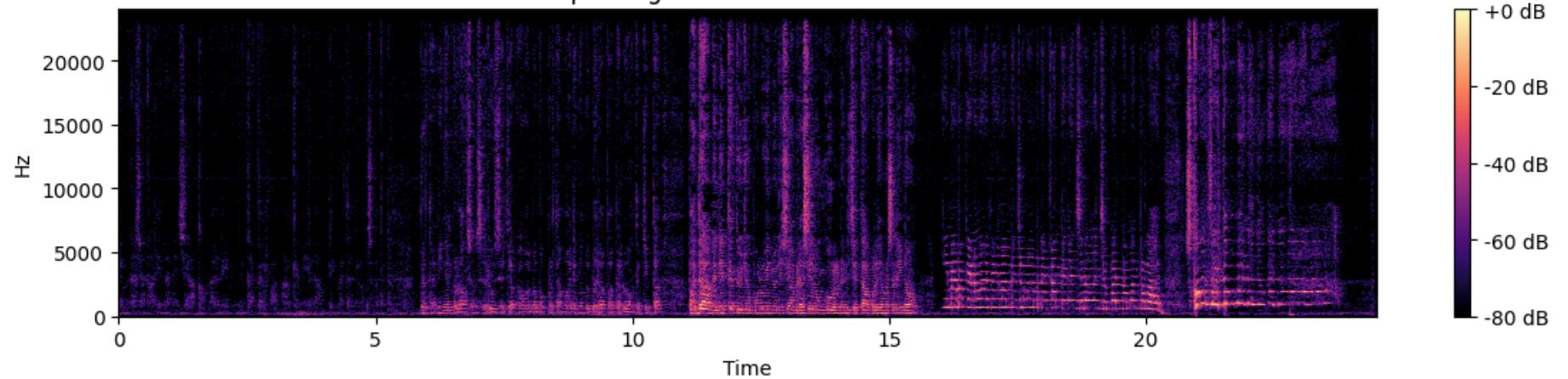
plt.tight_layout()
plt.show()
```



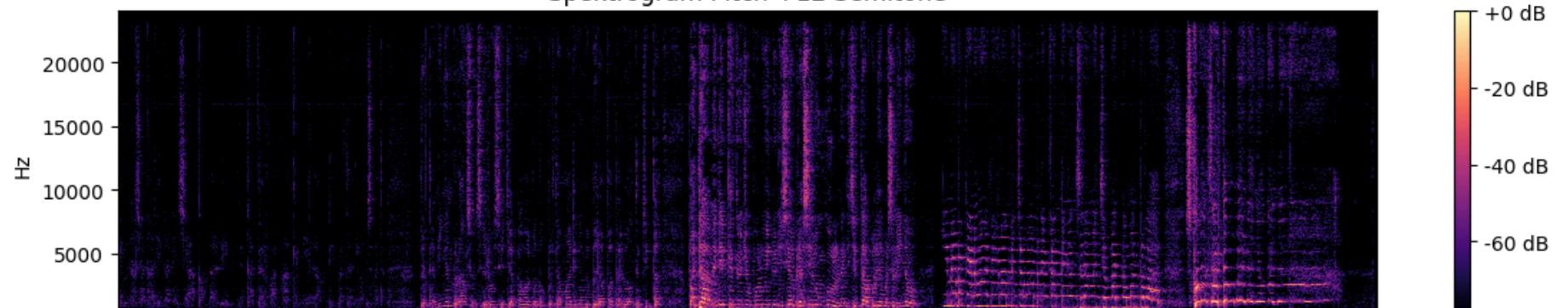
Spektrogram Asli



Spektrogram Pitch +7 Semitone



Spektrogram Pitch +12 Semitone





4. penjelasan dan proses pitch shifting yang dilakukan.

A. Parameter yang digunakan

Parameter yang digunakan pada proses pitch shifting ini yaitu parameter `y` sebagai sinyal audio asli, parameter `sr` sebagai sampling rate, dan parameter `n_steps` sebagai jumlah dari semi tone yang dinaikkan dalam proses pitch shifting ini.

B. Perbedaan dalam representasi visual antara suara asli dan suara yang telah dimodifikasi

Berdasarkan dari hasil perbandingan waveform diatas bisa dilihat untuk pola dari waveform nya masih sama dengan suara asli tetapi untuk amplitudo tiap gelombangnya mengalami penurunan bisa dilihat jika pitch semitone nya makin tinggi maka amplitudonya juga ikut semakin rendah, sedangkan pada spektrogram perbandingan terlihat frekuensi suara yang dihasilkan semakin besar dari suara asli yang hanya sekitar 16 kHz menjadi 22 kHz.

C. Bagaimana perubahan pitch memengaruhi kualitas dan kejelasan suara

Kualitas suara yang dihasilkan setelah dilakukan proses pitch shifting membuat suara menjadi lebih cempreng seperti chipmunk dan semakin tinggi pitch semitone yang pada suara tersebut maka kejelasan suara juga semakin berkurang terlihat dari perbedaan suara dengan semitone +7 masih terdengar jelas dan pada semitone +12 suara sudah mulai kedengeran kurang jelas.

5. Menggabungkan 2 rekaman yang sudah dilakukan pitch shifting ke dalam satu audio.

In [87]:

```
import numpy as np
import soundfile as sf

# Gabungkan suara chipmunk +7 dan +12
combined = np.concatenate((y_pitch7, y_pitch12))

# Simpan hasil gabungan
sf.write("rekaman Rafki_chipmunk.wav", combined, sr)
```

```
print("File gabungan berhasil dibuat: rekaman_rafki_chipmunk.wav")
display(Audio(combined, rate=sr))
```

File gabungan berhasil dibuat: rekaman_rafki_chipmunk.wav

▶ 0:00 / 0:48 ━━━━ 🔊 ⋮

Soal 4: Audio Processing Chain

1. Load audio yang dihasilkan dari proses pitch shifting pada soal nomor 3.

In [88]:

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

# Muat audio hasil pitch shifting
y, sr = librosa.load("rekaman_rafki_chipmunk.wav", sr=None)

print(f"Sample rate: {sr} Hz")
print(f"Durasi awal: {len(y)/sr:.2f} detik")

# Putar audio awal
print("Audio sebelum processing:")
display(Audio(y, rate=sr))
```

Sample rate: 48000 Hz

Durasi awal: 48.94 detik

Audio sebelum processing:

▶ 0:00 / 0:48 ━━━━ 🔊 ⋮

2. Proses Equalizer.

```
In [89]: import scipy.signal as signal

# Fungsi EQ sederhana
def equalizer(y, sr):
    # high-pass filter (hilangkan frekuensi di bawah 100 Hz)
    b_hp, a_hp = signal.butter(4, 100/(sr/2), btype='high')
    y_hp = signal.filtfilt(b_hp, a_hp, y)
    # low-pass filter (hilangkan frekuensi di atas 8000 Hz)
    b_lp, a_lp = signal.butter(4, 8000/(sr/2), btype='low')
    y_eq = signal.filtfilt(b_lp, a_lp, y_hp)
    return y_eq

y_eq = equalizer(y, sr)
print("Hasil suara dari proses equalizer")
display(Audio(y_eq, rate=sr))
```

Hasil suara dari proses equalizer



3. Proses Gain/fade

```
In [90]: # Gain boost + Fade
gain_db = 3 # tambahkan 3 dB
gain = 10 ** (gain_db / 20)
y_gain = y_eq * gain

# Fade in dan fade out selama 0.5 detik
fade_len = int(sr * 0.5)
fade_in = np.linspace(0, 1, fade_len)
fade_out = np.linspace(1, 0, fade_len)
y_gain[:fade_len] *= fade_in
y_gain[-fade_len:] *= fade_out
```

```
print("Hasil suara dari proses Gain dan Fade")
display(Audio(y_gain, rate=sr))
```

Hasil suara dari proses Gain dan Fade

▶ 0:00 / 0:48 ⏸ :

3. Proses Normalization

```
In [91]: import pyloudnorm as pyln

# Normalisasi Peak
y_peak = y_gain / np.max(np.abs(y_gain))

# Normalisasi LUFS ke -16 LUFS
meter = pyln.Meter(sr)
loudness = meter.integrated_loudness(y_gain)
y_lufs = pyln.normalize.loudness(y_gain, loudness, -16.0)

print(f'Loudness awal: {loudness:.2f} LUFS → setelah normalisasi: -16.0 LUFS')
display(Audio(y_lufs, rate=sr))
```

Loudness awal: -20.94 LUFS → setelah normalisasi: -16.0 LUFS

```
/usr/local/lib/python3.12/dist-packages/pyloudnorm/normalize.py:62: UserWarning: Possible clipped samples in output.
  warnings.warn("Possible clipped samples in output.")
```

▶ 0:00 / 0:48 ⏸ :

4. Proses Compression dan Noise Gate

```
In [92]: def compressor(y, threshold=-20, ratio=4, makeup_gain=5):
    y_db = 20 * np.log10(np.abs(y) + 1e-6)
    over = y_db > threshold
    y_db[over] = threshold + (y_db[over] - threshold) / ratio
    y_lin = 10 ** ((y_db + makeup_gain) / 20)
```

```
return np.sign(y) * y_lin

def noise_gate(y, threshold_db=-40):
    y_db = 20 * np.log10(np.abs(y) + 1e-6)
    mask = y_db > threshold_db
    return y * mask

y_comp = compressor(y_lufs)
y_gate = noise_gate(y_comp)

print("Hasil dari proses Compression dan Noise Gate")
display(Audio(y_gate, rate=sr))
```

Hasil dari proses Compression dan Noise Gate

▶ 0:00 / 0:48 ━━━━ 🔊 ⋮

5. Proses Silence trimming dan menyimpan hasil akhir pemprosesan

```
In [93]: y_trimmed, _ = librosa.effects.trim(y_gate, top_db=25)

sf.write("rekaman Rafki_processed.wav", y_trimmed, sr)
print("File akhir disimpan: rekaman Rafki_processed.wav")
display(Audio(y_trimmed, rate=sr))
```

