

# Hands On Audio Processing

Sistem & Teknologi Multimedia

Bintang Fikri Fauzan - 122140008

## Deskripsi Tugas

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

## Link repository GitHub

[GitHub](#)

In [ ]:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import librosa
import soundfile as sf
import os
import scipy.signal as signal
from IPython.display import Audio
from pydub import AudioSegment
from pydub.effects import compress_dynamic_range
from pydub.silence import detect_nonsilent
import pyloudnorm as pyln
```

In [ ]:

```
PATH_1 = os.path.join(os.getcwd(), 'media', 'Multimedia_1.wav')
PATH_2 = os.path.join(os.getcwd(), 'media', 'Multimedia_2.1.wav')
PATH_3 = os.path.join(os.getcwd(), 'media', 'Rex Orange County - Pluto Projector (0
PATH_4 = os.path.join(os.getcwd(), 'media', 'The Weeknd, Playboi Carti - Timeless (
```

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

1. Rekamlah suara Anda sendiri selama 25 detik dimana Anda membaca sebuah teks berita.
2. 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
3. Rekam dalam format WAV (atau konversikan ke WAV sebelum dimuat ke notebook).
4. Visualisasikan waveform dan spektrogram dari rekaman suara Anda.
5. Sertakan penjelasan singkat mengenai hasil visualisasi tersebut.
6. Lakukan resampling pada file audio Anda kemudian bandingkan kualitas dan durasinya

```
In [ ]: if not os.path.exists(PATH_1):
    raise FileNotFoundError(f"File not found: {PATH_1}")

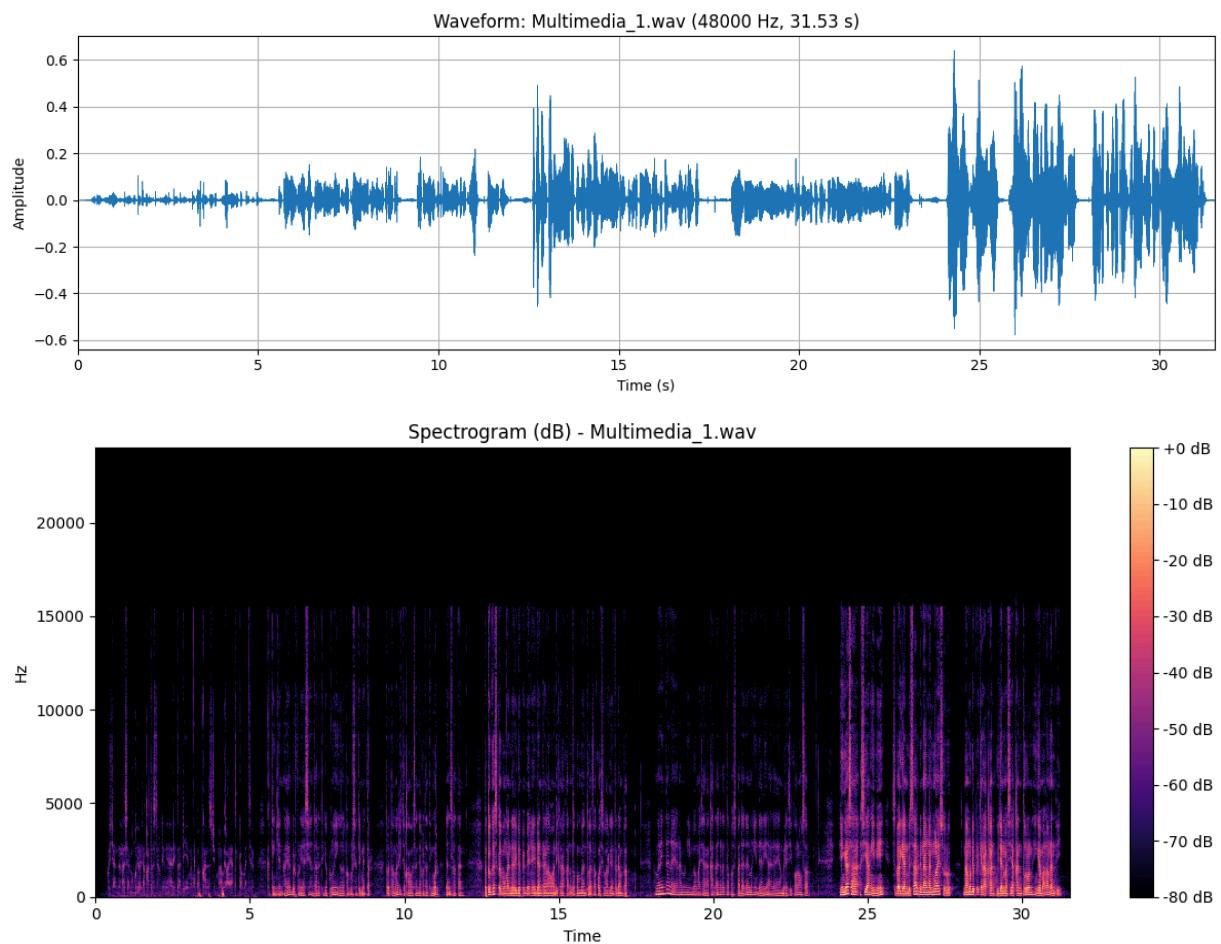
y, sr = librosa.load(PATH_1, sr=None, mono=True)
duration = len(y) / sr
t = np.linspace(0, duration, len(y))

plt.figure(figsize=(12, 4))
plt.plot(t, y, linewidth=0.5)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title(f"Waveform: {os.path.basename(PATH_1)} ({sr} Hz, {duration:.2f} s)")
plt.xlim(0, duration)
plt.grid(True)
plt.tight_layout()
plt.show()

# Spectrogram plot
n_fft = 2048
hop_length = 512

S = np.abs(librosa.stft(y, n_fft=n_fft, hop_length=hop_length))
S_db = librosa.amplitude_to_db(S, ref=np.max)

plt.figure(figsize=(12, 5))
librosa.display.specshow(S_db, sr=sr, hop_length=hop_length,
                        x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%.+2.0f dB')
plt.title(f"Spectrogram (dB) - {os.path.basename(PATH_1)}")
plt.ylim(0, sr/2)
plt.tight_layout()
plt.show()
```



## Penjelasan Visualisasi Waveform dan Spectrogram Audio Original

**Waveform:** Audio rekaman terlihat terbagi menjadi 5 segmentasi dimana tiap bagiannya terlihat lonjakan yang berbeda menandakan ciri khas masing masing segmentasi rekaman. Pada 5 detik pertama amplitudo waveform terlihat paling kecil, artinya suara di 5 detik pertama merupakan suara berbisik. Sedangkan amplitudo tertinggi terlihat pada 5 detik akhir yang artinya pada rentang waktu ini terekam suara bernada sangat tinggi (teriakan).

**Spectrogram:** Sama seperti waveform, di visualisasi spectrogram juga terlihat rekaman suara yang tersegmentasi menjadi 5 bagian dengan sebaran frekuensi yang berbeda beda. Pada detik awal, intensitas (dB) terlihat sangat rendah. Dilanjut diatas detik ke-5 sebaran intensitas terlihat meningkat namun masih belum signifikan, artinya suara sudah cukup tergambar dengan jelas. Lalu pada bagian setelahnya muncul lonjakan intensitas yang signifikan di frekuensi rendah, menandakan terdapat suara besar di awal. Lalu setelahnya intensitas kembali mengurang namun kerapatan dari intensitas itu meningkat, menandakan adanya perubahan tone suara. Dan di segmen terakhir terlihat nilai intensitas yang paling tinggi pada frekuensi rendah hingga menengah, menandakan suara yang dihasilkan bernada tinggi.

```
In [ ]: # Target Sample Rate
target_sr = 4000
y_resampled = librosa.resample(y, orig_sr=sr, target_sr=target_sr)

# Plot Resampled Waveform
duration = len(y_resampled) / target_sr
t = np.linspace(0, duration, len(y_resampled))

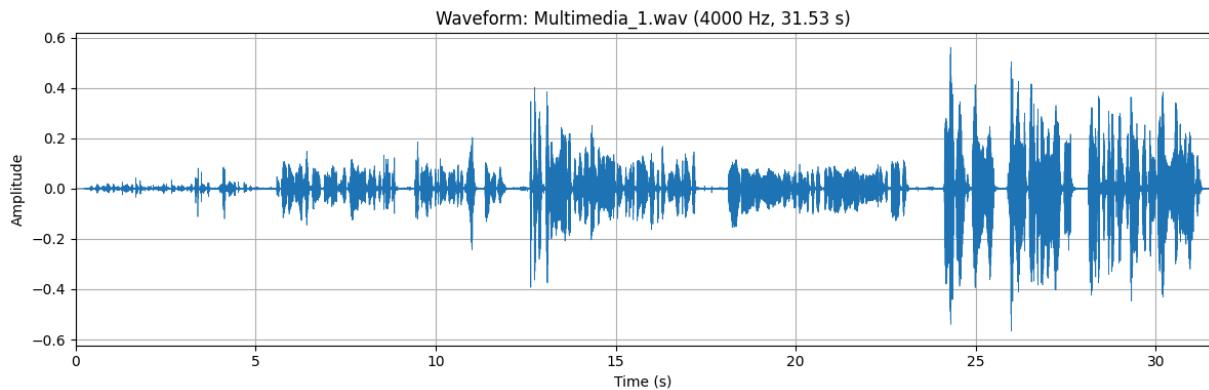
plt.figure(figsize=(12, 4))
plt.plot(t, y_resampled, linewidth=0.5)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title(f"Waveform: {os.path.basename(PATH_1)} ({target_sr} Hz, {duration:.2f} s")
plt.xlim(0, duration)
plt.grid(True)
plt.tight_layout()
plt.show()

