Nirvana Smells Like Teen Spirit - 1993

In [ ]:

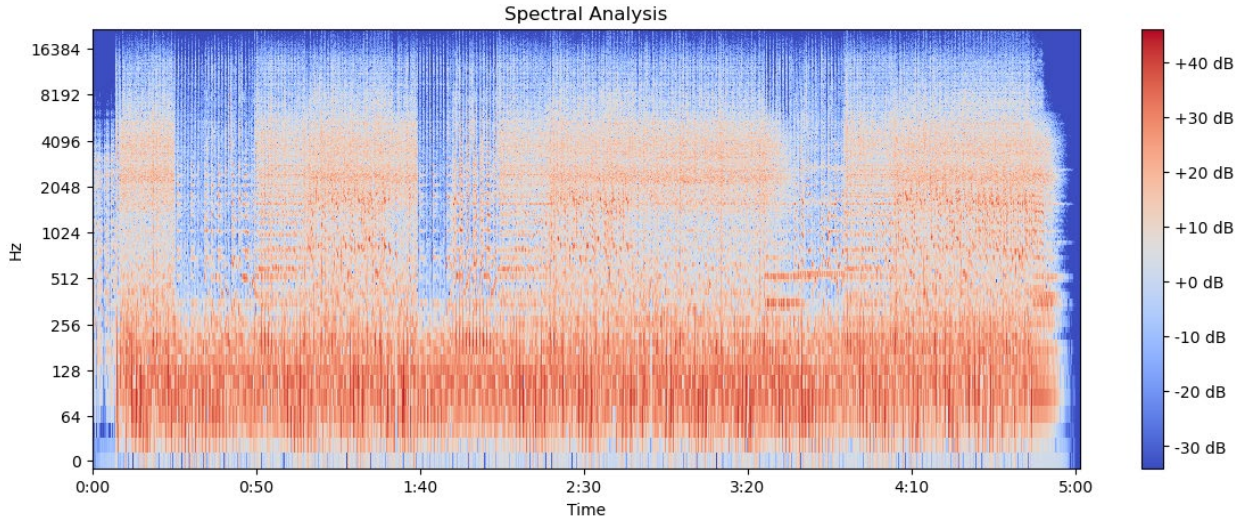
```
import librosa
import ffmpeg
import librosa.display
import matplotlib.pyplot as plt
import soundfile
import audioread
```

In [ ]:

```
from librosa import display
import librosa.display as display
from librosa.display import waveshow
```

In [ ]:

```
# Load the audio file
file_path = '01 Smells Like Teen Spirit.wav'
audio, sample_rate = librosa.load(file_path, sr=None)

# Perform a Short-Time Fourier Transform (STFT) to get the frequency content over time
stft = librosa.stft(audio)

# Convert the STFT to decibels, which is a more useful measure for human hearing
db_stft = librosa.amplitude_to_db(abs(stft))

# Plot the spectrogram
plt.figure(figsize=(14, 5))
librosa.display.specshow(db_stft, sr=sample_rate, x_axis='time', y_axis='log', cmap='coolwarm')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectral Analysis')
plt.show()
```



For "Smells Like Teen Spirit," we can observe several characteristics of the song's frequency content over time:

Low Frequencies: There is significant energy below 256 Hz, indicating strong bass and kick drum presence. This area likely needs to be tight and punchy, without too much boominess.

Mid Frequencies: The range from 256 Hz to around 2 kHz seems to have varying energy levels, which can correspond to the body of the guitars, the snare drum, and the lower register of the vocals. The mids are crucial for the clarity and presence of the track.

High Mid Frequencies: Between approximately 2 kHz and 4 kHz, we see some peaks which might indicate the presence of vocal sibilance, the attack of the snare drum, and guitar string noise. These frequencies can add definition but also harshness if not balanced well.

High Frequencies: Above 4 kHz, there is less energy, but it's still essential for the sense of "air" and brightness in the track. Cymbals, the high end of the guitars, and certain vocal harmonics will live in this range

Using this information, you can make informed decisions on how to shape the EQ curve for your mix. Remember, the goal of EQ in mixing is to ensure that each element has its own space in the frequency spectrum, and that the overall mix sounds balanced and clear. If you're mixing this on an Allen & Heath dLive s7000 console, you would typically apply EQ adjustments in a more detailed way, considering each instrument's and vocal's role in the mix.

Frequency Range Identification:

Software: Digital Audio Workstations (DAWs) like Pro Tools, Ableton Live, Logic Pro, and others have built-in spectrum analyzers and EQ plugins that can help identify frequency ranges of instruments and vocals. Python Libraries: librosa is a Python library that can perform spectral analysis, and numpy can be used to analyze frequency data. Dynamics Analysis:

Software: You can use DAWs with dynamics plugins or dedicated analysis tools like iZotope Insight or Waves PAZ Analyzer for visualizing and analyzing dynamics. Python Libraries: pydub can help analyze loudness and dynamics to some extent, although it's less sophisticated compared to professional audio software. Transient Analysis:

Software: Specialized transient analysis plugins, like those from FabFilter or SPL, can visualize and alter transients. DAWs also typically have transient detection and editing capabilities. Python Libraries: librosa provides some functionality for onset detection, which is a form of transient analysis. Harmonic Analysis:

Software: DAWs with pitch-correction plugins like Celemony's Melodyne or Antares Auto-Tune can help visualize and analyze harmonics. Python Libraries: librosa can also perform harmonic analysis by separating harmonic elements from percussive elements in an audio signal.

Spectral Analysis: This provides a visual representation of the different frequencies present in the audio track over time. It helps to identify which frequencies are most prominent or may need adjustment.

Frequency Range Identification: Determining the frequency ranges of various instruments and vocals within the track to understand where they sit in the mix.

Dynamics Analysis: Understanding the dynamic range of the audio can help in making decisions about compression before EQ adjustments.

Transient Analysis: This involves looking at the attack and decay of various sounds, which can influence EQ decisions, especially in terms of clarity and separation of instruments.

Harmonic Analysis: Identifying the harmonic content of the track can assist in making EQ choices that affect the warmth, brightness, or presence of the audio.

In [ ]:

```
y, sr = librosa.load('01 Smells Like Teen Spirit.wav')
```

In [ ]:

```
 # Calculate the spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
```

```
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]
```

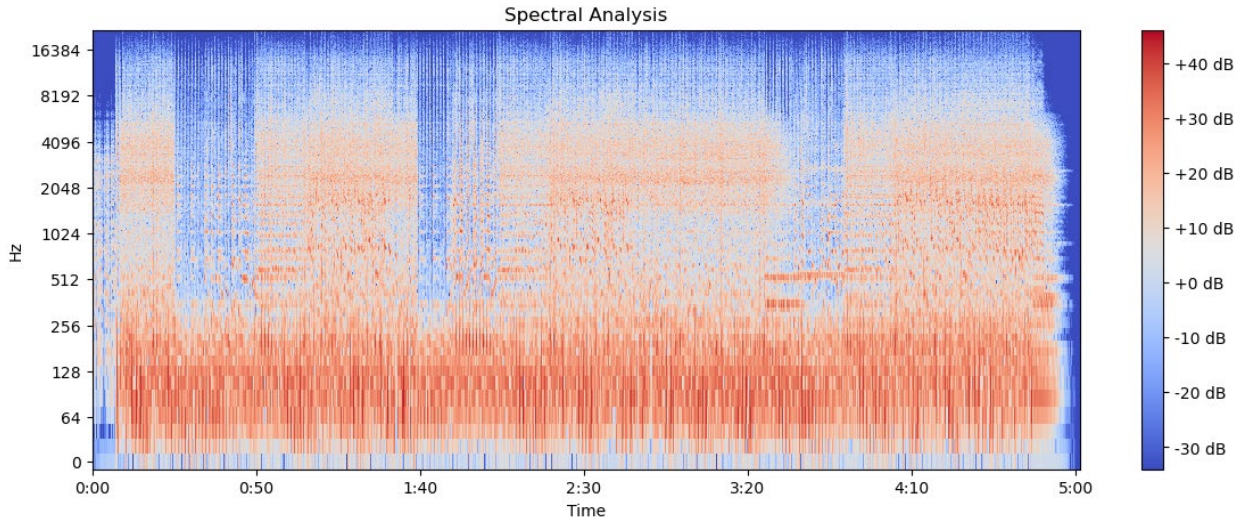
In [ ]:

```
# Convert frame counts to time
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames, sr=sr)
```

In [ ]:

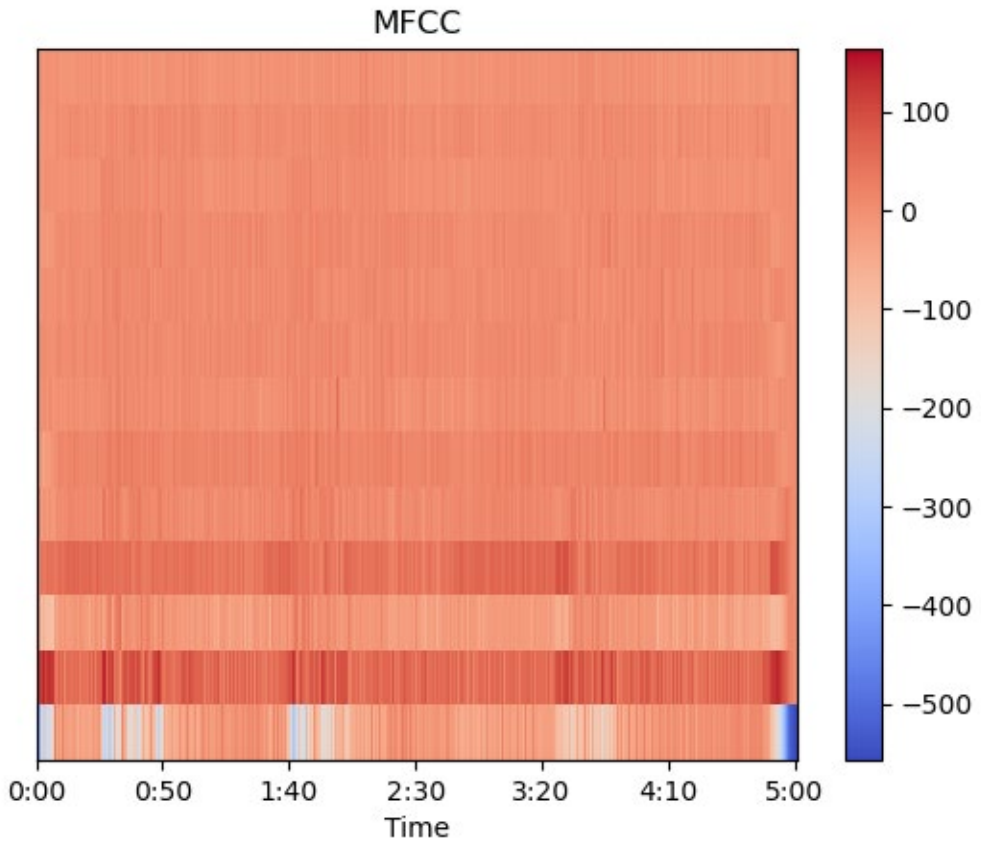
```
# Load the audio file
file_path = '01 Smells Like Teen Spirit.wav'
audio, sample_rate = librosa.load(file_path, sr=None)

# Perform a Short-Time Fourier Transform (STFT) to get the frequency content over time
stft = librosa.stft(audio)

# Convert the STFT to decibels, which is a more useful measure for human hearing
db_stft = librosa.amplitude_to_db(abs(stft))

# Plot the spectrogram
plt.figure(figsize=(14, 5))
librosa.display.specshow(db_stft, sr=sample_rate, x_axis='time', y_axis='log', cmap='coolwarm')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectral Analysis')
plt.show()
```

```
 # Calculate the spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]

# Convert frame counts to time
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames, sr=sr)
```