File akhir disimpan: rekaman Rafki_processed.wav

▶ 0:00 / 0:46 ━━━━ 🔊 ⋮

6. Visualisasikan waveform dan spektrogram sebelum dan sesudah proses normalisasi.

```
In [94]: plt.figure(figsize=(12, 6))
plt.subplot(2,1,1)
librosa.display.waveform(y, sr=sr)
plt.title("Waveform Sebelum Normalisasi")
```

```
plt.subplot(2,1,2)
librosa.display.waveshow(y_trimmed, sr=sr)
plt.title("Waveform Sesudah Normalisasi dan Processing")
plt.tight_layout()
plt.show()

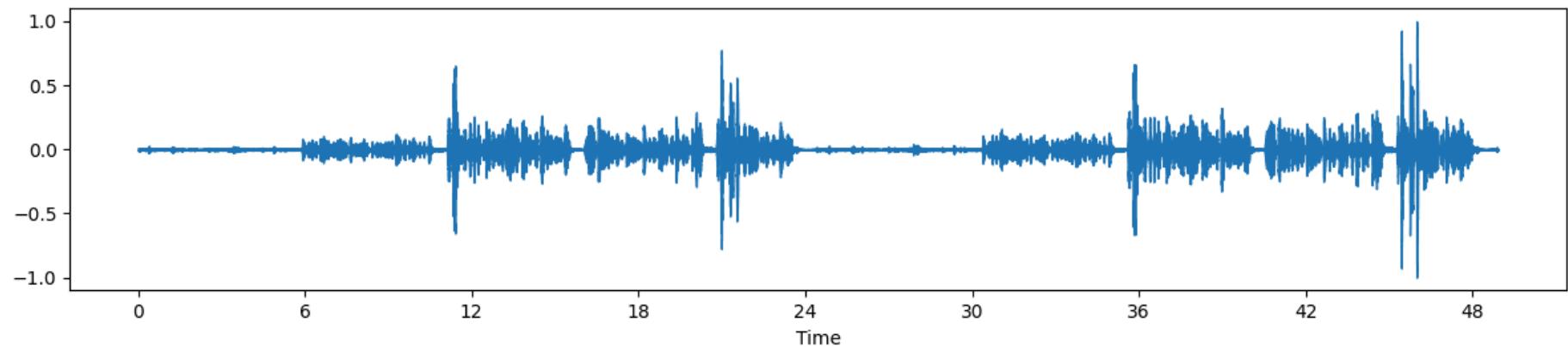
# Spektrogram perbandingan
plt.figure(figsize=(12, 8))

def plot_spec(y, sr, title):
    D = np.abs(librosa.stft(y))
    DB = librosa.amplitude_to_db(D, ref=np.max)
    librosa.display.specshow(DB, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    plt.title(title)
    plt.colorbar(format="%+2.0f dB")

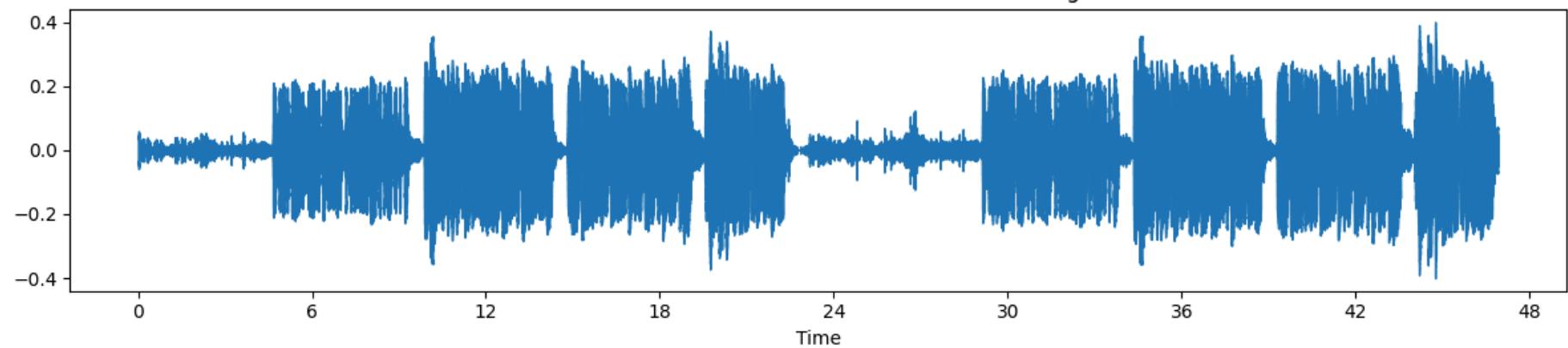
plt.subplot(2,1,1)
plot_spec(y, sr, "Spektrogram Sebelum Processing")

plt.subplot(2,1,2)
plot_spec(y_trimmed, sr, "Spektrogram Sesudah Processing")
plt.tight_layout()
plt.show()
```