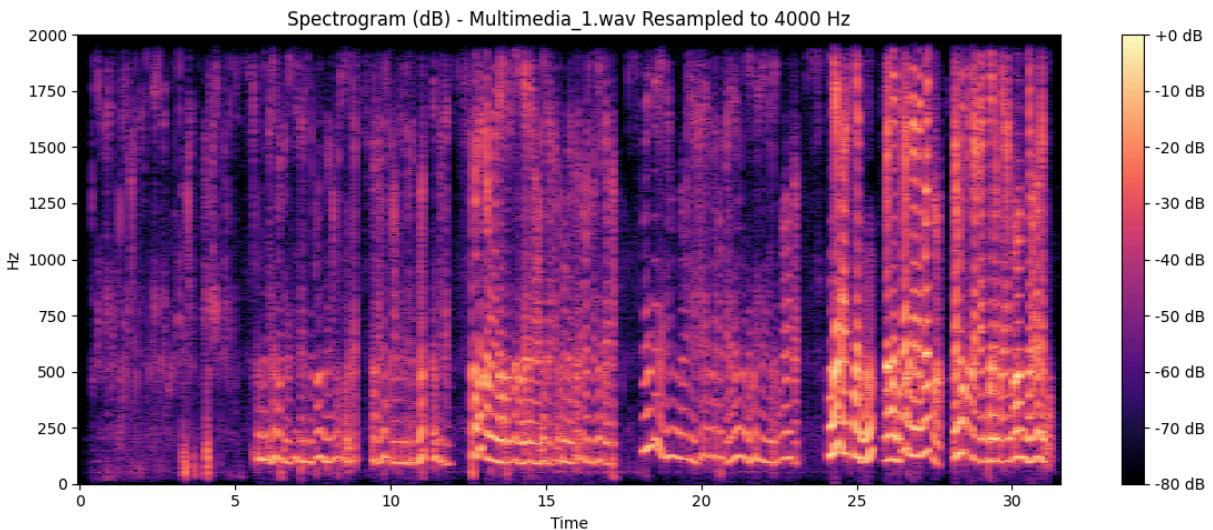
# Plot Resampled Spectrogram
S = np.abs(librosa.stft(y_resampled, n_fft=n_fft, hop_length=hop_length))
S_db = librosa.amplitude_to_db(S, ref=np.max)

plt.figure(figsize=(12, 5))
librosa.display.specshow(S_db, sr=target_sr, hop_length=hop_length,
                        x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%.+2.0f dB')
plt.title(f"Spectrogram (dB) - {os.path.basename(PATH_1)} Resampled to {target_sr}")
plt.ylim(0, target_sr/2)
plt.tight_layout()
plt.show()

# Preview Resampled Audio
print("Original Audio")
display(Audio(data=y, rate=sr))

print("Resampled Audio")
display(Audio(data=y_resampled, rate=target_sr))
```





Original Audio

▶ 0:00 / 0:31 ⏸ 🔊 ⋮

Resampled Audio

▶ 0:00 / 0:31 ⏸ 🔊 ⋮

```
In [ ]: # Specify the Time Range
zoom_start = 24.5
zoom_end = 24.55

# Create a time axis for each waveform.
t_orig = np.linspace(0, len(y) / sr, num=len(y))
t_resampled = np.linspace(0, len(y_resampled)) / target_sr, num=len(y_resampled))

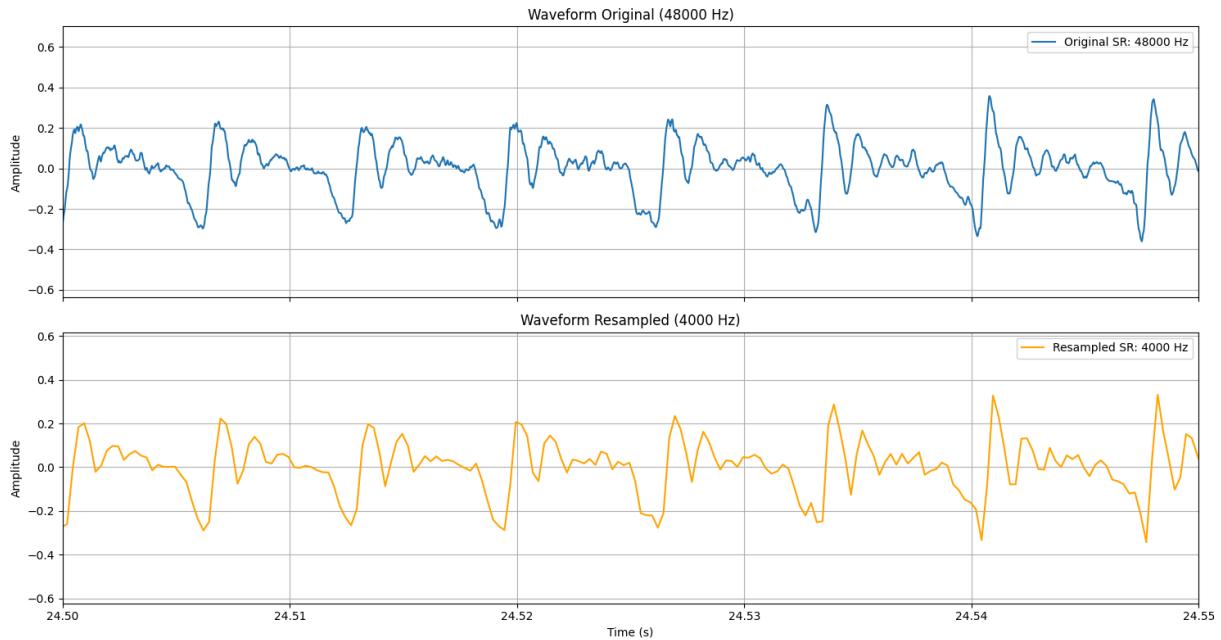
# Create Subplot
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15, 8), sharex=True)

# Plot Waveform Original
ax[0].plot(t_orig, y, label=f'Original SR: {sr} Hz')
ax[0].set_title(f"Waveform Original ({sr} Hz)")
ax[0].set_ylabel("Amplitude")
ax[0].grid(True)
ax[0].legend()

# Plot Resampled Waveform
ax[1].plot(t_resampled, y_resampled, label=f'Resampled SR: {target_sr} Hz', color='red')
ax[1].set_title(f"Waveform Resampled ({target_sr} Hz)")
ax[1].set_xlabel("Time (s)")
ax[1].set_ylabel("Amplitude")
ax[1].grid(True)
ax[1].legend()

# Zoomed in
plt.xlim(zoom_start, zoom_end)
```

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



```
In [ ]: # Save Resampled Audio
y_resampled_path = os.path.join(os.getcwd(), "media", "1_resampled.wav")
sf.write(y_resampled_path, y_resampled, target_sr)
```

## Penjelasan Audio Setelah dilakukan Resampling

Audio berhasil di resampling dari 48000 Hz menjadi 4000 Hz. Dari visual waveform yang telah di sandingkan, terlihat perbedaan waveform setelah dilakukan zoom-in hingga skala 0.05 detik. Terlihat gelombang audio yang telah dilakukan resample cenderung lebih kasar dengan beberapa sudut tajam. Ini dapat terjadi karena sample ratenya sudah jauh lebih rendah dari audio aslinya. Sehingga detail-detail halus pada titik samplenya hilang.

Dari spektrogram juga terlihat perbedaan karena sekarang hanya di batasi hingga 4000 Hz, maka batas atasnya menjadi 2000 Hz.

## Soal 2: Noise Reduction dengan Filtering

1. 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).
2. Gunakan filter equalisasi (high-pass, low-pass, dan band-pass) untuk menghilangkan noise pada rekaman tersebut.

3. Lakukan eksperimen dengan berbagai nilai frekuensi cutoff (misalnya 500 Hz, 1000 Hz, 2000 Hz).
4. Visualisasikan hasil dari tiap filter dan bandingkan spektrogramnya.
5. Jelaskan:
  - Jenis noise yang muncul pada rekaman Anda
  - Filter mana yang paling efektif untuk mengurangi noise tersebut
  - Nilai cutoff yang memberikan hasil terbaik
  - Bagaimana kualitas suara (kejelasan ucapan) setelah proses filtering

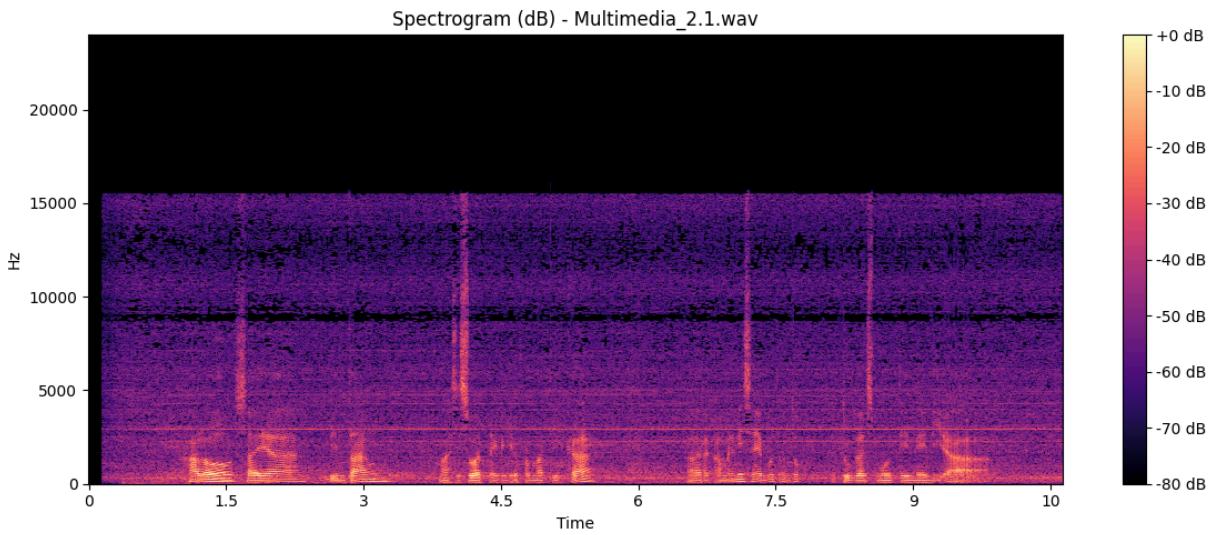
```
In [ ]: if not os.path.exists(PATH_2):
    raise FileNotFoundError(f"File not found: {PATH_2}")