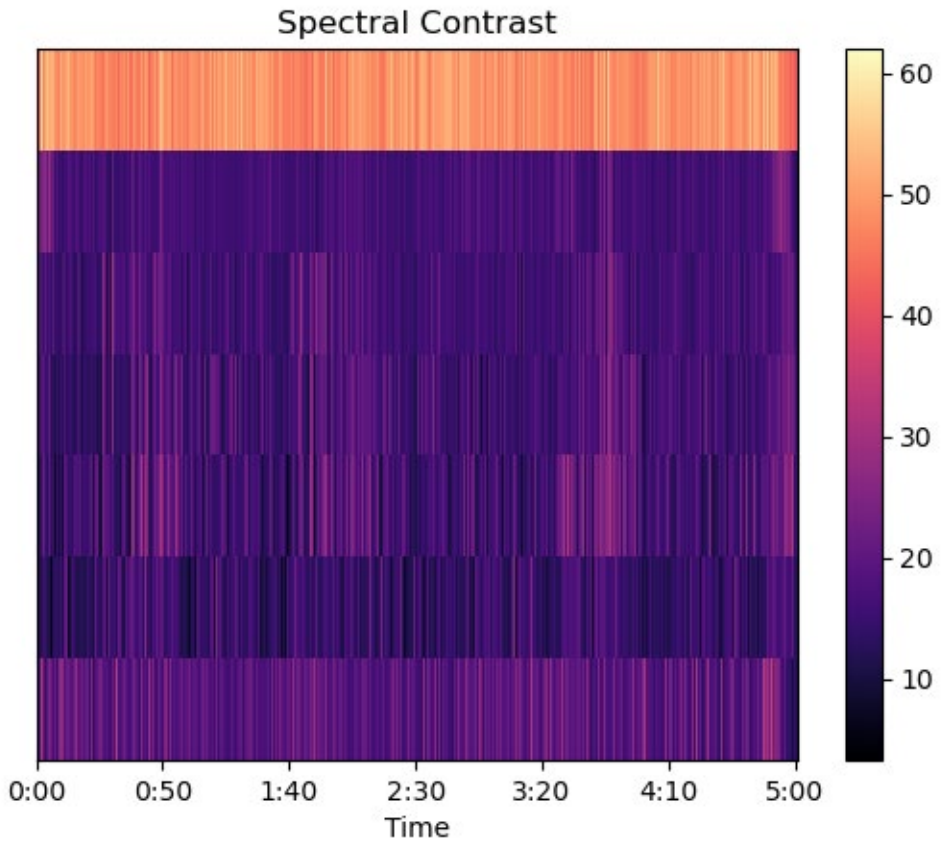
In [ ]:

```
# Mel-Frequency Cepstral Coefficients (MCFFs):
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('MFCC')
plt.show()
```



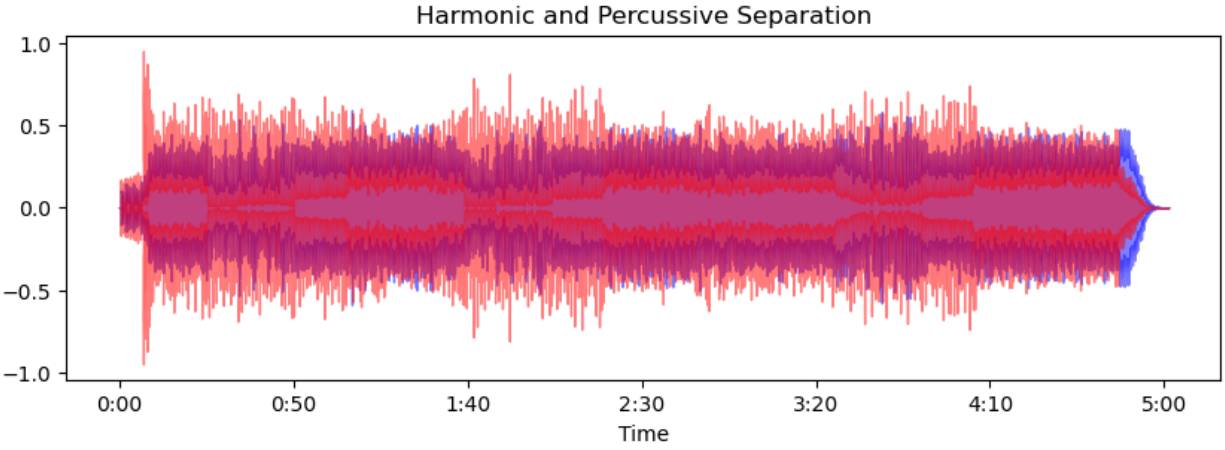
In [ ]:

```
# Spectral Contrast:
spec_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
librosa.display.specshow(spec_contrast, x_axis='time')
plt.colorbar()
plt.title('Spectral Contrast')
plt.show()
```



In [ ]:

```
# Harmonic & Percussove Separation:
y_harm, y_perc = librosa.effects.hpss(y)
plt.figure(figsize=(10, 3))
librosa.display.waveshow(y_harm, sr=sr, alpha=0.5, color='b')
librosa.display.waveshow(y_perc, sr=sr, alpha=0.5, color='r')
plt.title('Harmonic and Percussive Separation')
plt.show()
```



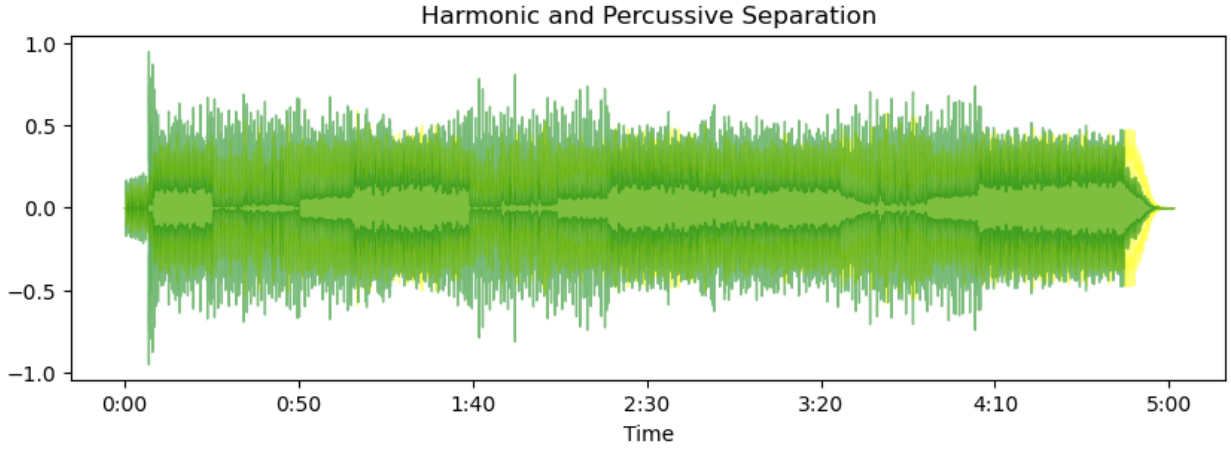
In [ ]:

```
# # Load an audio file
# y, sr = librosa.load('01 Smells Like Teen Spirit.wav')
```

```
# # Perform the harmonic-percussive separation
# y_harm, y_perc = librosa.effects.hpss(y)# Plotting
# plt.figure(figsize=(10, 5))
# plt.subplot(2, 1, 1)
# librosa.display.waveshow(y_harmonic, sr=sr, color='b', alpha=0.5)
# plt.title('Harmonic Part')

# plt.subplot(2, 1, 2)
# librosa.display.waveshow(y_percussive, sr=sr, color='r', alpha=0.5)
# plt.title('Percussive Part')
# plt.tight_layout()
# plt.show()
```

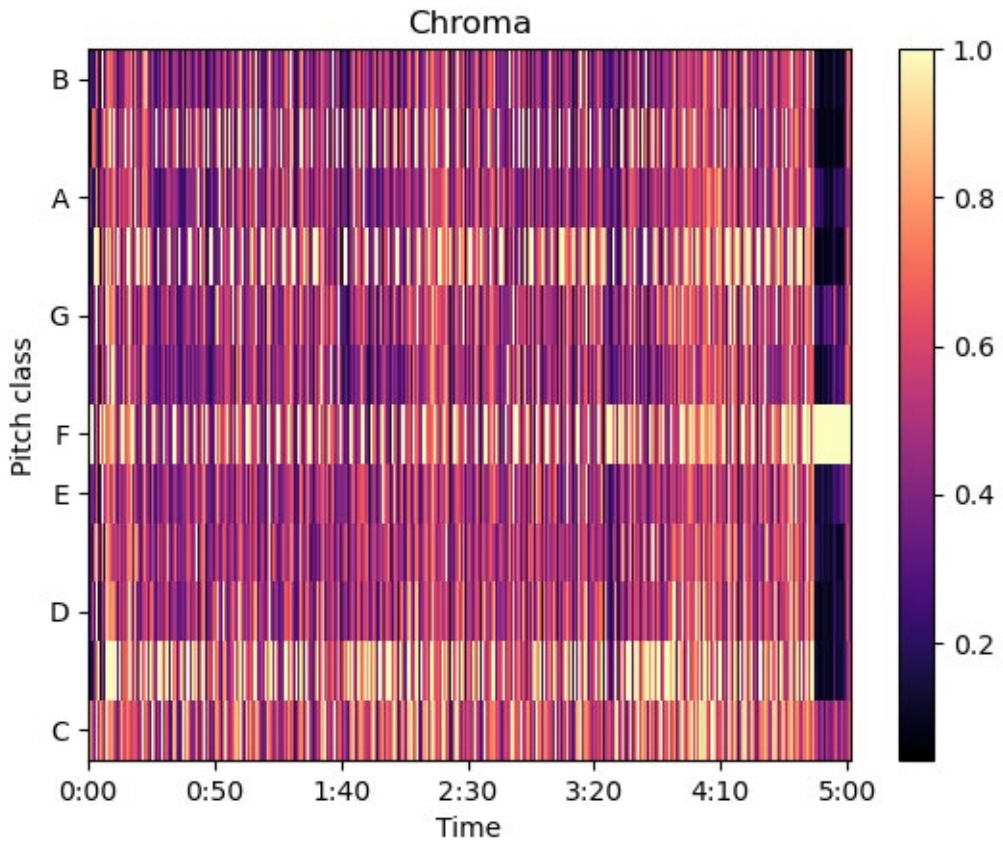
In [ ]:

```
y_harm, y_perc = librosa.effects.hpss(y)
plt.figure(figsize=(10, 3))
librosa.display.waveshow(y_harm, sr=sr, alpha=0.5, color='yellow')
librosa.display.waveshow(y_perc, sr=sr, alpha=0.5, color='green')
plt.title('Harmonic and Percussive Separation')
plt.show()
```



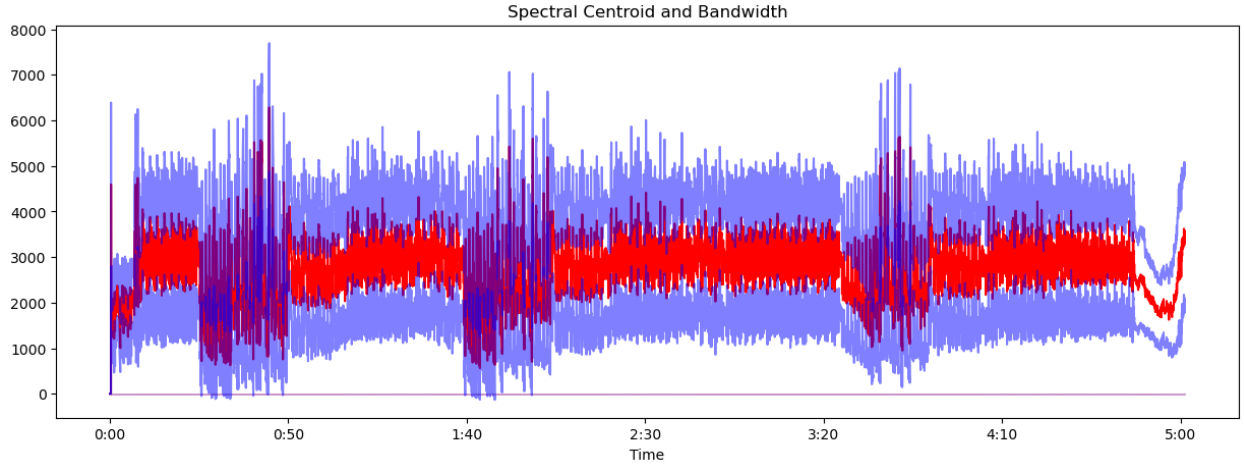
In [ ]:

```
# Chroma Feature Extraction:
chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
librosa.display.specshow(chroma, y_axis='chroma', x_axis='time')
plt.colorbar()
plt.title('Chroma')
plt.show()
```