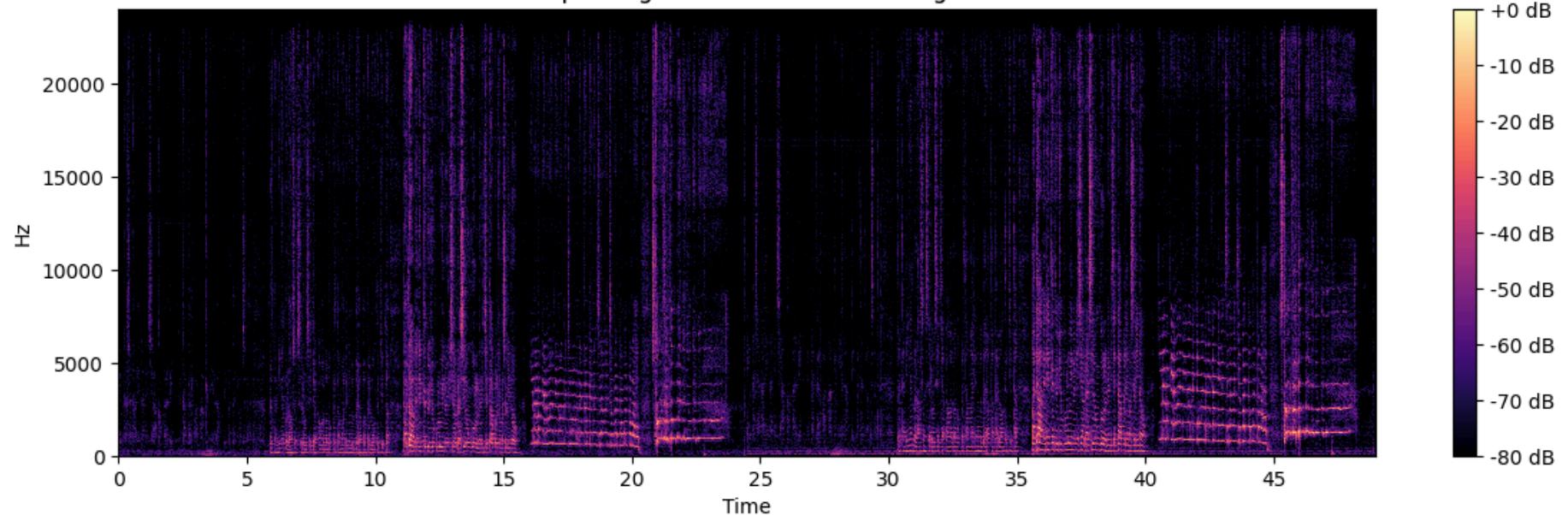
Waveform Sebelum Normalisasi



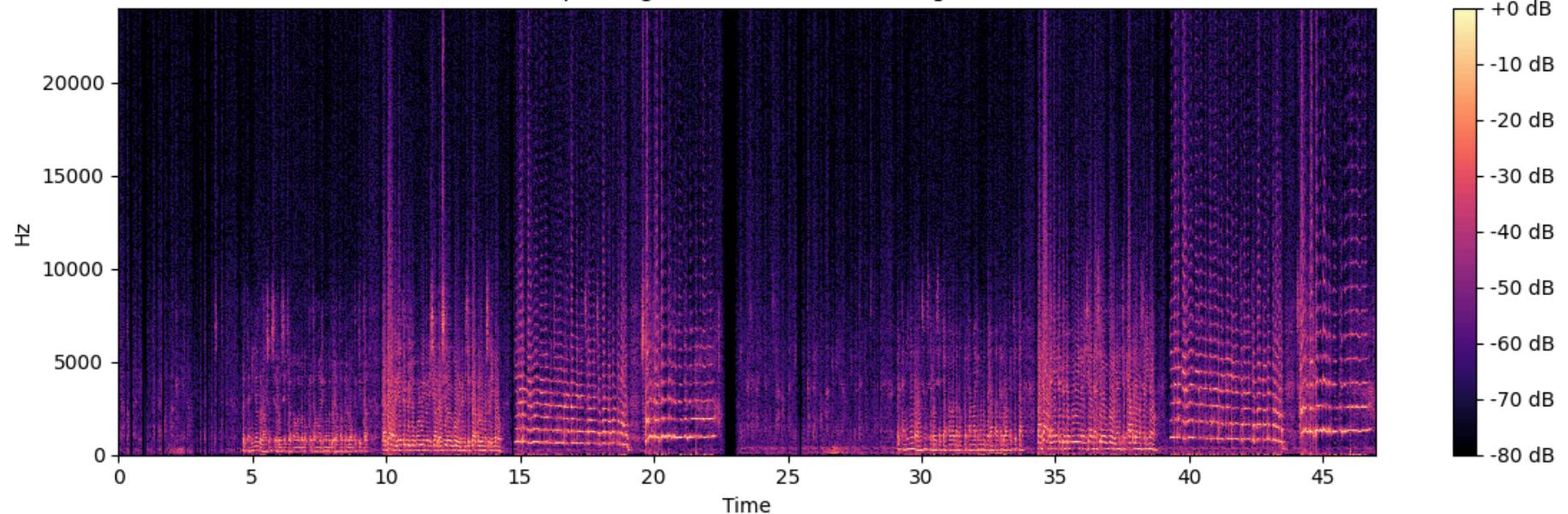
Waveform Sesudah Normalisasi dan Processing



Spektrogram Sebelum Processing



Spektrogram Sesudah Processing



7. Analisis setelah dilakukan pemrosesan dengan beberapa teknik pemrosesan pada audio.

A. Perubahan dinamika suara yang terjadi

Berdasarkan hasil perbandingan waveform dan spektrogram audio sebelum dan setelah melakukan pemrosesan bisa dilihat bahwa gelombang suara menjadi lebih rapat dan seimbang sehingga perbedaan antara suara pelan dan keras menjadi lebih kecil,

B. Perbedaan antara normalisasi peak dan normalisasi LUFS

Peak normalization hanya fokus ke nilai teknis, sedangkan LUFS normalization lebih berfokus pada kenyamanan pendengaran dan standar broadcast/audio digital. Jadi Dalam audio ini penyesuaian ke -16 LUFS menghasilkan loudness yang lebih baik dan seragam.