y_2, sr_2 = librosa.load(PATH_2, sr=None, mono=True)
duration_2 = len(y_2) / sr_2
t = np.linspace(0, duration_2, len(y_2))

n_fft = 2048
hop_length = 512

S_2 = np.abs(librosa.stft(y_2, n_fft=n_fft, hop_length=hop_length))
S_db_2 = librosa.amplitude_to_db(S_2, ref=np.max)

plt.figure(figsize=(12, 5))
librosa.display.specshow(S_db_2, sr=sr_2, hop_length=hop_length,
                        x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%+2.0f dB')
plt.title(f"Spectrogram (dB) - {os.path.basename(PATH_2)}")
plt.ylim(0, sr_2/2)
plt.tight_layout()
```



```
In [ ]: # Apply Low-pass Filter
lowpass_cutoff = 1600
nyquist_freq = sr_2 / 2
lowpass_cutoff_normalized = lowpass_cutoff / nyquist_freq

# Butterworth Filter
```

```
b, a = signal.butter(5, lowpass_cutoff_normalized, btype='low', analog=False)
y_lowpass = signal.filtfilt(b, a, y_2)
```

```
In [ ]: # Apply High-pass Filter
highpass_cutoff = 400
nyquist_freq = sr_2 / 2
highpass_cutoff_normalized = highpass_cutoff / nyquist_freq

# Butterworth Filter
b, a = signal.butter(5, highpass_cutoff_normalized, btype='high', analog=False)
y_highpass = signal.filtfilt(b, a, y_2)
```

```
In [ ]: # Apply Band-pass Filter
lowcut = 400
highcut = 1600
nyquist_freq = sr_2 / 2
lowcut_normalized = lowcut / nyquist_freq
highcut_normalized = highcut / nyquist_freq

# Butterworth Filter
b, a = signal.butter(5, [lowcut_normalized, highcut_normalized], btype='band', analog=False)
y_bandpass = signal.filtfilt(b, a, y_2)
```

```
In [ ]: # Save Filtered Signal
lowpass_path = os.path.join(os.getcwd(), "media", "2_lowpass.wav")
highpass_path = os.path.join(os.getcwd(), "media", "2_highpass.wav")
bandpass_path = os.path.join(os.getcwd(), "media", "2_bandpass.wav")

sf.write(lowpass_path, y_lowpass, sr_2, format='WAV')
sf.write(highpass_path, y_highpass, sr_2, format='WAV')
sf.write(bandpass_path, y_bandpass, sr_2, format='WAV')
```

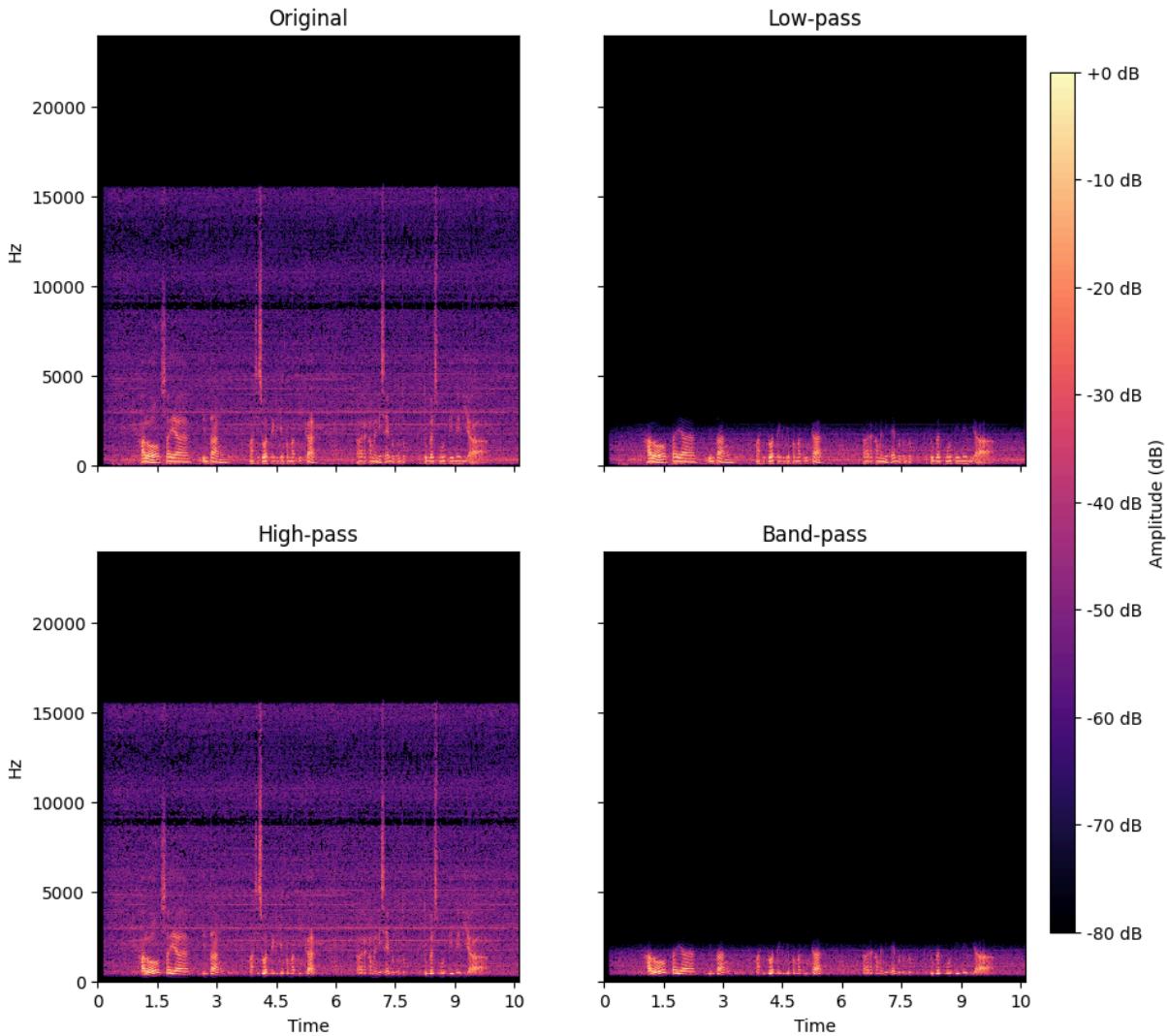
```
In [ ]: # Load File
def plot_spectrogram(ax, signal, sr, title):
    S = np.abs(librosa.stft(signal))
    S_db = librosa.amplitude_to_db(S, ref=np.max)
    img = librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz', cmap='magma')
    ax.set_title(title)
    ax.label_outer()
    return img

# Plot side by side Comparison
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
img0 = plot_spectrogram(axs[0, 0], y_2, sr, 'Original')
img1 = plot_spectrogram(axs[0, 1], y_lowpass, sr, 'Low-pass')
img2 = plot_spectrogram(axs[1, 0], y_highpass, sr, 'High-pass')
img3 = plot_spectrogram(axs[1, 1], y_bandpass, sr, 'Band-pass')
plt.subplots_adjust(right=0.88)

# Add Colorbar
cbar_ax = fig.add_axes([0.9, 0.15, 0.02, 0.7])
fig.colorbar(img3, cax=cbar_ax, format='%.2f dB').set_label('Amplitude (dB)')
plt.show()

# Preview Audio
```

```
print(f"Original Audio")
display(Audio(data=y_2, rate=sr_2))
print(f"Lowpass Filter Applied")
display(Audio(data=y_lowpass, rate=sr_2))
print(f"Highpass Filter Applied")
display(Audio(data=y_highpass, rate=sr_2))
print(f"Bandpass Filter Applied")
display(Audio(data=y_bandpass, rate=sr_2))
```



Original Audio

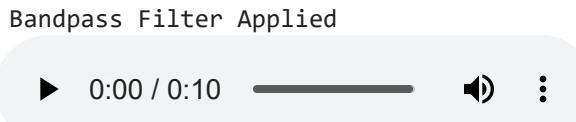
▶ 0:00 / 0:10 ━━━━ 🔊 ⋮

Lowpass Filter Applied

▶ 0:00 / 0:10 ━━━━ 🔊 ⋮

Highpass Filter Applied

▶ 0:00 / 0:10 ━━━━ 🔊 ⋮



## Penjelasan Noise Reduction

1. Noise yang muncul pada rekaman saya adalah suara mesin air yang sedang menyala.
2. Berdasarkan hasil filter suara yang telah dilakukan, Bandpass filter merupakan hasil yang paling efektif di antara filter lainnya. Saya melakukan bandpass filter dengan lowcut di frekuensi 400 Hz dan highcut di frekuensi 1600 Hz.
3. Nilai cutoff yang memberikan hasil terbaik adalah pada Highcut di frekuensi 1600 Hz. Karena dapat memfilter suara dengung dari mesin air yang merupakan suara dengan frekuensi tinggi.
4. Vokal dari rekaman suara menjadi lebih jelas karena tidak ada background noise yang mengganggu. Namun secara kualitas, suara yang ditangkap menjadi terkesan sedikit "tenggelam".

## Soal 3: Pitch Shifting dan Audio Manipulation

1. Lakukan pitch shifting pada rekaman suara Soal 1 untuk membuat suara terdengar seperti chipmunk (dengan mengubah pitch ke atas).
  2. Visualisasikan waveform dan spektrogram sebelum dan sesudah pitch shifting.
  3. 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
  4. Gunakan dua buah pitch tinggi, misalnya pitch +7 dan pitch +12.
  5. Gabungkan kedua rekaman yang telah di-pitch shift ke dalam satu file audio.
- (Gunakan ChatGPT / AI untuk membantu Anda dalam proses ini)