In [ ]:

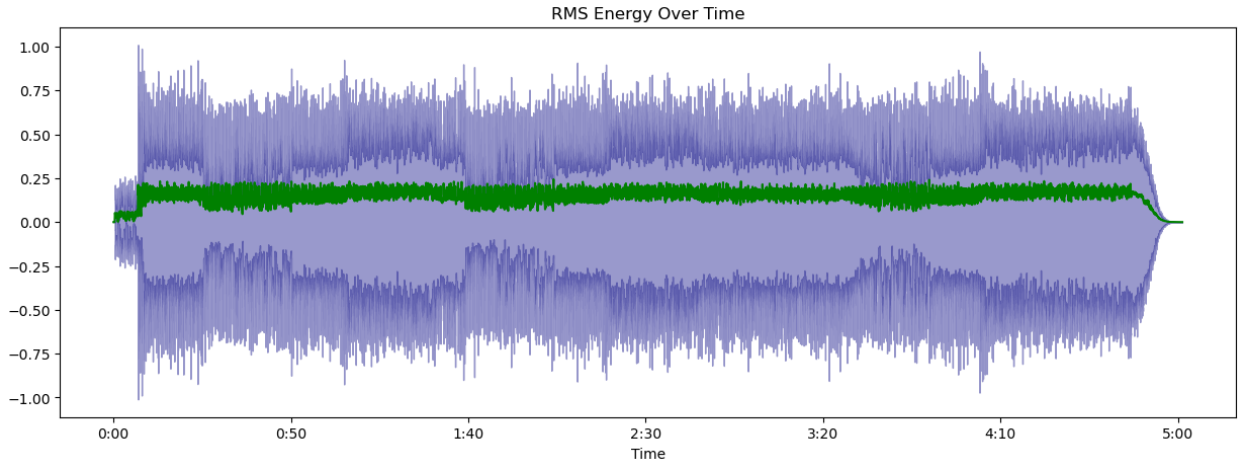
```
# Plotting the Spectral Centroid along the waveform
plt.figure(figsize=(15, 5))
librosa.display.waveshow(y, sr=sr, alpha=0.4, color="purple")
plt.plot(t, spectral_centroids, color='r') # Spectral centroid
plt.plot(t, spectral_centroids - spectral_bandwidth / 2, color='b', alpha=0.5) # Min range
plt.plot(t, spectral_centroids + spectral_bandwidth / 2, color='b', alpha=0.5) # Max range
plt.title('Spectral Centroid and Bandwidth')
plt.show()
```



In [ ]:

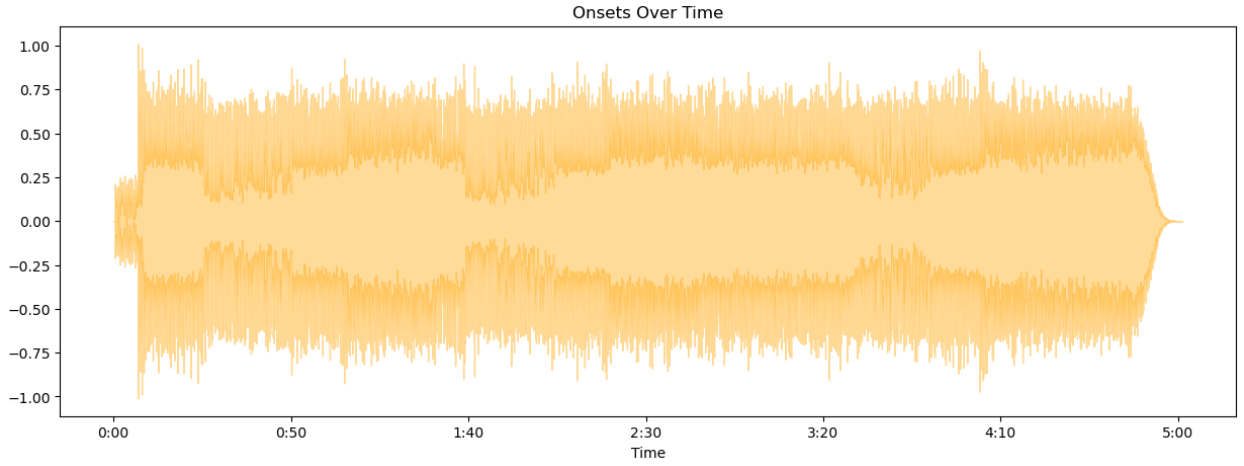
```
# Calculate the RMS energy
rms_energy = librosa.feature.rms(y=y)[0]
```

```
# Plotting the RMS along the waveform
plt.figure(figsize=(15, 5))
librosa.display.waveshow(y, sr=sr, alpha=0.4, color="navy")
plt.plot(t, rms_energy, color='g') # RMS Energy
plt.title('RMS Energy Over Time')
plt.show()
```



In [ ]:

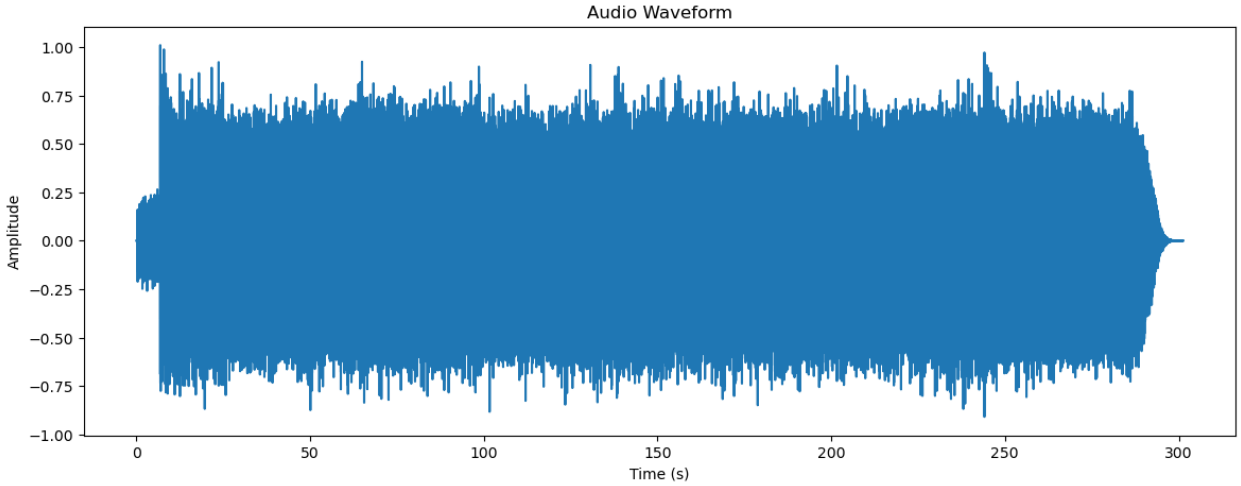
```
# Detect onsets
onset_frames = librosa.onset.onset_detect(y=y, sr=sr)
onset_times = librosa.frames_to_time(onset_frames, sr=sr)

# Plotting the onsets
plt.figure(figsize=(15, 5))
# librosa.display.waveshow(y, sr=sr, alpha=0.4)
# plt.vlines(onset_times, ymin=-1, ymax=1, color='r')
librosa.display.waveshow(y, sr=sr, alpha=0.4, color="orange")
plt.title('Onsets Over Time')
plt.show()
```



In [ ]:

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

# Load your audio file
```

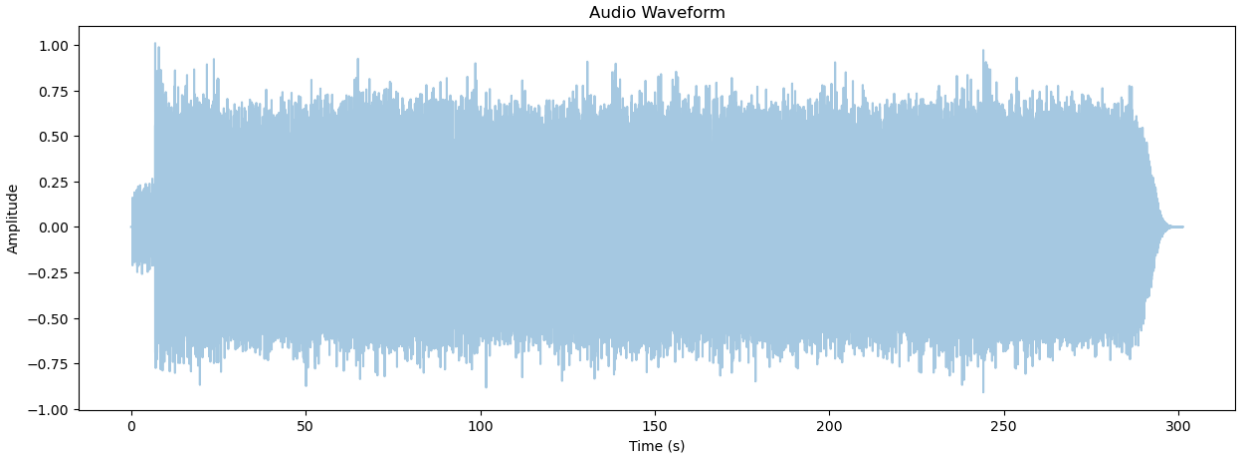
```
y, sr = librosa.load('01 Smells Like Teen Spirit.wav')