C. Bagaimana kualitas suara berubah setelah proses normalisasi dan loudness optimization

Setelah dilakukan semua pemrosesan suara, jadi suara yang dihasilkan berbeda" tiap jenis suaranya terlihat pada suara di 5 detik awal yang aslinya merupakan suara kecil atau bisik dan setelah dilakukan pemrosesan suara tersebut jadi semakin kecil karena dilakukan proses silence trim dan beberapa jenis suara ada yang berubah juga jadi semakin besar.

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

Berdasarkan aspek kualitas suara yang dihasilkan kelebihannya yaitu volume menjadi konsisten dan jelas di seluruh bagian sedangkan kekurangannya bisa kehilangan dinamika alami (terlalu datar) dan dari segi standar industri kelebihannya Sesuai standar platform (-16 LUFS untuk YouTube, Spotify, dsb) sedangkan kekurangannya jika terlalu agresif, bisa membuat suara terdengar "terkompresi".

Soal 5: Music Analysis dan Remix

1. Load 2 lagu yang ingin di remix dan melakukan potong pada bagian lagu yang ingin digunakan.

In [95]:

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
```

```

import numpy as np
import soundfile as sf # Import soundfile

# --- Load kedua lagu ---
lagu_lambat, sr_lambat = librosa.load("/data-rekaman/lagu-lambat.wav", sr=None)
lagu_cepat, sr_cepat = librosa.load("/data-rekaman/lagu-cepat.wav", sr=None)

# --- Fungsi untuk memotong bagian tertentu ---
def potong_audio(audio, sr, start, end):
    return audio[int(start*sr):int(end*sr)]

# --- Potong bagian terbaik masing-masing Lagu ---
lagu1 = potong_audio(lagu_lambat, sr_lambat, 40, 80) # Let Her Go
lagu2 = potong_audio(lagu_cepat, sr_cepat, 30, 70) # Happy

# --- Simpan hasil potongan ke file (opsional) ---
sf.write("lagu1_potongan.wav", lagu1, sr_lambat) # Use sf.write
sf.write("lagu2_potongan.wav", lagu2, sr_cepat) # Use sf.write

#print hasil potongan
print("Hasil Potongan Lagu 1:")
display(Audio(lagu1, rate=sr_lambat))

print("Hasil Potongan Lagu 2:")
display(Audio(lagu2, rate=sr_cepat))

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

plt.subplot(2, 1, 1)
librosa.display.waveshow(lagu1, sr=sr_lambat, color='blue')
plt.title("Potongan Lagu 1 (Let Her Go - Passenger)")

plt.subplot(2, 1, 2)
librosa.display.waveshow(lagu2, sr=sr_cepat, color='orange')
plt.title("Potongan Lagu 2 (Happy - Pharrell Williams)")