```
In [ ]: # Pitch Shifting
def pitch_shift(signal, sr, n_steps):
    return librosa.effects.pitch_shift(signal, sr=sr, n_steps=n_steps)

y1_pitch_shifted_7 = pitch_shift(y, sr, n_steps=7)
y1_pitch_shifted_12 = pitch_shift(y, sr, n_steps=12)

# Side by Side Waveform Comparasion
plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
```

```

librosa.display.waveshow(y, sr=sr, color='darkgreen', alpha=0.7)
plt.title('Waveform - Audio Original', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 2)
librosa.display.waveshow(y1_pitch_shifted_7, sr=sr, color='red', alpha=0.7)
plt.title('Waveform - Audio Pitch Shifted +7', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 3)
librosa.display.waveshow(y1_pitch_shifted_12, sr=sr, color='darkblue', alpha=0.7)
plt.title('Waveform - Audio Pitch Shifted +12', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.xlabel("Waktu (Detik)", fontsize=12)

plt.tight_layout(pad=3.0)
plt.show()

# Side by Side Spectrogram Comparasion
N_fft = 2048
hop_length = 512
audio_data = [y, y1_pitch_shifted_7, y1_pitch_shifted_12]
labels = ["Audio Original", "Audio Pitch Shifted +7", "Audio Pitch Shifted +12"]
D1, D2, D3 = [np.abs(librosa.stft(y, n_fft=N_fft, hop_length=hop_length)) for y in
S_db1, S_db2, S_db3 = [librosa.amplitude_to_db(D, ref=np.max) for D in [D1, D2, D3]

# v_min and v_max global
v_min = min(S_db1.min(), S_db2.min(), S_db3.min())
v_max = max(S_db1.max(), S_db2.max(), S_db3.max())

plt.figure(figsize=(12, 10))

# Spectrogram Plot
plt.subplot(3, 1, 1)
librosa.display.specshow(S_db1,
                        sr=sr,
                        x_axis='time',
                        y_axis='hz',
                        hop_length=hop_length,
                        cmap='magma', # Colormap
                        vmin=v_min,
                        vmax=v_max)
plt.title(f'Spektrogram - {labels[0]}', fontsize=14)
plt.ylabel('Frekuensi (Hz)')
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 2)
librosa.display.specshow(S_db2,
                        sr=sr,
                        x_axis='time',
                        y_axis='hz',
                        hop_length=hop_length,
                        cmap='magma',
                        vmin=v_min,

```

```

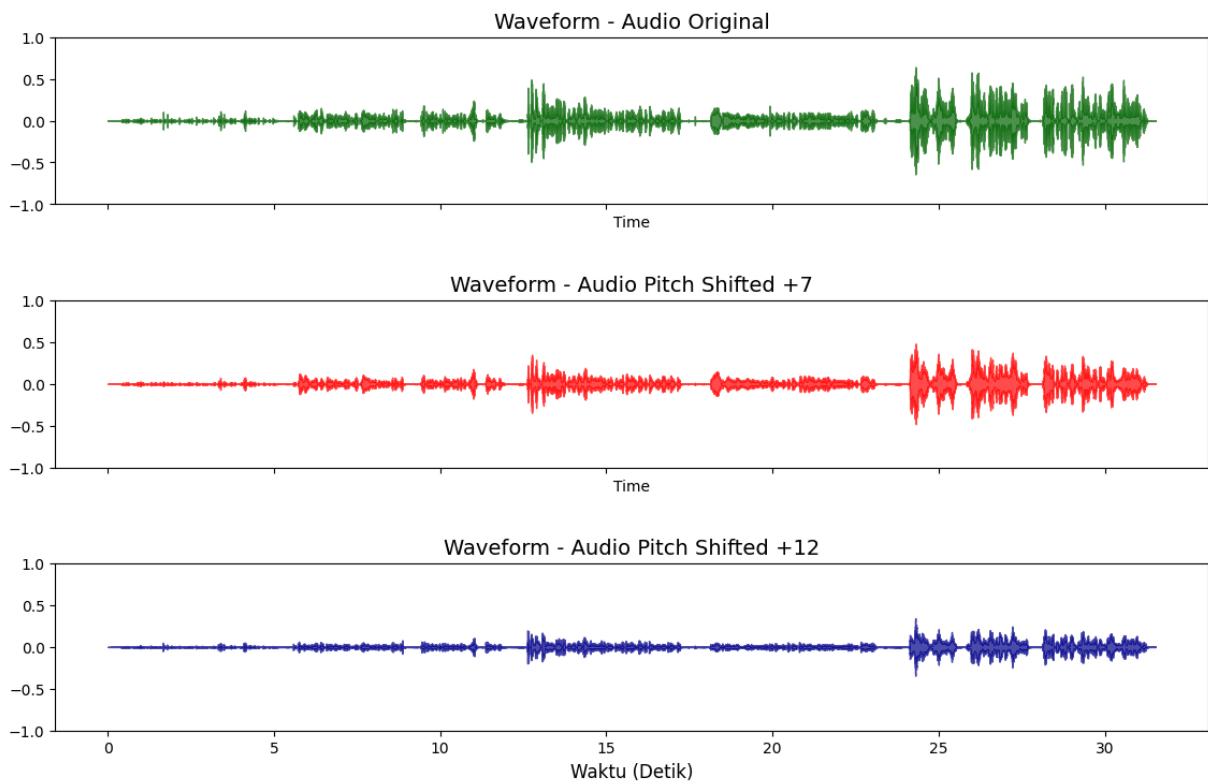
    vmax=v_max)
plt.title(f'Spekrogram - {labels[1]}', fontsize=14)
plt.ylabel('Frekuensi (Hz)')
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 3)
img = librosa.display.specshow(S_db3,
                               sr=sr,
                               x_axis='time',
                               y_axis='hz',
                               hop_length=hop_length,
                               cmap='magma',
                               vmin=v_min,
                               vmax=v_max)
plt.title(f'Spekrogram - {labels[2]}', fontsize=14)
plt.ylabel('Frekuensi (Hz)')
plt.xlabel('Waktu (Detik)', fontsize=12) # Tampilkan Label X

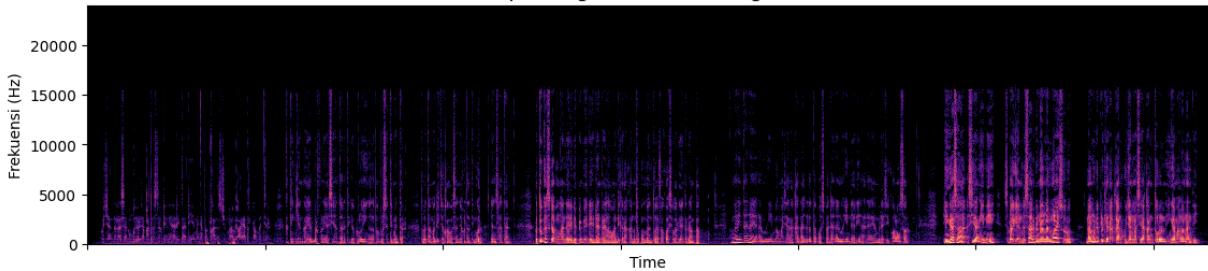
plt.tight_layout(pad=3.0)
plt.show()

# Preview Audio
print("Original Audio")
display(Audio(data=y, rate=sr))
print("Pitch Shifted Audio (+7)")
display(Audio(data=y1_pitch_shifted_7, rate=sr))
print("Pitch Shifted Audio (+12)")
display(Audio(data=y1_pitch_shifted_12, rate=sr))