# Generate time axis data
time = np.linspace(0, len(y) / sr, num=len(y))

# Plot the waveform
plt.figure(figsize=(14, 5))
plt.plot(time, y)
plt.title('Audio Waveform')
plt.ylabel('Amplitude')
plt.xlabel('Time (s)')
plt.show()
```

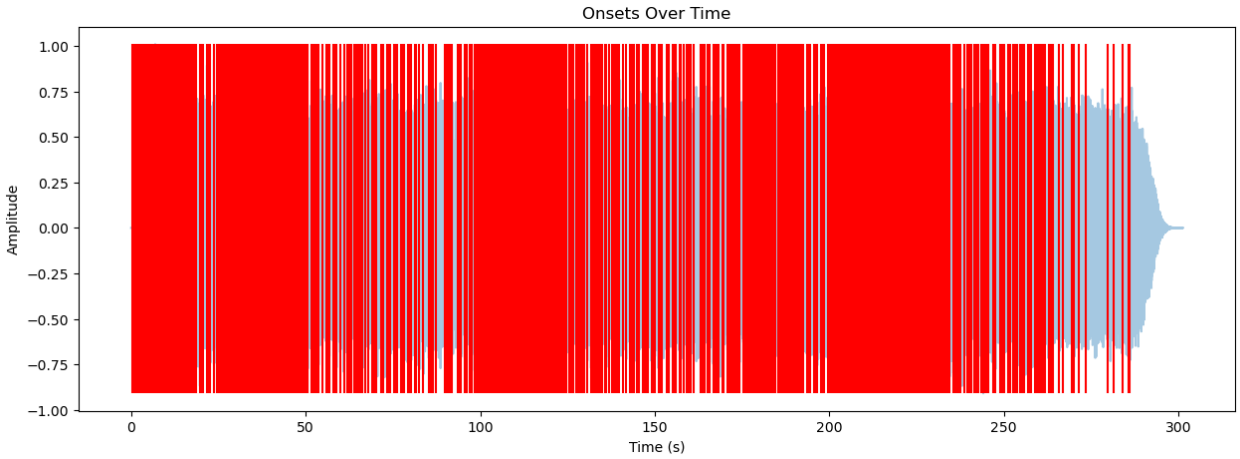


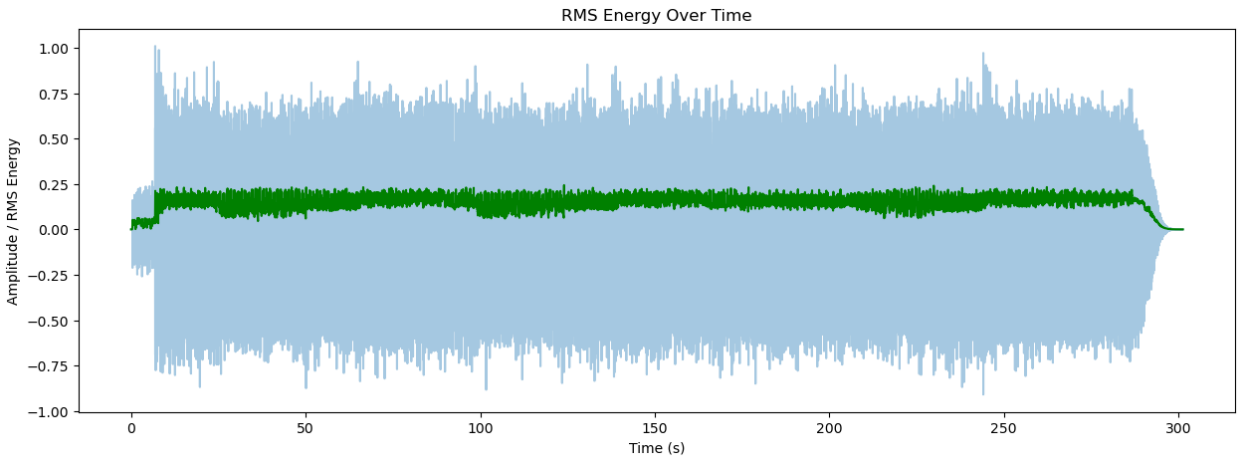
In [ ]:

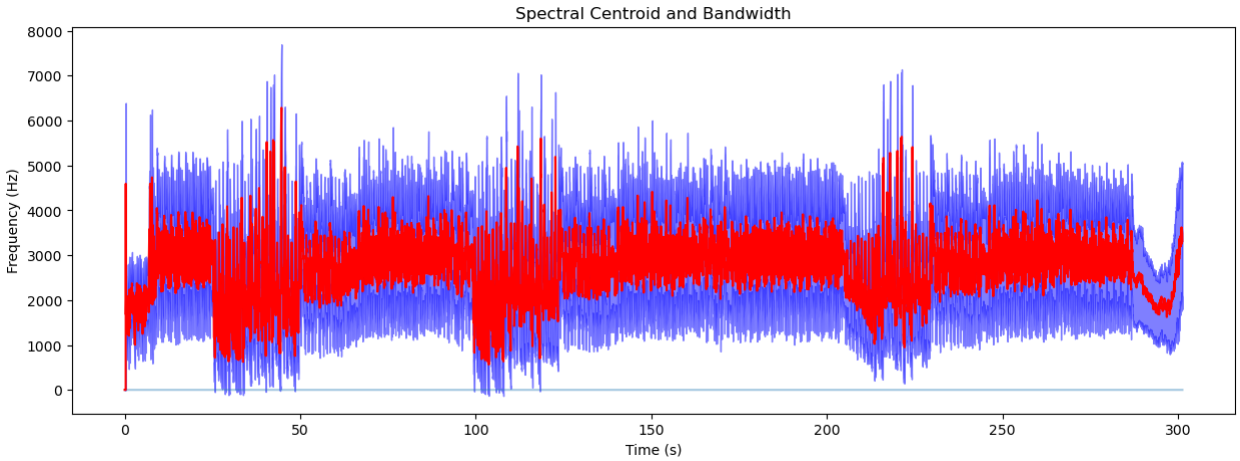
```
import librosa
import numpy as np
import matplotlib.pyplot as plt

# Load the audio file
y, sr = librosa.load('01 Smells Like Teen Spirit.wav')

# Time axis for the audio file
t = np.linspace(0, len(y) / sr, num=len(y))

# Plot the audio waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.title('Audio Waveform')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

# Detect onsets
onset_frames = librosa.onset.onset_detect(y=y, sr=sr)
onset_times = librosa.frames_to_time(onset_frames, sr=sr)

# Plotting the onsets over the waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.vlines(onset_times, ymin=min(y), ymax=max(y), color='r')
```

```
plt.title('Onsets Over Time')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

# Calculate the RMS energy
rms_energy = librosa.feature.rms(y=y)[0]
rms_times = librosa.frames_to_time(range(len(rms_energy)), sr=sr)

# Plotting the RMS energy over time
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.plot(rms_times, rms_energy, color='g')
plt.title('RMS Energy Over Time')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / RMS Energy')
plt.show()

# Calculate the spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]
centroid_times = librosa.frames_to_time(range(len(spectral_centroids)), sr=sr)

# Plotting the Spectral Centroid and Bandwidth over the waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.plot(centroid_times, spectral_centroids, color='r')
plt.fill_between(centroid_times, spectral_centroids - spectral_bandwidth / 2, spectral_centroids +
spectral_bandwidth / 2, color='b', alpha=0.5)
plt.title('Spectral Centroid and Bandwidth')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')
plt.show()
```





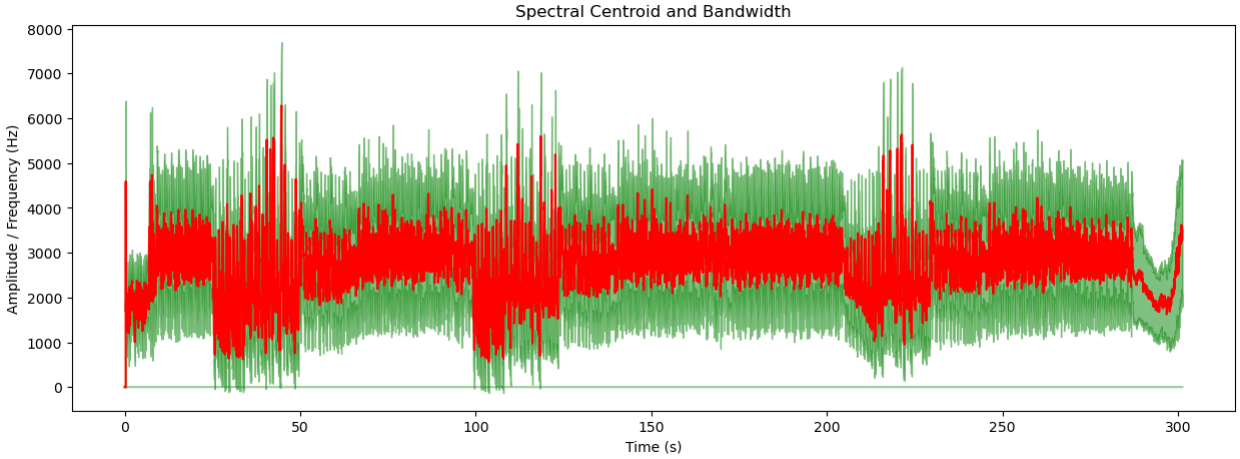




In [ ]:

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

# Load your audio file
y, sr = librosa.load('01 Smells Like Teen Spirit.wav')

# Generate time axis data
t = np.linspace(0, len(y) / sr, num=len(y))
```

```
# Calculate spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]
frames = range(len(spectral_centroids))
t_centroid = librosa.frames_to_time(frames, sr=sr)  # Time for centroid plots

# Plot the audio waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4, color="green")  # Waveform in blue
plt.plot(t_centroid, spectral_centroids, color='r')  # Spectral centroid in red
plt.fill_between(t_centroid, spectral_centroids - spectral_bandwidth / 2, spectral_centroids +
spectral_bandwidth / 2, color='g', alpha=0.5)  # Bandwidth range in blue
plt.title('Spectral Centroid and Bandwidth')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / Frequency (Hz)')
plt.show()
```



In [ ]:

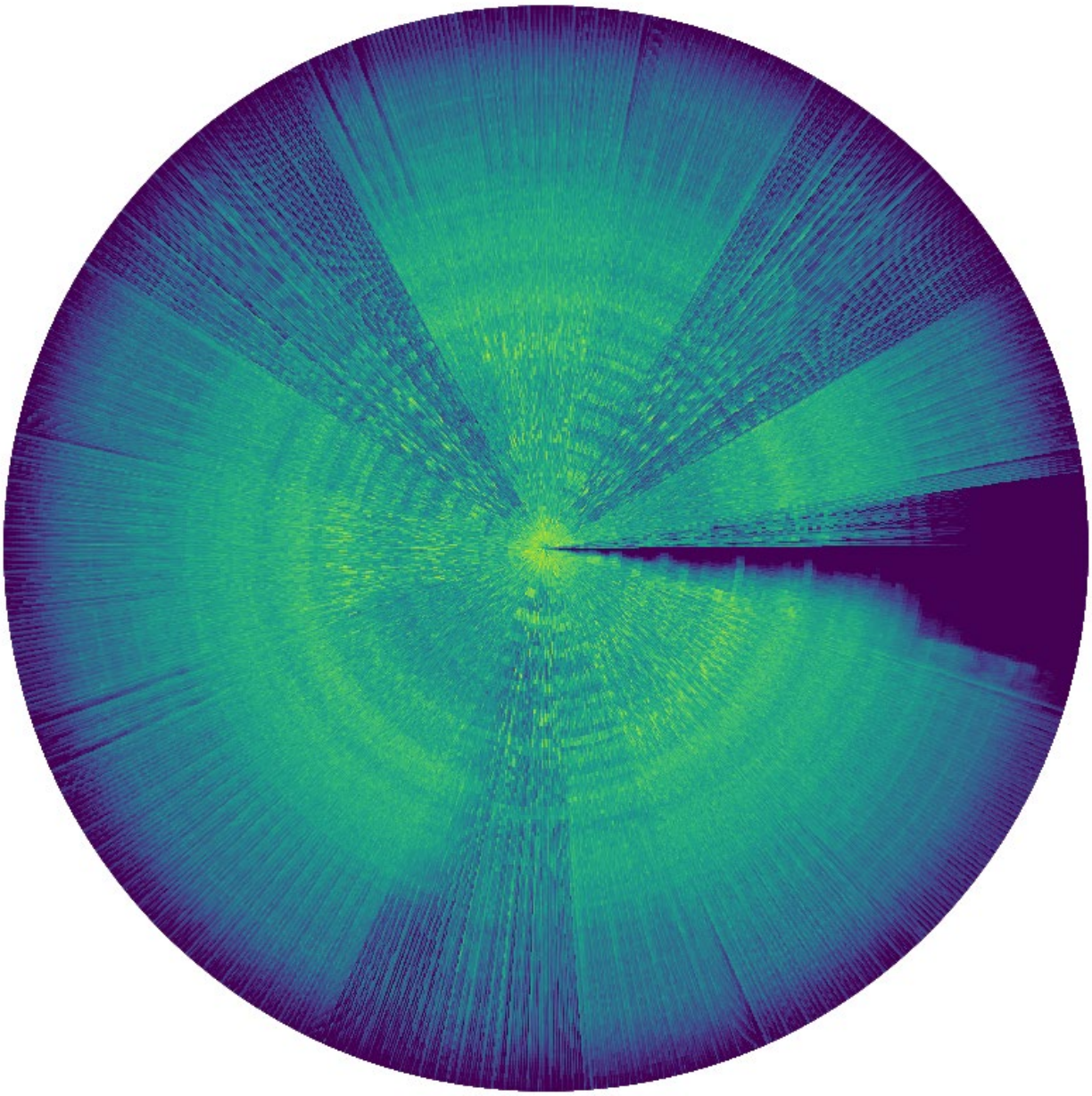
```
zero_crossings = librosa.zero_crossings(y, pad=False)
print("Total Zero Crossings:", sum(zero_crossings))
```