plt.tight_layout()
plt.show()

```

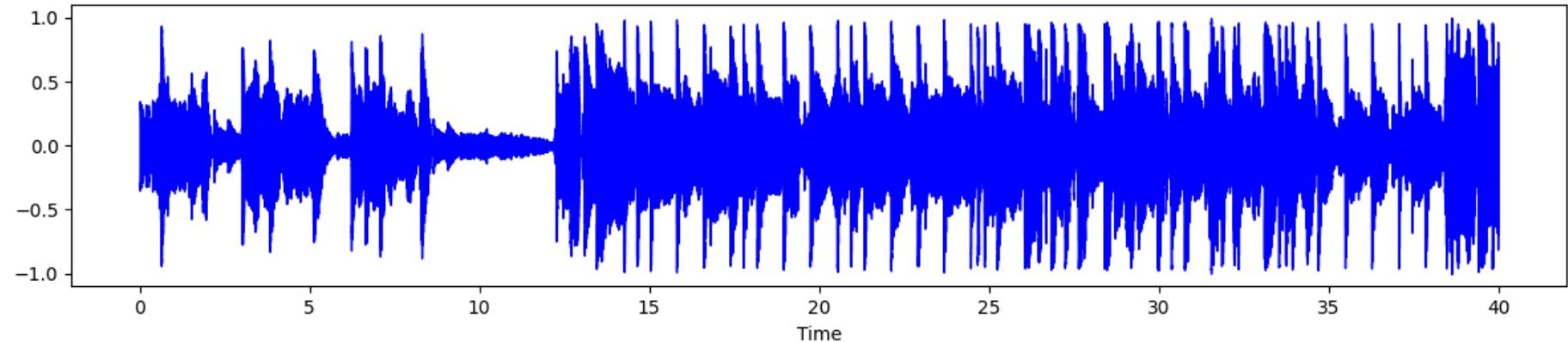
Hasil Potongan Lagu 1:

▶ 0:00 / 0:40 ⏸ 🔊 ⋮

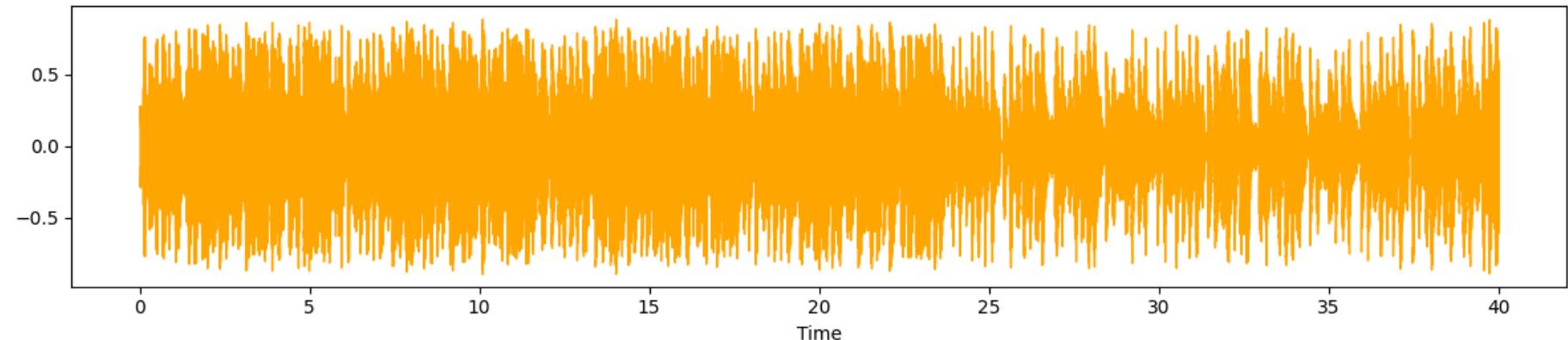
Hasil Potongan Lagu 2:

▶ 0:00 / 0:40 ⏸ 🔊 ⋮

Potongan Lagu 1 (Let Her Go - Passenger)



Potongan Lagu 2 (Happy - Pharrell Williams)



2. Analisis Tempo dan Key lagu.

In [96]:

```
import librosa
import numpy as np

# --- Analisis tempo (Beat Tracking) ---
tempo1, _ = librosa.beat.beat_track(y=lagu1, sr=sr_lambat)
tempo2, _ = librosa.beat.beat_track(y=lagu2, sr=sr_cepat)

# --- Estimasi key dengan chroma ---
chroma1 = librosa.feature.chroma_cqt(y=lagu1, sr=sr_lambat)
chroma2 = librosa.feature.chroma_cqt(y=lagu2, sr=sr_cepat)

# --- Dapatkan indeks nada dominan (0=C, 1=C#, 2=D, dst) ---
# Mengambil rata-rata across time and find the peak chroma
key1_idx = np.argmax(np.mean(chroma1, axis=1))
key2_idx = np.argmax(np.mean(chroma2, axis=1))

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

key1 = notes[key1_idx]
key2 = notes[key2_idx]

# Extract scalar tempo if it's an array
if isinstance(tempo1, np.ndarray):
    tempo1_scalar = tempo1.item()
else:
    tempo1_scalar = tempo1

if isinstance(tempo2, np.ndarray):
    tempo2_scalar = tempo2.item()
else:
    tempo2_scalar = tempo2

print(f"♪ Lagu 1 (Let Her Go): {tempo1_scalar:.2f} BPM | Perkiraan Key: {key1}")
print(f"♫ Lagu 2 (Happy): {tempo2_scalar:.2f} BPM | Perkiraan Key: {key2}")
```

♪ Lagu 1 (Let Her Go): 76.00 BPM | Perkiraan Key: D
♫ Lagu 2 (Happy): 161.50 BPM | Perkiraan Key: D

3. Proses Remix Lagu.

```
In [97]: import librosa
import numpy as np
from pydub import AudioSegment
import matplotlib.pyplot as plt

# Time Stretch: Samakan tempo kedua Lagu
tempo1_scalar = float(tempo1.item())
tempo2_scalar = float(tempo2.item())

target_tempo_scalar = (tempo1_scalar + tempo2_scalar) / 2

rate1 = tempo1_scalar / target_tempo_scalar
rate2 = tempo2_scalar / target_tempo_scalar

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

print(f"Time Stretch selesai | Tempo disamakan ke {target_tempo_scalar:.2f} BPM")

# Pitch Shift: Samakan kunci (key)
shift = (key1_idx - key2_idx)
lagu2_shift = librosa.effects.pitch_shift(y=lagu2_stretch, sr=float(sr_cepat), n_steps=shift)

print(f"Pitch Shift selesai | Lagu 2 digeser sebanyak {shift:+} nada")

# Crossfade: Gabungkan kedua Lagu dengan transisi halus
def to_segment(audio, sr):
    """Konversi numpy array ke AudioSegment"""
    audio_norm = audio / np.max(np.abs(audio))
    audio_16bit = np.int16(audio_norm * 32767)
    return AudioSegment(
        audio_16bit.tobytes(),
        frame_rate=int(sr),
        sample_width=2,
        channels=1
    )
```

```

seg1 = to_segment(lagu1_stretch, sr_lambat)
seg2 = to_segment(lagu2_shift, sr_cepat)