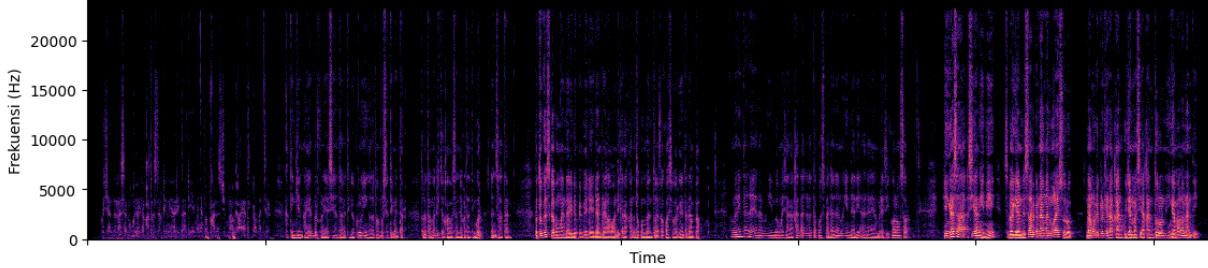
```



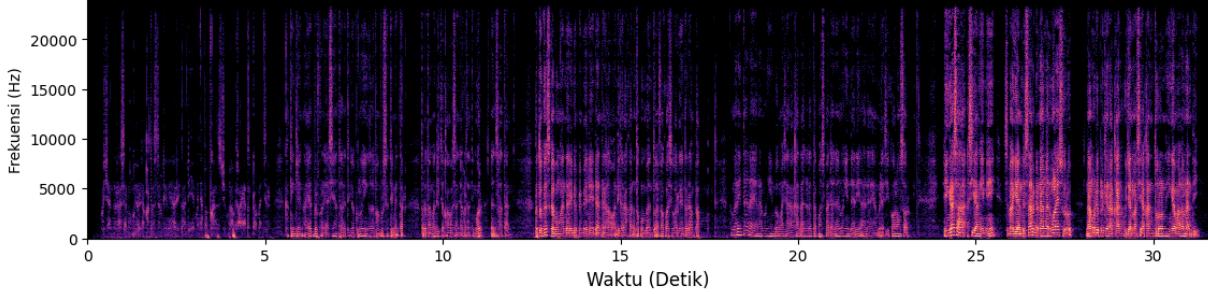
Spektrogram - Audio Original



Spektrogram - Audio Pitch Shifted +7



Spektrogram - Audio Pitch Shifted +12



Original Audio

▶ 0:00 / 0:31 ⏸ 🔊 ⋮

Pitch Shifted Audio (+7)

▶ 0:00 / 0:31 ⏸ 🔊 ⋮

Pitch Shifted Audio (+12)

▶ 0:00 / 0:31 ⏸ 🔊 ⋮

```
In [ ]: # Combine Pitch Shifted Audio
combined_audio = np.concatenate([y1_pitch_shifted_7, y1_pitch_shifted_12])

print(f"Combined Audio")
display(Audio(data=combined_audio, rate=sn))
```

Combined Audio

▶ 0:00 / 1:03 ⏸ ⏴ ⏵

```
In [ ]: # Save Pitch Shifted Audio
pitch_shifted_7_path = os.path.join(os.getcwd(), "media", "3_pitch_shifted_7.wav")
sf.write(pitch_shifted_7_path, y1_pitch_shifted_7, sr)
pitch_shifted_12_path = os.path.join(os.getcwd(), "media", "3_pitch_shifted_12.wav")
sf.write(pitch_shifted_12_path, y1_pitch_shifted_12, sr)
combined_audio_path = os.path.join(os.getcwd(), "media", "3_combined_audio.wav")
sf.write(combined_audio_path, combined_audio, sr)
```

## Penjelasan Pitch Shifting Audio

1. Proses pitch shifting dilakukan secara otomatis menggunakan library librosa. Di dalam library librosa, terdapat tahapan proses pitch shifting dimulai dengan melakukan Short-Time Fourier Transformation (STFT) dilanjut dengan melakukan time-streching, dan diakhiri dengan me-resampling kembali ke sample rate aslinya. Seluruh tahapan ini, membutuhkan parameter sebagai berikut:
  - **n\_steps** yaitu berapa banyak pitch naik/turunnya.
  - **sr** yaitu sample rate original.
  - **res\_type** yaitu kualitas hasil resampling
2. Secara visual, hasil pitch shifting dapat dilihat dengan rincian sebagai berikut.
  - **waveform:**
  - **spectrogram:**
- 3.

## Soal 4: Audio Processing Chain

1. Lakukan processing pada rekaman yang sudah di-pitch shift pada Soal 3 dengan tahapan:
  - Equalizer
  - Gain/fade
  - Normalization
  - Compression
  - Noise Gate
  - Silence trimming
2. Atur nilai target loudness ke -16 LUFS.
3. Visualisasikan waveform dan spektrogram sebelum dan sesudah proses normalisasi.
4. 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

```
In [ ]: # Equilizer (Bandpass Filter)
def apply_equalizer(signal, sr, lowcut, highcut):
    nyquist_freq = sr / 2
    lowcut_normalized = lowcut / nyquist_freq
    highcut_normalized = highcut / nyquist_freq

    b, a = signal
    return signal.filtfilt(b, a, signal)

# Gain Control
gain_db = 5
gain = 10 ** (gain_db / 20)
y_gained = y1_pitch_shifted_7 * gain

# Fade Control
fade_in_duration = int(2 * sr) # 2 detik fade in
fade_out_duration = int(2 * sr) # 2 detik fade out

fade_in = np.linspace(0, 1, fade_in_duration)
fade_out = np.linspace(1, 0, fade_out_duration)

y_gained[:fade_in_duration] *= fade_in
y_gained[-fade_out_duration:] *= fade_out

# Normalization
normalized_audio = y_gained / np.max(np.abs(y_gained))

# Dynamic Range Compression
def compressor(x, threshold=0.2, ratio=4.0):
    x_clipped = np.copy(x)
    mask = np.abs(x) > threshold
    x_clipped[mask] = np.sign(x[mask]) * (threshold + (np.abs(x[mask]) - threshold) /
    return x_clipped
audio_comp = compressor(normalized_audio)

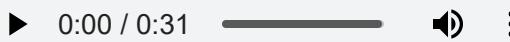
# Noise Gate
def noise_gate(audio, threshold=0.02):
    gated = np.where(np.abs(audio) < threshold, 0, audio)
    return gated

audio_gated = noise_gate(audio_comp)

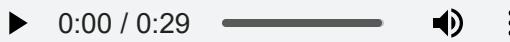
# Silence Trimming
trimmed, _ = librosa.effects.trim(audio_gated, top_db=30)

# Preview the Audio
print("Original Audio")
display(Audio(data=y1_pitch_shifted_7, rate=sr))
print("Processed Audio")
display(Audio(data=trimmed, rate=sr))
```

Original Audio



Processed Audio



In [ ]:

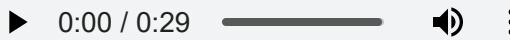
```
# Loudness Control
meter = pyln.Meter(sr) # EBU R128 loudness meter
loudness = meter.integrated_loudness(trimmed)

# Target Loudness
target_loudness = -16.0

# Normalisasi ke target
loudness_normalized_audio = pyln.normalize.loudness(trimmed, loudness, target_loudn

# Preview the Audio
print("Processed Audio (Target Loudness -16 LUFS)")
display(Audio(data=loudness_normalized_audio, rate=sr))
```

Processed Audio (Target Loudness -16 LUFS)



In [ ]:

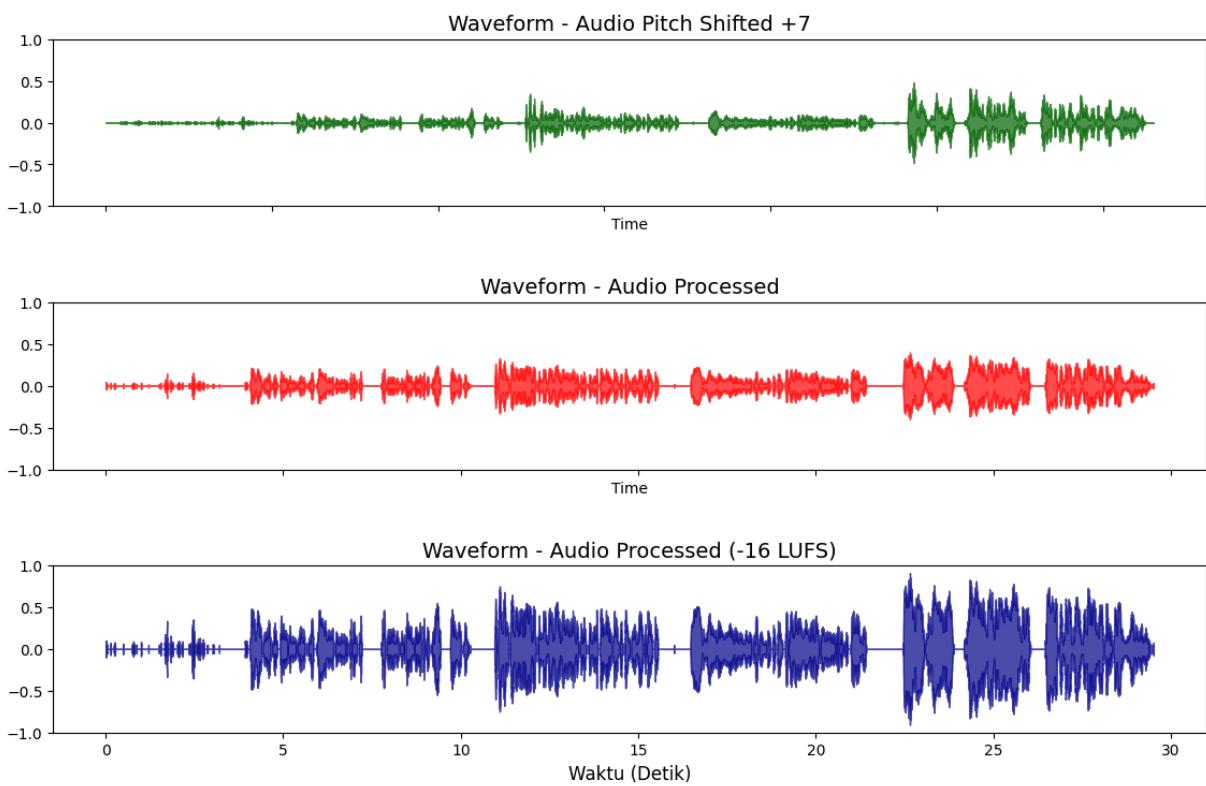
```
# Side by Side Waveform Comparasion
plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
librosa.display.waveshow(y1_pitch_shifted_7, sr=sr, color='darkgreen', alpha=0.7)
plt.title('Waveform - Audio Pitch Shifted +7', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 2)
librosa.display.waveshow(trimmed, sr=sr, color='red', alpha=0.7)
plt.title('Waveform - Audio Processed', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 3)
librosa.display.waveshow(loudness_normalized_audio, sr=sr, color='darkblue', alpha=
plt.title('Waveform - Audio Processed (-16 LUFS)', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.xlabel("Waktu (Detik)", fontsize=12)

plt.tight_layout(pad=3.0)
plt.show()
```



In [139...]