Total Zero Crossings: 1029764

In [ ]:

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

# Load the audio file
y, sr = librosa.load('01 Smells Like Teen Spirit.wav', sr=None)

# Extract features
spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)

# Convert to decibels
spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Create the meshgrid for the spectrogram plot
```

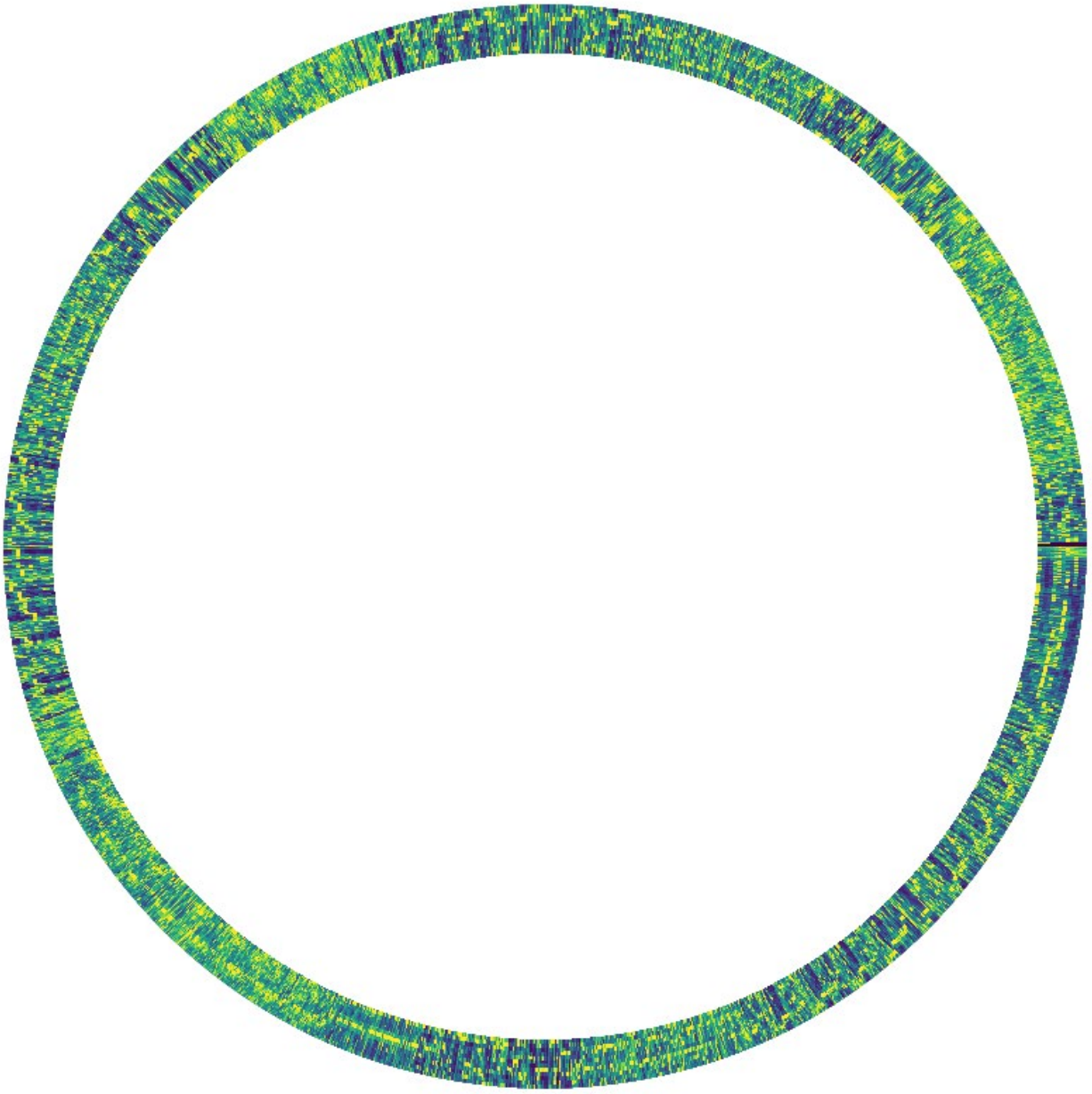
```
# Ensure theta and r have one more point than the spectrogram_db dimensions
theta = np.linspace(0, 2*np.pi, spectrogram_db.shape[1] + 1)
r = np.linspace(0, 1, spectrogram_db.shape[0] + 1)
theta, r = np.meshgrid(theta, r)

# Plot the spectrogram
ax.pcolormesh(theta, r, spectrogram_db, shading='flat')

plt.show()
```



In [ ]:

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

# Load the audio file
```

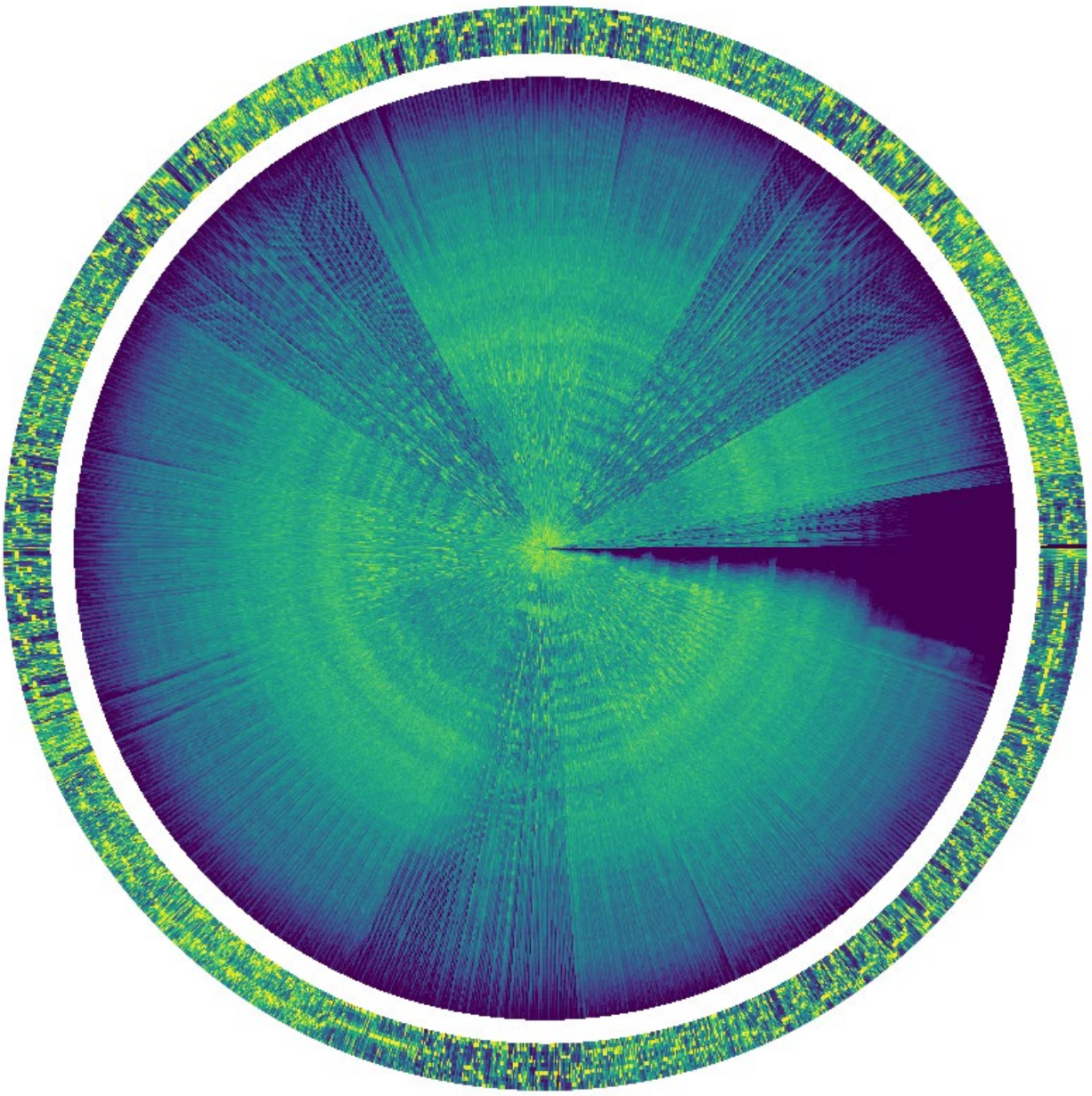
```
y, sr = librosa.load('01 Smells Like Teen Spirit.wav', sr=None)

# Extract features
chromagram = librosa.feature.chroma_stft(y=y, sr=sr)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Create the meshgrid for the chromagram plot
# Ensure theta and r have one more point than the chromagram dimensions
chroma_theta = np.linspace(0, 2*np.pi, chromagram.shape[1] + 1)
chroma_r = np.linspace(1, 1.1, chromagram.shape[0] + 1)
chroma_theta, chroma_r = np.meshgrid(chroma_theta, chroma_r)

# Plot the chromagram
ax.pcolormesh(chroma_theta, chroma_r, chromagram, shading='flat')

plt.show()
```



In [ ]:

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

# Load the audio file
y, sr = librosa.load('01 Smells Like Teen Spirit.wav', sr=None)

# Extract features
spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr)

# Convert spectrogram to decibels
spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)
```

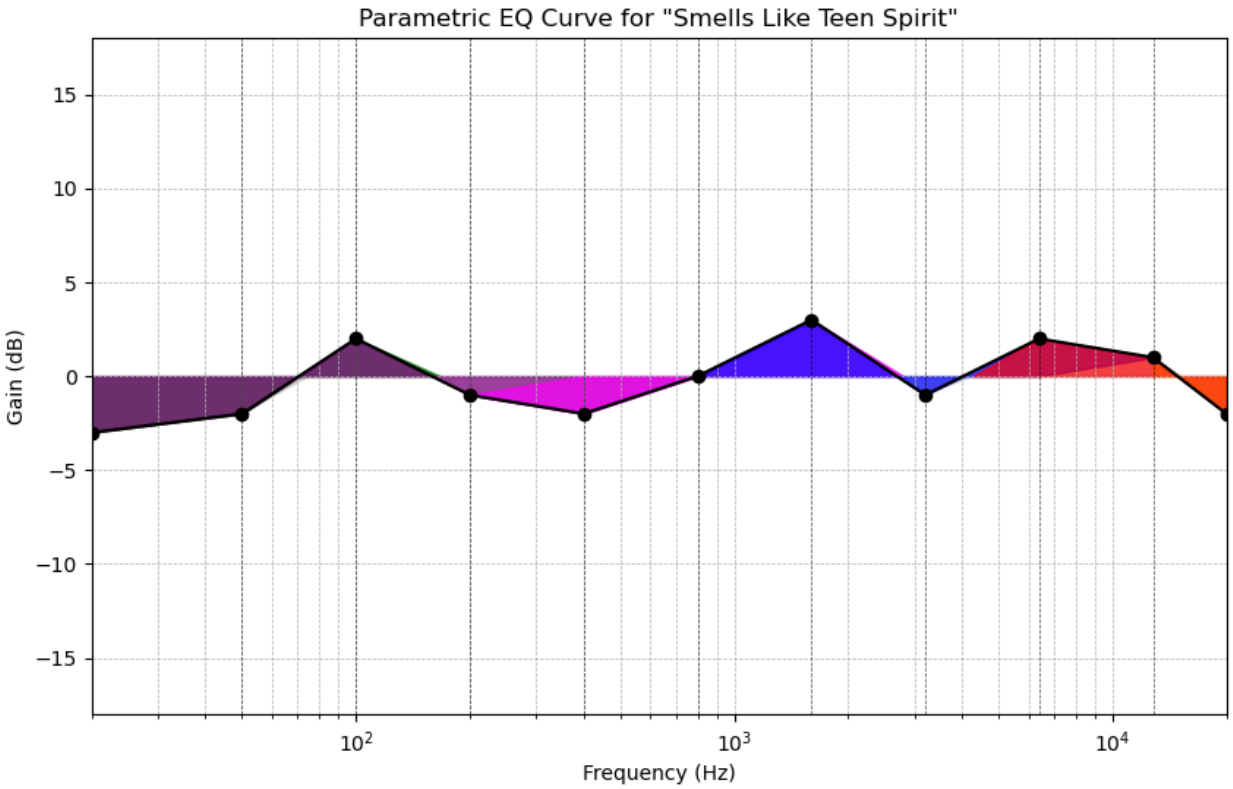
```
# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Spectrogram grid
theta = np.linspace(0, 2 * np.pi, spectrogram_db.shape[1] + 1)
r = np.linspace(0, 1, spectrogram_db.shape[0] + 1)
theta_grid, r_grid = np.meshgrid(theta, r)