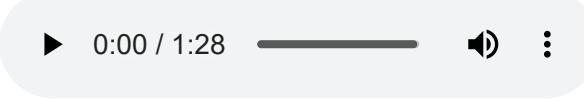
# Gabungkan dengan crossfade 3 detik
remix = seg1.append(seg2, crossfade=3000)
print("Crossfade 3 detik berhasil diterapkan")

# Simpan hasil remix akhir
output_file = "hasil_remix.wav"
remix.export(output_file, format="wav")
print(f"Hasil remix telah disimpan sebagai '{output_file}'")

# --- 5. Putar hasil langsung ---
display(Audio(output_file))

```

Time Stretch selesai | Tempo disamakan ke 118.75 BPM
 Pitch Shift selesai | Lagu 2 digeser sebanyak +0 nada
 Crossfade 3 detik berhasil diterapkan
 Hasil remix telah disimpan sebagai 'hasil_remix.wav'



▶ 0:00 / 1:28 ━━━━ 🔊 ⋮

4. Analisis proses dan parameter yang digunakan

Proses remix dilakukan melalui beberapa tahap utama, yaitu time stretching untuk menyamakan tempo kedua lagu, pitch shifting untuk menyesuaikan nada agar harmoninya serasi, serta crossfade selama tiga detik untuk menghasilkan transisi yang halus dan natural. Setelah ketiga tahap ini, kedua lagu dapat berpadu dengan baik tanpa perbedaan tempo atau nada yang mencolok, sehingga menghasilkan remix dengan perpaduan suara yang selaras dan enak didengar.

5. Visualisasi waveform dan spektrogram dari lagu yang sudah di remix.

In [98]:

```

import librosa.display

# Load hasil remix untuk analisis visual
y_remix, sr_remix = librosa.load("hasil_remix.wav", sr=None)

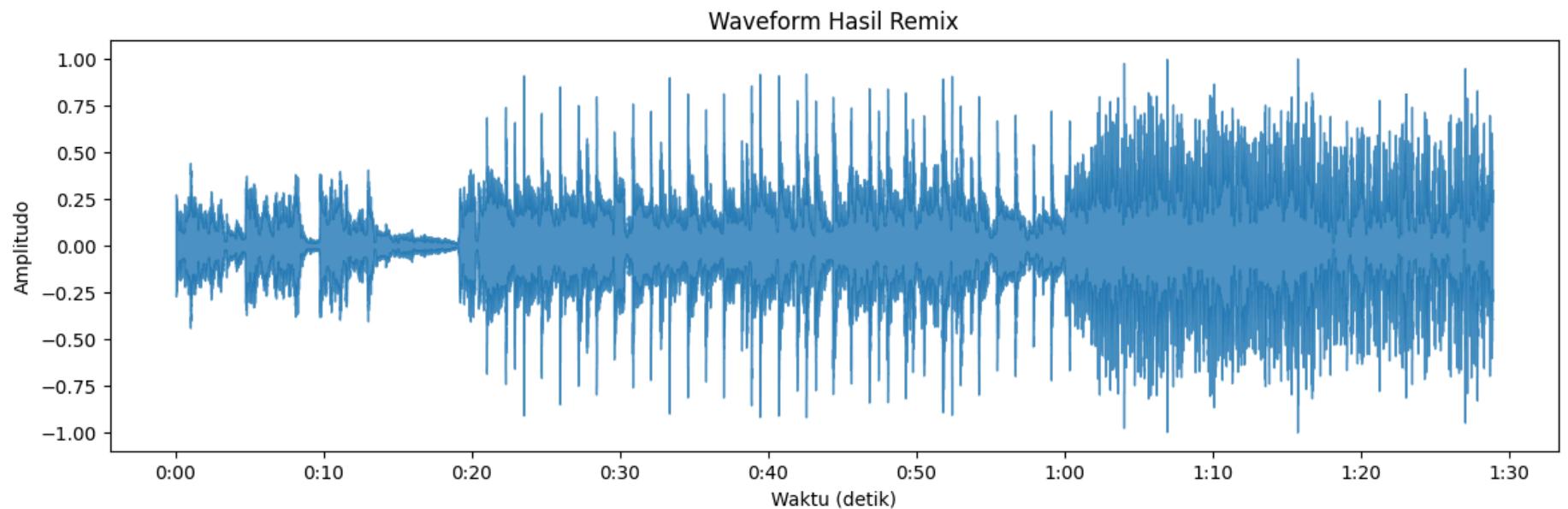
```

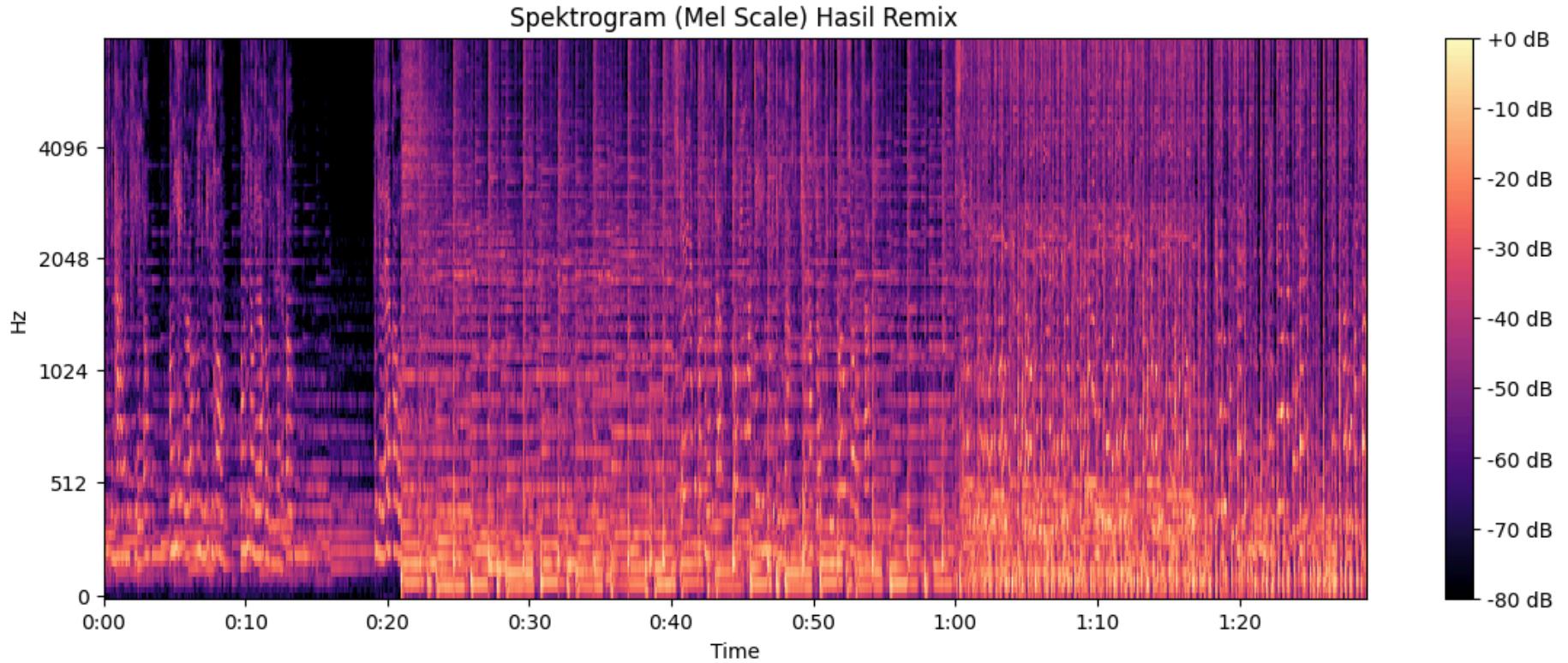
```