```
# Side by Side Waveform Comparasion
plt.figure(figsize=(12, 12))

plt.subplot(6, 1, 1)
librosa.display.waveshow(y1_pitch_shifted_7, sr=sr, color='darkgreen', alpha=0.7)
plt.title('Waveform - Audio Pitch Shifted +7 (Original)', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(6, 1, 2)
librosa.display.waveshow(y_gained, sr=sr, color='red', alpha=0.7)
plt.title('Waveform - Audio After Gain Control', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(6, 1, 3)
librosa.display.waveshow(audio_comp, sr=sr, color='purple', alpha=0.7)
plt.title('Waveform - Audio After Dynamic Range Compression', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

plt.subplot(6, 1, 4)
librosa.display.waveshow(audio_gated, sr=sr, color='orange', alpha=0.7)
plt.title('Waveform - Audio After Noise Gate', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)

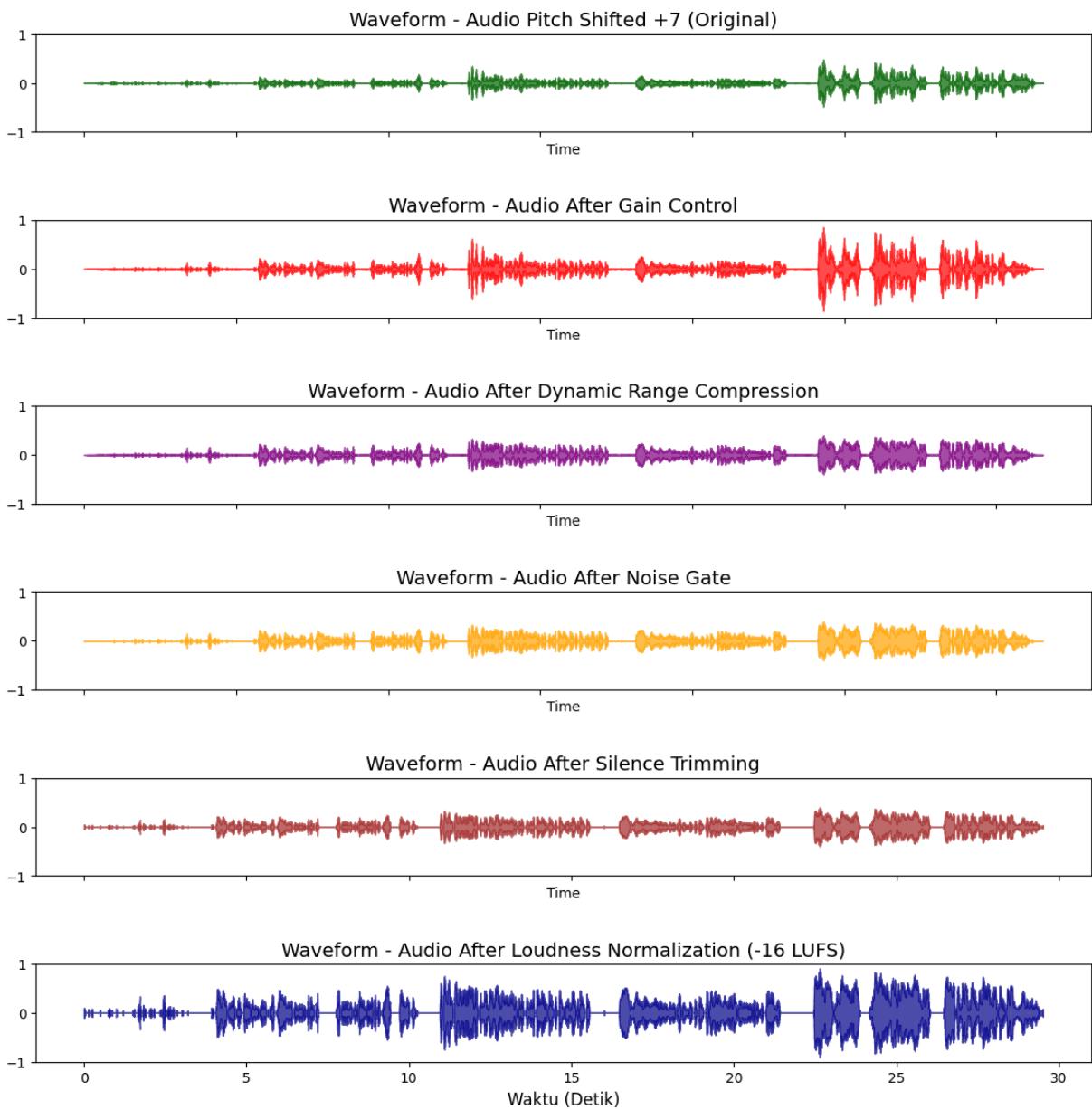
plt.subplot(6, 1, 5)
librosa.display.waveshow(trimmed, sr=sr, color='brown', alpha=0.7)
plt.title('Waveform - Audio After Silence Trimming', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.tick_params(labelbottom=False)
```

```

plt.subplot(6, 1, 6)
librosa.display.waveshow(loudness_normalized_audio, sr=sr, color='darkblue', alpha=1)
plt.title('Waveform - Audio After Loudness Normalization (-16 LUFS)', fontsize=14)
plt.ylim(-1.0, 1.0)
plt.xlabel("Waktu (Detik)", fontsize=12)

plt.tight_layout(pad=3.0)
plt.show()

```



```

In [ ]: # Side by Side Spectrogram Comparasion
N_fft = 2048
hop_length = 512
audio_data = [y1_pitch_shifted_7, trimmed, loudness_normalized_audio]
labels = ["Audio Pitch Shifted +7", "Audio Processed", "Audio Processed (-16 LUFS)"]
D1, D2, D3 = [np.abs(librosa.stft(y, n_fft=N_fft, hop_length=hop_length)) for y in S_db1, S_db2, S_db3 = [librosa.amplitude_to_db(D, ref=np.max) for D in [D1, D2, D3]

# v_min and v_max global

```

```
v_min = min(S_db1.min(), S_db2.min(), S_db3.min())
v_max = max(S_db1.max(), S_db2.max(), S_db3.max())

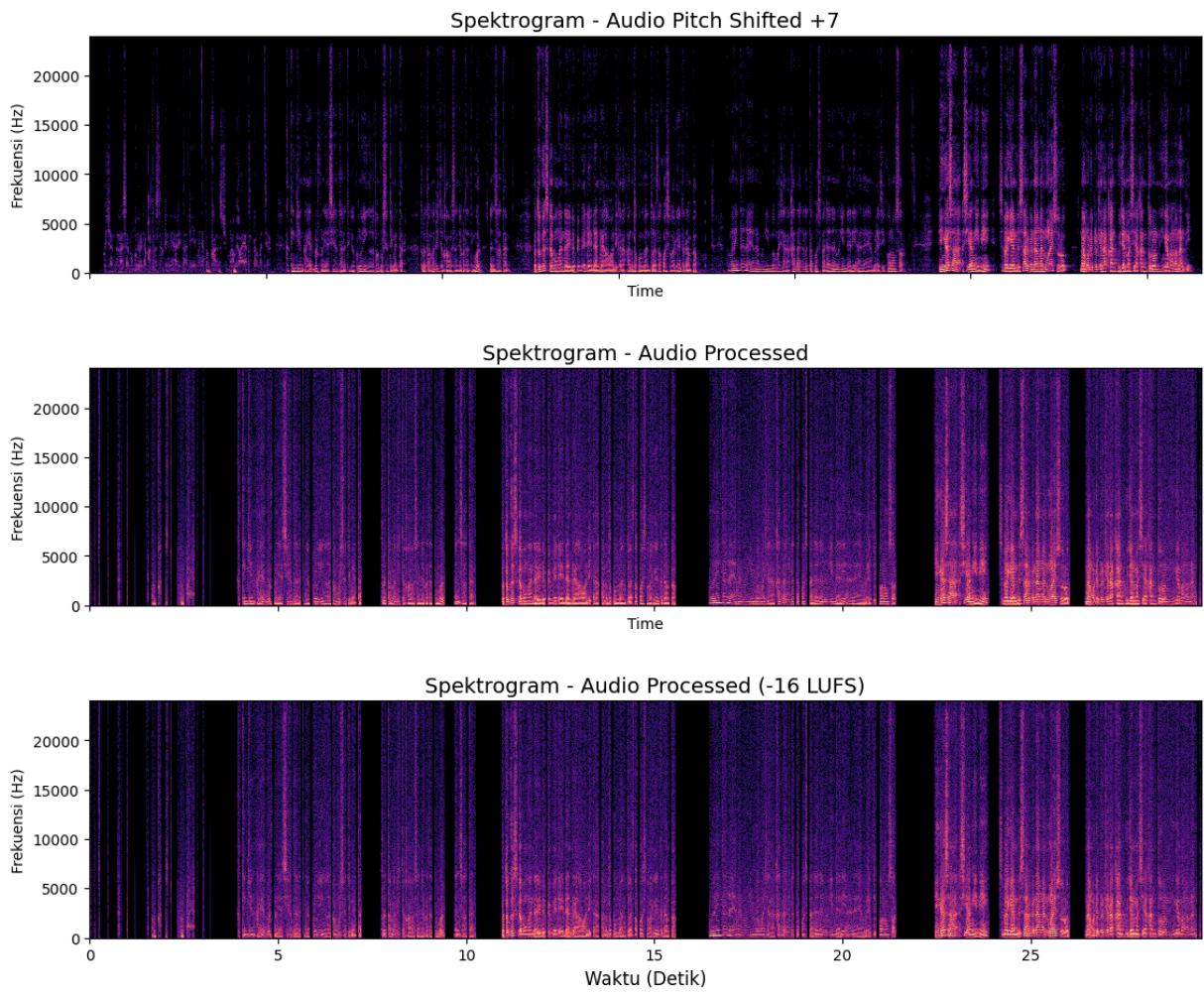
plt.figure(figsize=(12, 10))