# Chromagram grid
chroma_theta = np.linspace(0, 2 * np.pi, chromagram.shape[1] + 1)
chroma_r = np.linspace(1.05, 1.15, chromagram.shape[0] + 1)  # Adjusted to sit just outside the spectrogram
chroma_theta_grid, chroma_r_grid = np.meshgrid(chroma_theta, chroma_r)

# Plot the spectrogram
ax.pcolormesh(theta_grid, r_grid, spectrogram_db, shading='flat')

# Plot the chromagram
ax.pcolormesh(chroma_theta_grid, chroma_r_grid, chromagram, shading='flat', cmap='viridis')  # Optionally set a different colormap

plt.show()
```



In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# Since we don't have the actual audio data to analyze and create an EQ curve for "Smells Like Teen Spirit",
# we will create a mock-up EQ curve similar to the previously provided example for "Purple Rain".

# Frequency bands in Hz (mock values)
frequencies = np.array([20, 50, 100, 200, 400, 800, 1600, 3200, 6400, 12800, 20000])
# Gain values for each band in dB (mock values)
gains = np.array([-3, -2, 2, -1, -2, 0, 3, -1, 2, 1, -2])

# Create the plot
plt.figure(figsize=(10, 6))
```

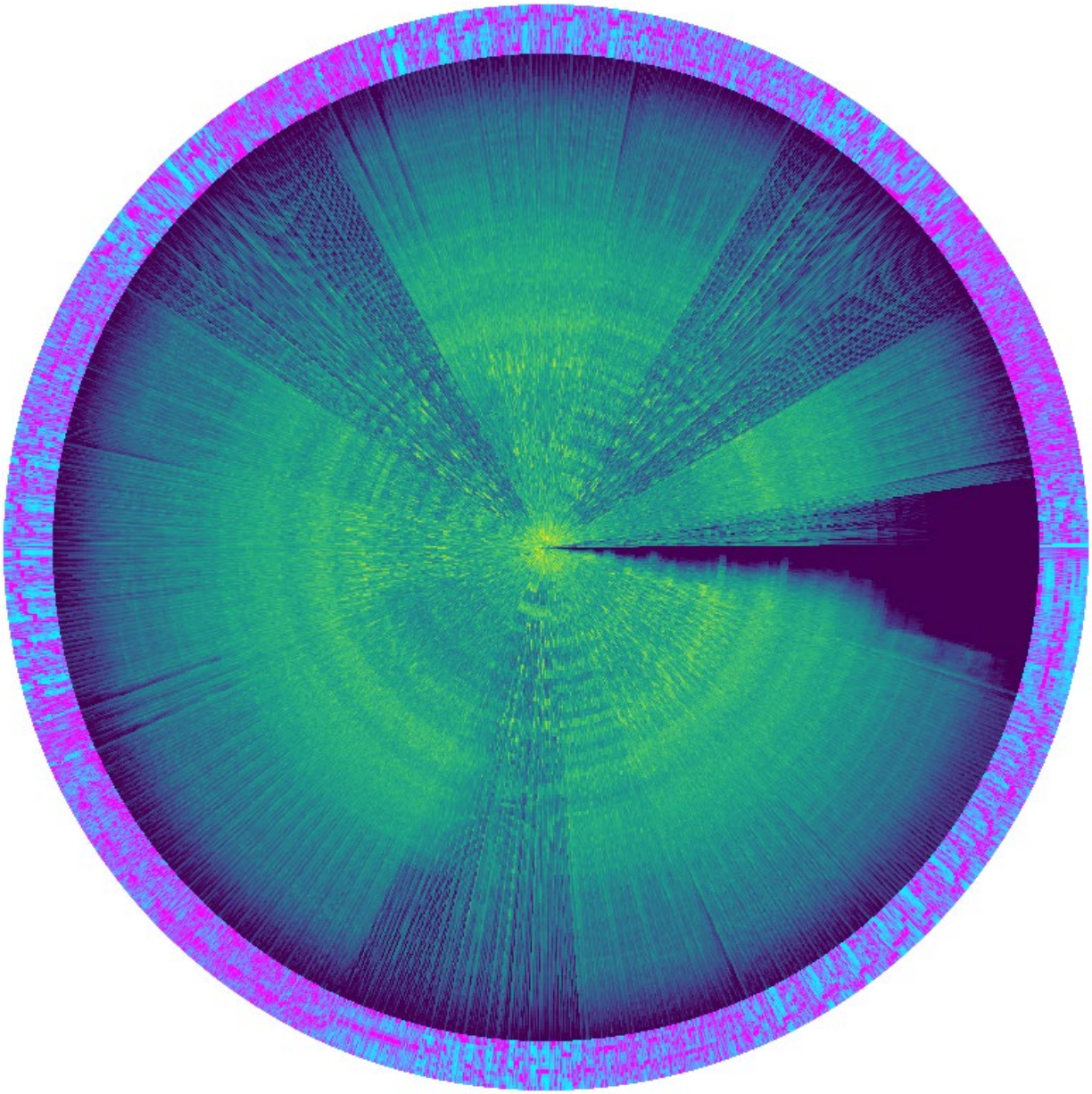
```
# Plotting the parametric EQ curve
plt.semilogx(frequencies, gains, marker='o', linestyle='-', color='black')

# Adding color coded areas for different filters
plt.fill_between(frequencies, gains, where=gains > 0, interpolate=True, color='grey', alpha=0.3)
plt.fill_between(frequencies, gains, where=gains <= 0, interpolate=True, color='grey', alpha=0.3)

# Highlight specific EQ bands with colors
colors = ['green', 'yellow', 'purple', 'fuchsia', 'blue', 'red']
filters = [(20, 100), (10000, 20000), (20, 400), (400, 1600), (1600, 6400), (6400, 20000)]
for (low_cut, high_cut), color in zip(filters, colors):
    plt.fill_between(frequencies, gains, where=(frequencies >= low_cut) & (frequencies <= high_cut),
interpolate=True, color=color, alpha=0.7)

# Setting the x-axis limits to the audible range
plt.xlim(20, 20000)

# Setting the y-axis limits to the dB range of the EQ
plt.ylim(-18, 18)

# Labels and grid
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
plt.title('Parametric EQ Curve for "Smells Like Teen Spirit"')
plt.grid(True, which="both", ls="--", linewidth=0.5)

# Adding key frequency markers
for freq in frequencies:
    plt.axvline(x=freq, color='k', linestyle='--', linewidth=0.5, alpha=0.7)

# Show the plot
plt.show()
```



In [ ]:

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

# Load the audio file
y, sr = librosa.load('01 Smells Like Teen Spirit.wav', sr=None)

# Extract features
spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Setup mesh for spectrogram
theta = np.linspace(0, 2*np.pi, spectrogram_db.shape[1] + 1)
r = np.linspace(0, 1, spectrogram_db.shape[0] + 1)
theta, r = np.meshgrid(theta, r)
ax.pcolormesh(theta, r, spectrogram_db, shading='flat')

# Setup mesh for chromagram
chroma_theta = np.linspace(0, 2*np.pi, chromagram.shape[1] + 1)
chroma_r = np.linspace(1, 1.1, chromagram.shape[0] + 1)
chroma_theta, chroma_r = np.meshgrid(chroma_theta, chroma_r)
ax.pcolormesh(chroma_theta, chroma_r, chromagram, shading='flat', cmap='cool')
```

plt.show()



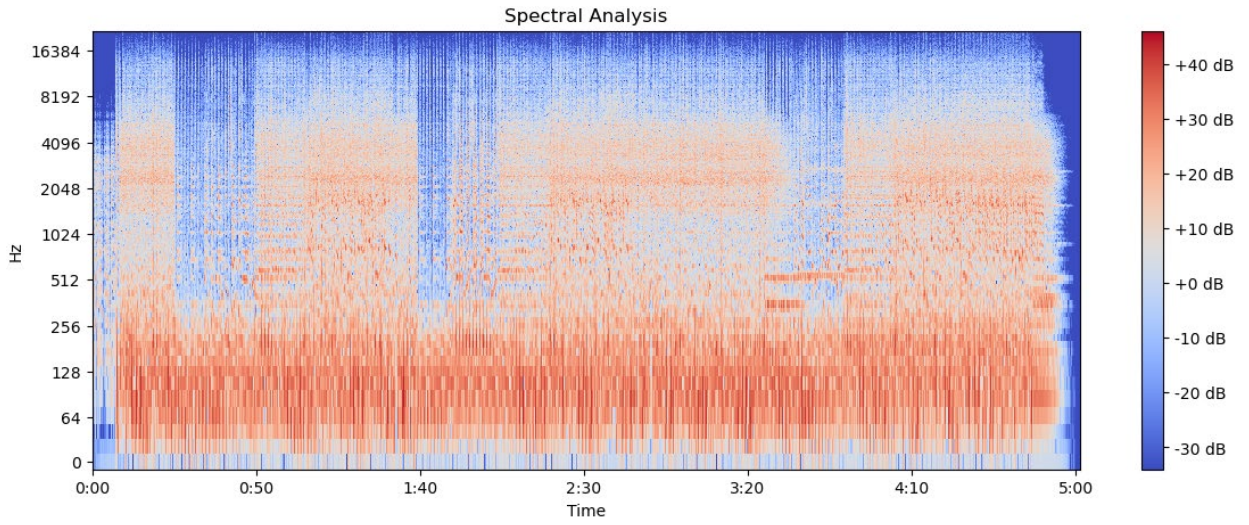
Spectral Centroid - To understand where most of the energy of the track is concentrated across frequencies. Spectral Roll-off - To see the frequency below which a specified percentage of the total spectral energy lies, useful for determining the brightness of the sound. Spectral Bandwidth - To measure the width of the band of light at half the maximum power (or half bandwidth), which indicates the spread of the sound. Here's how we can proceed with the Python code using librosa for these analyses:

PURPLE RAIN

In [ ]:

```
# Load the audio file
file_path = '01 Smells Like Teen Spirit.wav'
audio, sample_rate = librosa.load(file_path, sr=None)
```

```
# Perform a Short-Time Fourier Transform (STFT) to get the frequency content over time
stft = librosa.stft(audio)

# Convert the STFT to decibels, which is a more useful measure for human hearing
db_stft = librosa.amplitude_to_db(abs(stft))