# Plot waveform
plt.figure(figsize=(14, 4))
librosa.display.waveshow(y_remix, sr=sr_remix, alpha=0.8)
plt.title("Waveform Hasil Remix")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.show()

# Plot spektrogram (mel-spectrogram)
plt.figure(figsize=(14, 5))
S = librosa.feature.melspectrogram(y=y_remix, sr=sr_remix, n_mels=128, fmax=8000)
S_dB = librosa.power_to_db(S, ref=np.max)
librosa.display.specshow(S_dB, sr=sr_remix, x_axis='time', y_axis='mel', fmax=8000, cmap='magma')
plt.title("Spektrogram (Mel Scale) Hasil Remix")
plt.colorbar(format='%.+2.0f dB')
plt.show()

```





Analisis waveform dan spektogram:

Berdasarkan hasil visualisasi waveform dan spektrogram pada remix lagu, terlihat bahwa amplitudo meningkat secara bertahap dari bagian awal hingga akhir, menunjukkan transisi yang halus antara lagu pertama yang bernuansa lambat dan lagu kedua yang lebih cepat serta energik. Pada bagian waveform, peralihan terlihat jelas di sekitar detik ke-20, di mana dinamika suara menjadi lebih padat dan intens. Hal ini menandakan proses crossfade berjalan dengan baik tanpa adanya jeda atau lonjakan volume yang tajam. Sementara itu, pada spektrogram terlihat bahwa frekuensi rendah hingga menengah mendominasi di awal (lagu lambat), kemudian frekuensi tinggi semakin kuat setelah transisi (lagu cepat). Pola spektral yang lebih padat di bagian akhir menunjukkan peningkatan energi suara akibat tempo yang lebih cepat dan karakter vokal yang lebih ceria. Secara keseluruhan, hasil remix menunjukkan perpaduan yang seimbang antara kedua lagu dengan transisi yang halus dan harmoni yang serasi baik dari segi tempo maupun nada.

5. Analisis hasil remix yang dilakukan

Dari hasil remix kedua lagu ini menghasilkan gabungan lagu yang cukup baik dari segi tempo maupun dinamika suara lagunya. penggunaan teknik penggabungan seperti time stretching, pitch shifting, dan crossfade pada lagu ini cukup baik dan dapat menghasilkan gabungan dua lagu. Untuk menghasilkan suara yang lebih bagus lagi bisa dengan menggunakan teknik filtering.

Referensi Pengerjaan

<https://chatgpt.com/share/68f23ab8-85c0-800a-8671-fe65133c163f>

Repository Github

<https://github.com/RafkiHaykhalAlif/Hans-On-Pemrosesan-Audio>