# Spectrogram Plot
plt.subplot(3, 1, 1)
librosa.display.specshow(S_db1,
                        sr=sr,
                        x_axis='time',
                        y_axis='hz',
                        hop_length=hop_length,
                        cmap='magma', # Colormap
                        vmin=v_min,
                        vmax=v_max)
plt.title(f'Spektrogram - {labels[0]}', fontsize=14)
plt.ylabel('Frekuensi (Hz)')
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 2)
librosa.display.specshow(S_db2,
                        sr=sr,
                        x_axis='time',
                        y_axis='hz',
                        hop_length=hop_length,
                        cmap='magma',
                        vmin=v_min,
                        vmax=v_max)
plt.title(f'Spektrogram - {labels[1]}', fontsize=14)
plt.ylabel('Frekuensi (Hz)')
plt.tick_params(labelbottom=False)

plt.subplot(3, 1, 3)
img = librosa.display.specshow(S_db3,
                               sr=sr,
                               x_axis='time',
                               y_axis='hz',
                               hop_length=hop_length,
                               cmap='magma',
                               vmin=v_min,
                               vmax=v_max)
plt.title(f'Spektrogram - {labels[2]}', fontsize=14)
plt.ylabel('Frekuensi (Hz)')
plt.xlabel('Waktu (Detik)', fontsize=12)

plt.tight_layout(pad=3.0)
plt.show()
```



```
In [ ]: # Save Processed Audio
processed_audio_path = os.path.join(os.getcwd(), "media", "4_processed_audio.wav")
processed_normalized_audio_path = os.path.join(os.getcwd(), "media", "4_processed_normalized_audio.wav")

sf.write(processed_audio_path, trimmed, sr)
sf.write(processed_normalized_audio_path, loudness_normalized_audio, sr)
```

## Penjelasan Audio Processing Chain

1. Perubahan dinamika yang terjadi:

- **Audio Pitch Shifted +7 (Original).** Waveform ini menunjukkan hasil audio yang telah di-pitch shift naik tujuh semitone. Secara visual amplitudonya masih natural, dengan dinamika suara yang bervariasi sesuai intensitas bicara.
- **Setelah Gain Control.** Pada tahap gain control, amplitudo meningkat secara keseluruhan. Bagian yang sebelumnya pelan kini tampak lebih jelas dan seimbang, namun puncak amplitudo (peak) juga ikut naik sehingga rentang dinamik sedikit berkurang.
- **Setelah Dynamic Range Compression.** Proses compression menekan perbedaan antara bagian keras dan pelan. Terlihat bahwa bagian yang tadinya memiliki puncak

tinggi kini lebih rata. Hal ini meningkatkan kejelasan ucapan, terutama pada bagian lembut, namun mengurangi variasi dinamis alami.

- **Setelah Noise Gate.** Noise gate menghilangkan bagian dengan amplitudo rendah (noise latar belakang). Pada waveform terlihat beberapa segmen hening yang lebih bersih dibandingkan sebelumnya — jeda antar kata menjadi lebih jelas tanpa suara dengung.
- **Setelah Silence Trimming.** Tahapan ini memotong bagian diam di awal atau akhir rekaman. Waveform menjadi lebih padat karena durasi tanpa suara telah dihapus, sehingga fokus hanya pada bagian ucapan aktif.
- **Setelah Loudness Normalization (-16 LUFS).** Normalisasi loudness membuat keseluruhan volume audio konsisten dengan target -16 LUFS. Waveform tampak seragam dengan puncak dan dasar amplitudo yang seimbang. Hasil akhirnya memiliki tingkat kenyaringan yang stabil, cocok untuk distribusi standar audio digital.

2. Perbedaan Peak Normalization & LUFS Normalization:

- **Secara konsep,** Peak Normalization menyesuaikan amplitudo tertinggi dari sinyal untuk mencapai nilai target tertentu (0 atau -1 dBFS). Sedangkan LUFS Normalization menyesuaikan rata-rata persepsi keras suara (loudness) sesuai standar telinga manusia.
- **Secara teknis,** Peak Normalization dapat dilakukan dengan basis perhitungan menggunakan amplitudo maksimal dari suara. Sedangkan LUFS Normalization menggunakan fungsi perhitungan berbasis rata-rata persepsi energi dengan memanfaatkan library pyloudnorm.

3. Setelah proses normalisasi dan loudness optimization, kualitas suara menjadi lebih seimbang dan stabil. Volume antarbagian terdengar lebih seragam. Suara juga terasa lebih nyaman didengar karena sudah disesuaikan agar sesuai dengan tingkat kenyaringan standar (-16 LUFS). Namun, akibat penyesuaian ini, perbedaan antara bagian pelan dan keras sedikit berkurang, sehingga suara terdengar lebih rata tapi tetap jelas dan enak didengar.
4. Kelebihannya adalah pengoptimalan loudness membuat suara terdengar lebih konsisten dan profesional. Volume antarbagian jadi seimbang, sehingga pendengar tidak perlu sering menaikkan atau menurunkan volume. Sedangkan kekurangannya adalah membuat dinamika suara menjadi berkurang, suara yang lembut dan keras terdengar setara.

## Soal 5: Music Analysis and Remix

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

- Lagu 2: Nuansa ceria, cepat
2. Konversikan ke format WAV sebelum dimuat ke notebook.
  3. Lakukan deteksi tempo (BPM) dan estimasi kunci (key) dari masing-masing lagu dan berikan analisis singkat.
  4. 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)
  5. Jelaskan proses dan parameter yang digunakan.
  6. Tampilkan waveform dan spektrogram sesudah remix.
  7. Jelaskan hasil remix yang telah dilakukan.

```
In [ ]: # Check File Path
if not os.path.exists(PATH_3):
    raise FileNotFoundError(f"File not found: {PATH_3}")

if not os.path.exists(PATH_4):
    raise FileNotFoundError(f"File not found: {PATH_4}")

# Librosa Load
song_1, song1_sr = librosa.load(PATH_3, sr=None, mono=True)
song_2, song2_sr = librosa.load(PATH_4, sr=None, mono=True)

# BPM Detection
tempo_1, _ = librosa.beat.beat_track(y=song_1, sr=song1_sr)
tempo_2, _ = librosa.beat.beat_track(y=song_2, sr=song2_sr)

# Float
tempo_1 = float(np.squeeze(tempo_1))
tempo_2 = float(np.squeeze(tempo_2))

# Key Detection
def detect_key(signal, sr):
    chroma = librosa.feature.chroma_stft(y=signal, sr=sr)
    chroma_mean = np.mean(chroma, axis=1)
    notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
    return notes[np.argmax(chroma_mean)]

key_1 = detect_key(song_1, song1_sr)
key_2 = detect_key(song_2, song2_sr)

# Print the BPM and Key
print("Perbandingan Lagu")
print(f'Lagu 1: {os.path.basename(PATH_3)}')
print(f'Tempo = {tempo_1:.2f} BPM, Key = {key_1}')
print(f'Lagu 2: {os.path.basename(PATH_4)}')
print(f'Tempo = {tempo_2:.2f} BPM, Key = {key_2}'")
```

### Perbandingan Lagu

Lagu 1: Rex Orange County - Pluto Projector (Official Audio).wav  
 Tempo = 80.75 BPM, Key = C  
 Lagu 2: The Weeknd, Playboi Carti - Timeless (Official Lyric Video).wav  
 Tempo = 120.19 BPM, Key = F#

## Deskripsi Singkat Mengenai Tempo dan Kunci Lagu

1. Lagu pertama adalah lagu yang berjudul Pluto Projector dari Rex Orange County. Lagu ini memiliki tempo  $\approx$  80.75 BPM dan kunci di C, artinya lagu ini memiliki tempo yang lambat dan memiliki nuansa yang tenang dan meditatif.
2. Lagu ke-2 adalah lagu berjudul Timeless yang dibawakan oleh The Weeknd & Playboi Carti. Lagu ini memiliki tempo  $\approx$  120.19 BPM dan kunci di F#, artinya lagu ini memiliki tempo yang cepat dan juga enerjik.