# Plot the spectrogram
plt.figure(figsize=(14, 5))
librosa.display.specshow(db_stft, sr=sample_rate, x_axis='time', y_axis='log', cmap='coolwarm')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectral Analysis')
plt.show()
```



In [ ]:

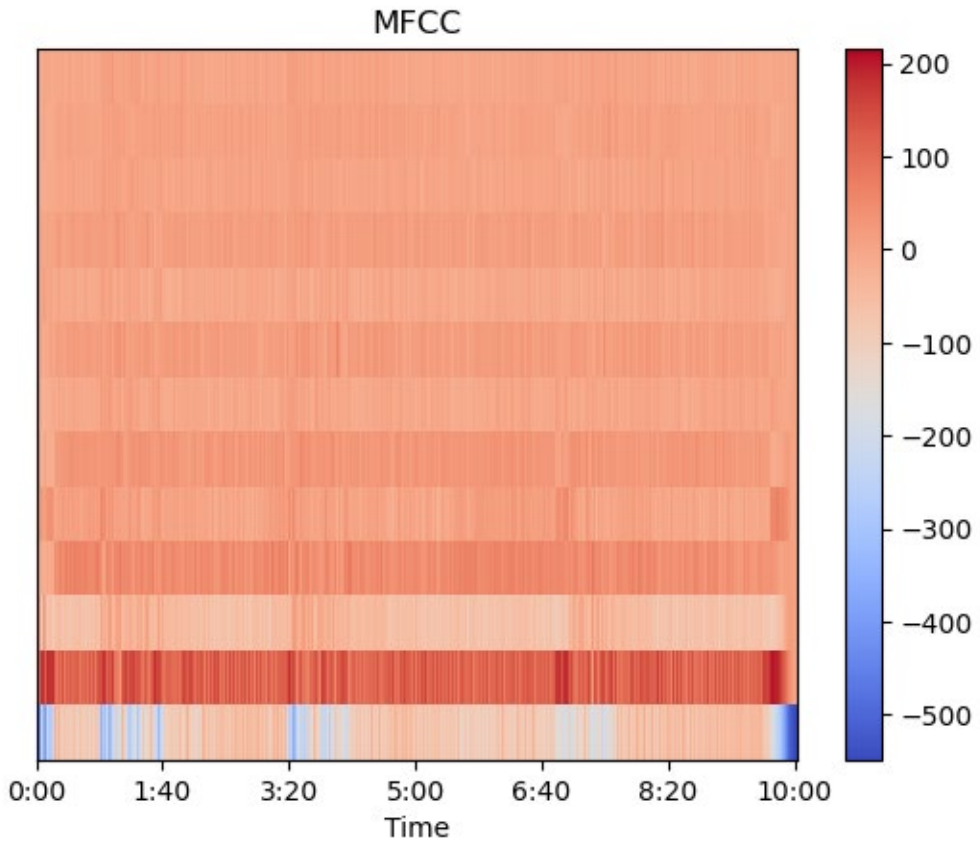
```
# Calculate the spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]
```

In [ ]:

```
# Convert frame counts to time
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames, sr=sr)
```

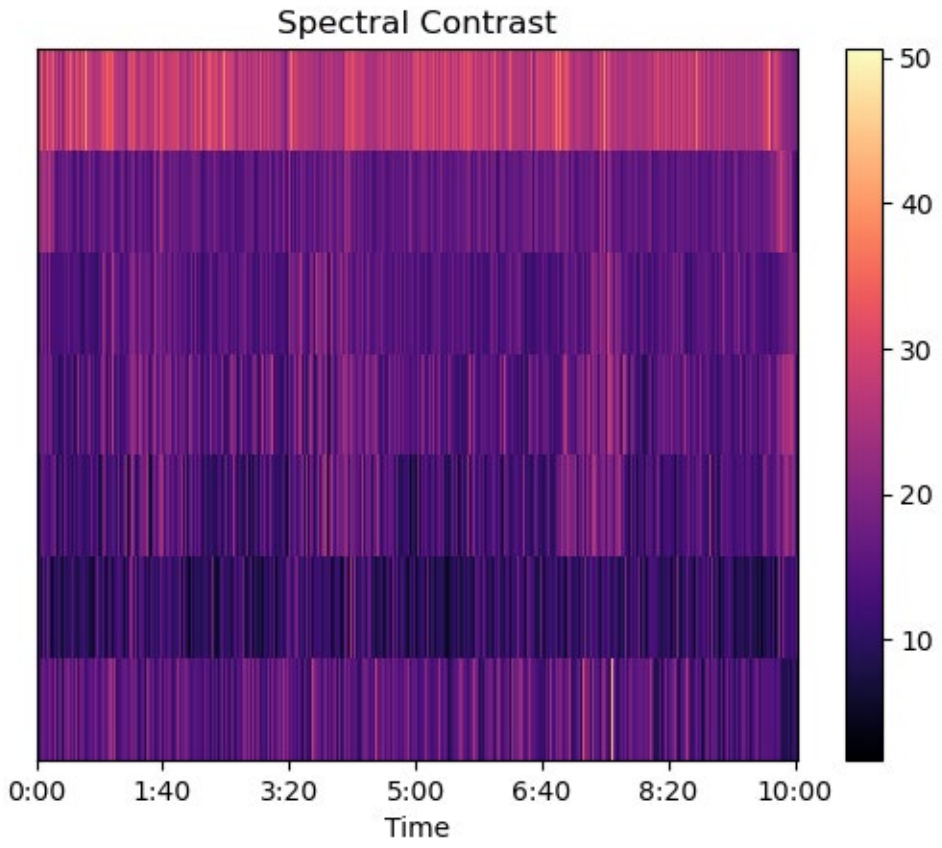
In [ ]:

```
# Mel-Frequency Cepstral Coefficients (MCFFs):
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('MFCC')
plt.show()
```



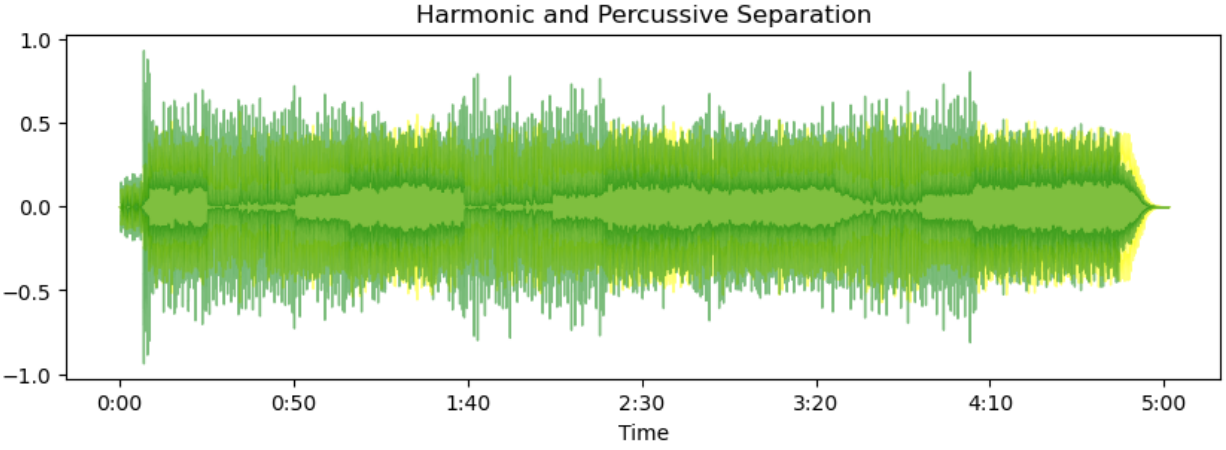
In [ ]:

```
# Spectral Contrast:
spec_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
librosa.display.specshow(spec_contrast, x_axis='time')
plt.colorbar()
plt.title('Spectral Contrast')
plt.show()
```



In [ ]:

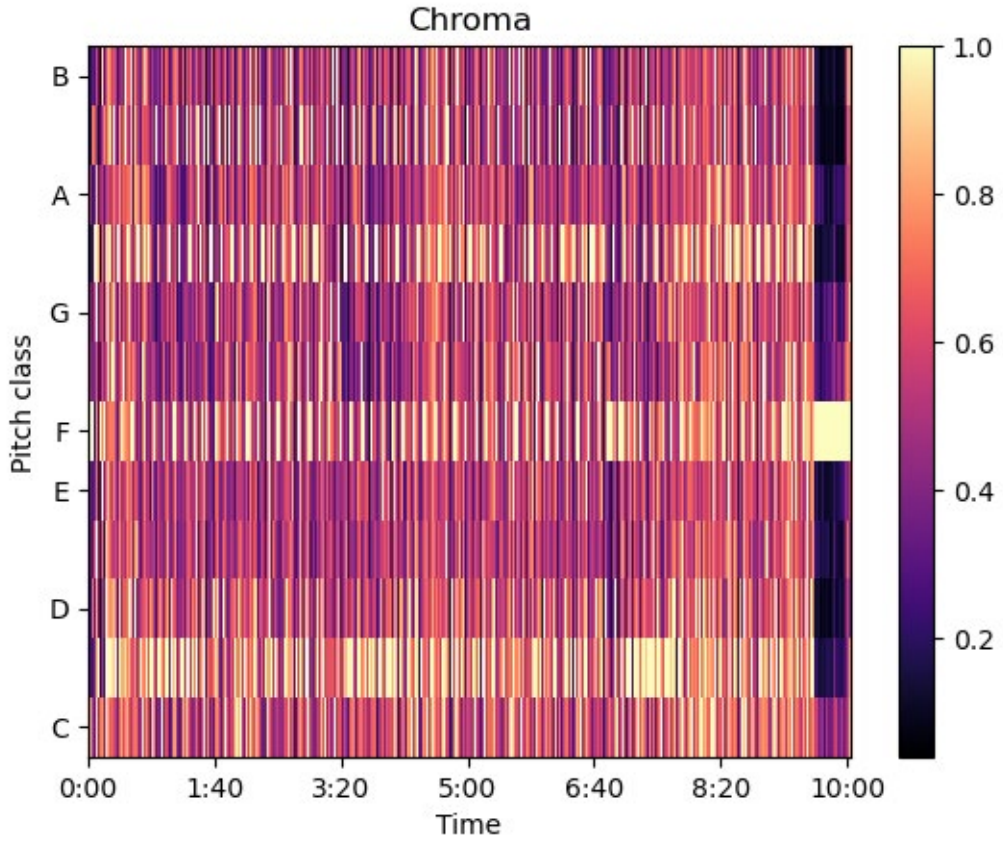
```
y_harm, y_perc = librosa.effects.hpss(y)
plt.figure(figsize=(10, 3))
librosa.display.waveshow(y_harm, sr=sr, alpha=0.5, color='yellow')
librosa.display.waveshow(y_perc, sr=sr, alpha=0.5, color='green')
plt.title('Harmonic and Percussive Separation')
plt.show()
```



In [ ]:

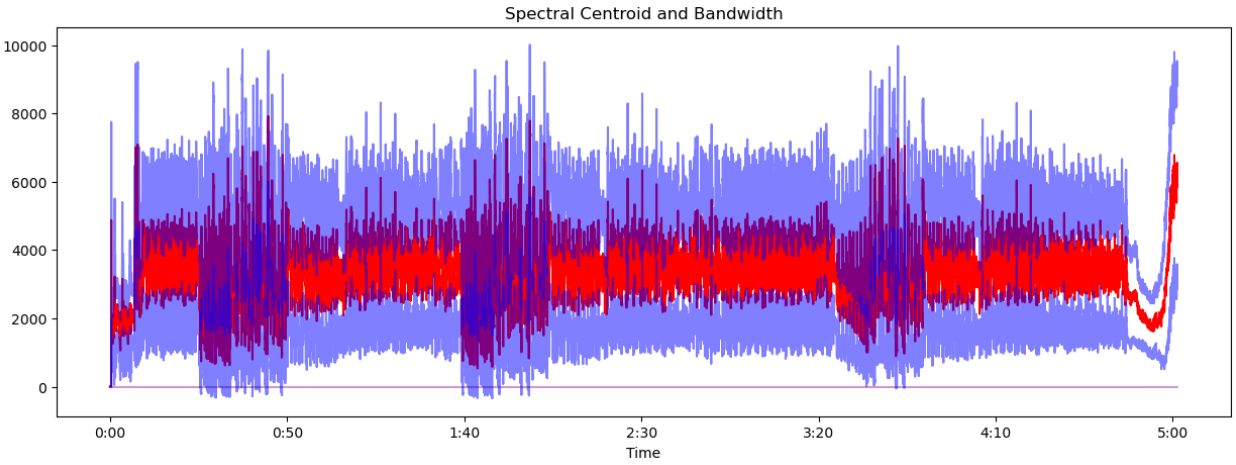
```
# Chroma Feature Extraction:
chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
librosa.display.specshow(chroma, y_axis='chroma', x_axis='time')
plt.colorbar()
```