```
In [ ]: # Extract Song Segment
def extract_segment(y, sr, start_min, start_sec, end_min, end_sec):
    start = int((start_min * 60 + start_sec) * sr)
    end = int((end_min * 60 + end_sec) * sr)
    return y[start:end]

seg1 = extract_segment(song_1, song1_sr, 3, 5, 3, 35)
seg2 = extract_segment(song_2, song2_sr, 0, 23, 0, 50)

# Match Tempo & Key
notes = ['C', 'C#', 'D', 'D#', 'E', 'F',
         'F#', 'G', 'G#', 'A', 'A#', 'B']
target_tempo = (tempo_1 + tempo_2) / 2
target_key_diff_1 = (notes.index(key_1) - notes.index(key_2)) / 2
target_key_diff_2 = -target_key_diff_1

# Adjust tempo of both songs
seg1_stretched = librosa.effects.time_stretch(seg1, rate=tempo_1 / target_tempo)
seg2_stretched = librosa.effects.time_stretch(seg2, rate=tempo_2 / target_tempo)

# Adjust key of both songs
seg1_shifted = librosa.effects.pitch_shift(y=seg1_stretched, sr=song1_sr, n_steps=-target_key_diff_1)
seg2_shifted = librosa.effects.pitch_shift(y=seg2_stretched, sr=song2_sr, n_steps=target_key_diff_2)

# Crossfade
crossfade_dur = 3
crossfade_samples = int(crossfade_dur * song1_sr)

fade_out = np.linspace(1, 0, crossfade_samples)
fade_in = np.linspace(0, 1, crossfade_samples)

seg1_shifted[-crossfade_samples:] *= fade_out
seg2_shifted[:crossfade_samples] *= fade_in

remix = np.concatenate([
    seg1_shifted[:-crossfade_samples],
    seg1_shifted[-crossfade_samples:] + seg2_shifted[:crossfade_samples],
    seg2_shifted[crossfade_samples:]]
```

])

Audio(remix, rate=song1\_sr)

Out[ ]:

▶ 0:00 / 0:56



## Proses dan Parameter Remix Lagu

- Pemotongan (Extract Segment).** Fungsi ini mengambil potongan lagu tertentu berdasarkan waktu mulai dan berakhir dalam menit-detik. Menggunakan parameter start dan end yang dikonversi ke satuan sample berdasarkan sampling rate (sr).
- Penyamaan Tempo dan Kunci (Tempo & Key Matching).** Menghitung rata-rata BPM dari kedua lagu untuk menyamakan BPM keduanya. Serta "menghitung" perbedaan kunci dari kedua lagu untuk menyamakan kunci keduanya. Menggunakan parameter tempo dan kunci (key) yang sudah dideteksi sebelumnya.
- Crossfade.** Menambahkan efek transisi halus selama 3 detik agar perpindahan lagu 1 ke lagu 2 terasa smooth. Menggunakan parameter durasi dari crossfade itu sendiri.
- Penggabungan (Concatenation).** Proses menggabungkan 3 bagian lagu menjadi 1 lagu remix.

In [ ]:

```
# Save Remixed Song
output_path = os.path.join(os.getcwd(), "media", "5_remixed_song.wav")
sf.write(output_path, remix, song1_sr)
print(f"Remix selesai dan disimpan ke: {output_path}")
```

Remix selesai dan disimpan ke: /content/media/5\_remixed\_song.wav

In [ ]:

```
# Waveform Plot of Remixed Song
duration = len(remix) / song1_sr
t = np.linspace(0, duration, len(remix))

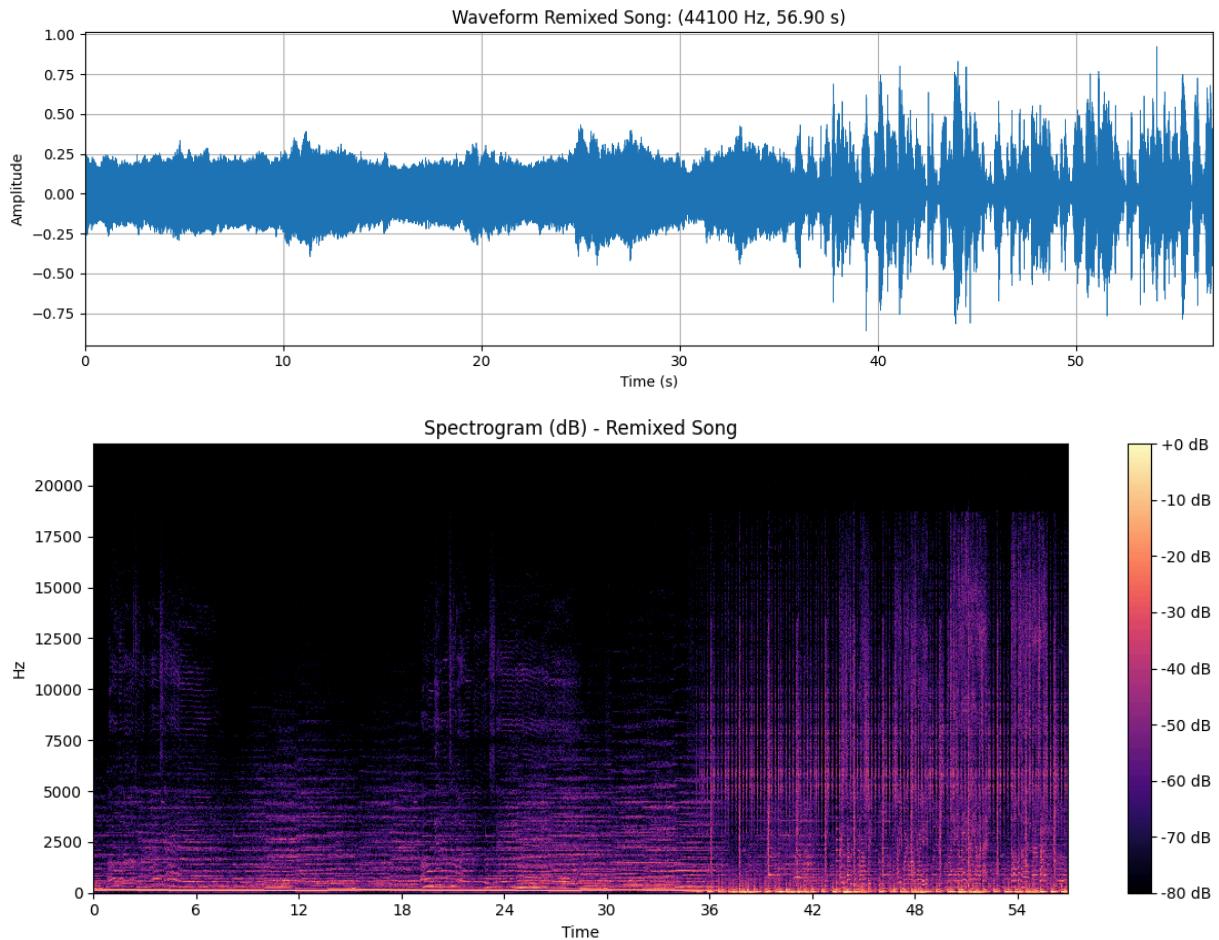
plt.figure(figsize=(12, 4))
plt.plot(t, remix, linewidth=0.5)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title(f"Waveform Remixed Song: ({song1_sr} Hz, {duration:.2f} s)")
plt.xlim(0, duration)
plt.grid(True)
plt.tight_layout()
plt.show()

# Spectrogram Plot of Remixed Song
n_fft = 2048
hop_length = 512

S = np.abs(librosa.stft(remix, n_fft=n_fft, hop_length=hop_length))
S_db = librosa.amplitude_to_db(S, ref=np.max)

plt.figure(figsize=(12, 5))
```

```
librosa.display.specshow(S_db, sr=song1_sr, hop_length=hop_length,
                        x_axis='time', y_axis='hz', cmap='magma')
plt.colorbar(format='%.2f dB')
plt.title(f"Spectrogram (dB) - Remixed Song")
plt.ylim(0, song1_sr/2)
plt.tight_layout()
plt.show()
```



## Penjelasan Hasil Remix

1. Penjelasan berdasarkan waveform:
  - Secara umum, amplitudo meningkat bertahap dari awal hingga akhir, menandakan adanya transisi dinamis antara dua lagu yang digabungkan.
  - Bagian awal terlihat lebih stabil dan rata, berasal dari lagu pertama (Pluto Projector) yang temponya lambat dan lembut.
  - Memasuki pertengahan hingga akhir, amplitudo meningkat dan lebih padat, menggambarkan perpindahan ke lagu kedua (Timeless) yang memiliki karakter cepat dan energik.
  - Transisi antarbagian tampak halus tanpa lonjakan ekstrem, menunjukkan bahwa proses crossfade berhasil menjaga kontinuitas audio.
2. Penjelasan berdasarkan spektrogram:

- Pada awal rekaman, energi lebih dominan di frekuensi menengah dan rendah (0-5000 Hz), sesuai karakter lagu bernuansa mellow.
- Di pertengahan hingga akhir, muncul frekuensi tinggi dengan intensitas lebih kuat. Hal ini mencerminkan bagian lagu kedua yang lebih ceria.
- Pergantian warna menunjukkan peningkatan energi dan kecerahan suara yang konsisten dengan perubahan tempo dan pitch hasil remix.
- Tidak tampak distorsi atau pemotongan energi tiba-tiba, menandakan bahwa proses time-stretch dan pitch-shift berjalan dengan baik.

### 3. Kesimpulan secara umum:

- Lagu yang dipilih adalah Pluto Projector dan Timeless. Alasan dipilihnya kedua lagu ini karena memiliki outro dan intro yang menarik untuk digabung dan dilakukan crossfading.
- Hasil remix dilakukan dengan 3 proses utama, yakni Time-Streching, Pitch Shift, dan juga Crossfading.
- Proses penyeragaman tempo dan pitch diterapkan pada kedua lagu. Langkah ini membuat karakteristik kedua lagu menjadi lebih seimbang, yang merupakan kunci untuk menciptakan crossfade yang mulus.

## Disclaimer Penggunaan LLM

Pengerjaan tugas ini melibatkan penggunaan AI LLM ChatGPT yang berperan dalam menghasilkan dan merapikan kode, serta mendukung proses analisis. Berikut ini adalah link bukti penggunaan LLM.

[Referensi LLM](#)