```
plt.title('Chroma')
plt.show()
```



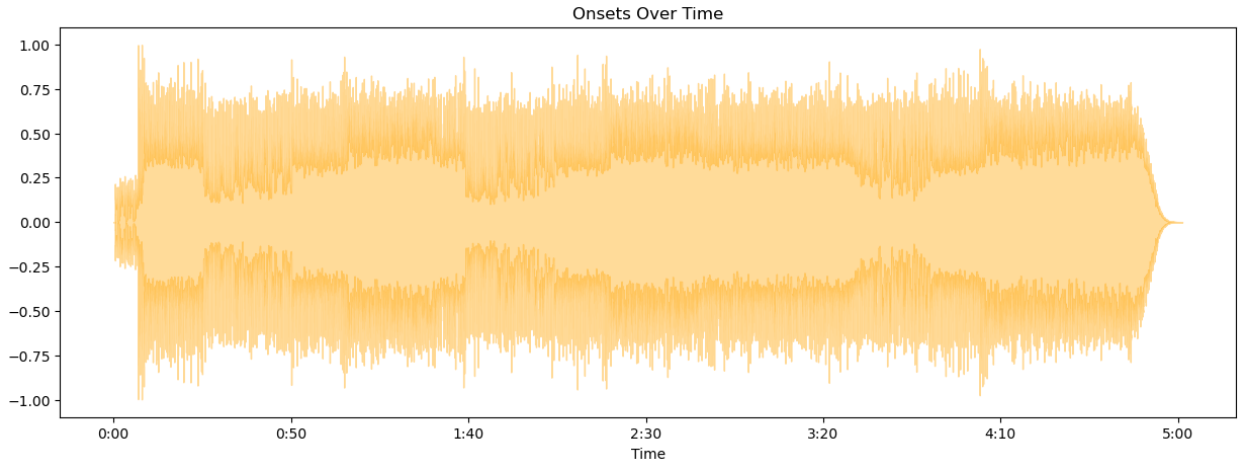
In [ ]:

```
# Plotting the Spectral Centroid along the waveform
plt.figure(figsize=(15, 5))
librosa.display.waveshow(y, sr=sr, alpha=0.4, color="purple")
plt.plot(t, spectral_centroids, color='r') # Spectral centroid
plt.plot(t, spectral_centroids - spectral_bandwidth / 2, color='b', alpha=0.5) # Min range
plt.plot(t, spectral_centroids + spectral_bandwidth / 2, color='b', alpha=0.5) # Max range
plt.title('Spectral Centroid and Bandwidth')
plt.show()
```



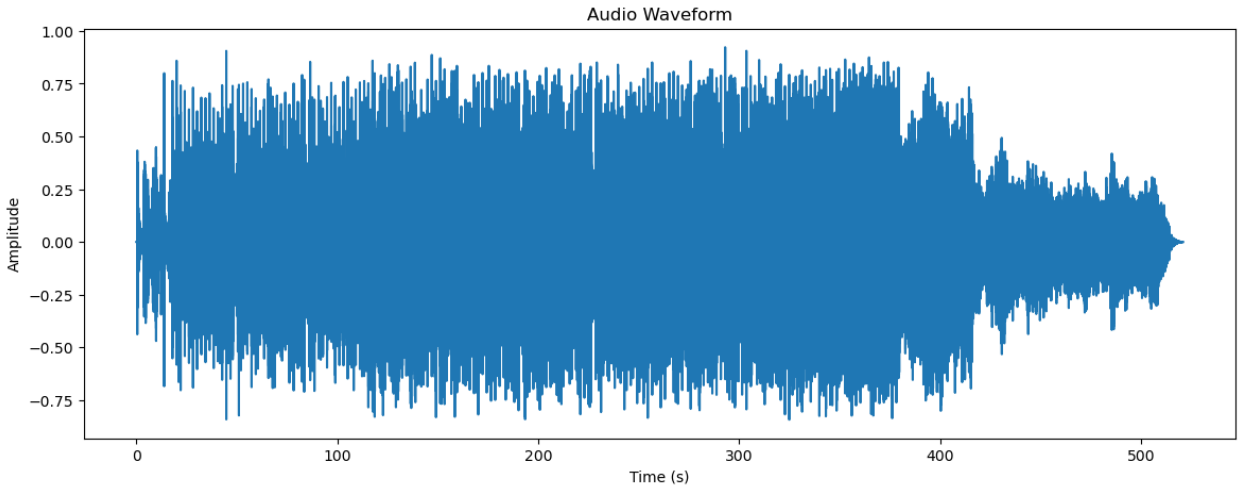
In [ ]:

```
# Detect onsets
onset_frames = librosa.onset.onset_detect(y=y, sr=sr)
onset_times = librosa.frames_to_time(onset_frames, sr=sr)

# Plotting the onsets
plt.figure(figsize=(15, 5))
# librosa.display.waveshow(y, sr=sr, alpha=0.4)
# plt.vlines(onset_times, ymin=-1, ymax=1, color='r')
librosa.display.waveshow(y, sr=sr, alpha=0.4, color="orange")
plt.title('Onsets Over Time')
plt.show()
```



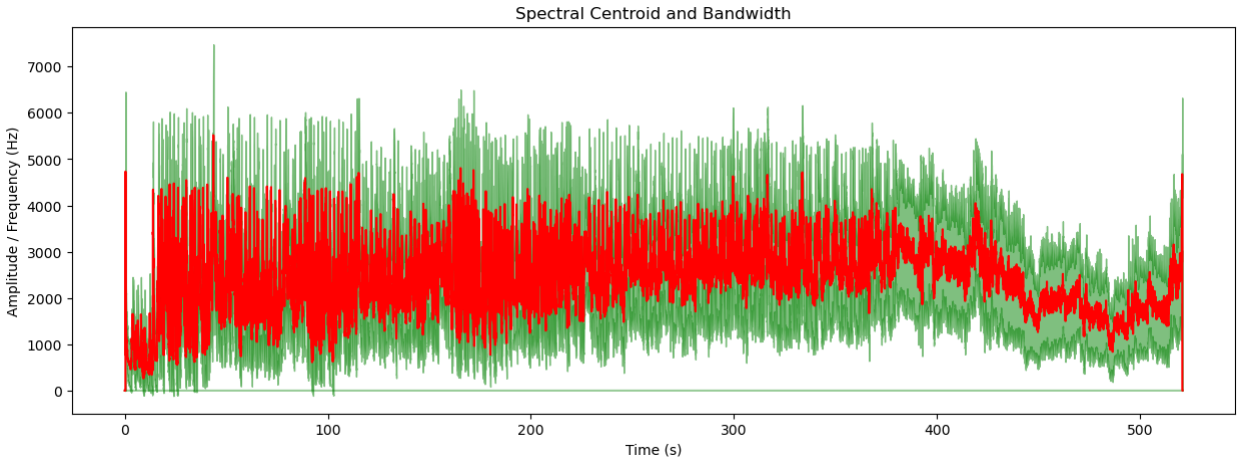
In [ ]:

```
# Load your audio file
y, sr = librosa.load('09 Purple Rain.wav')

# Generate time axis data
time = np.linspace(0, len(y) / sr, num=len(y))

# Plot the waveform
plt.figure(figsize=(14, 5))
plt.plot(time, y)
plt.title('Audio Waveform')
plt.ylabel('Amplitude')
plt.xlabel('Time (s)')
plt.show()
```



In [ ]:

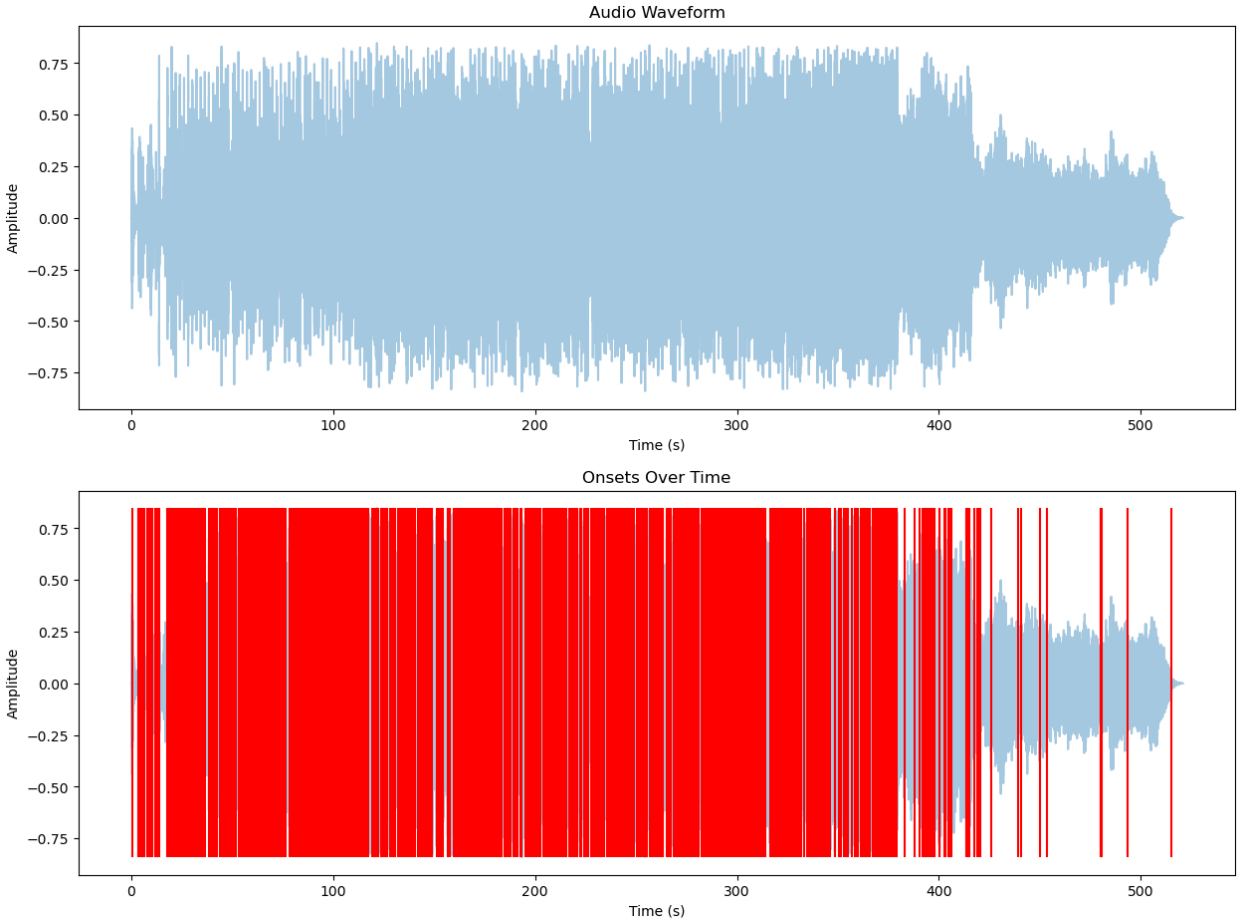
```
# Load your audio file
y, sr = librosa.load('09 Purple Rain.wav')

# Generate time axis data
t = np.linspace(0, len(y) / sr, num=len(y))

# Calculate spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]
frames = range(len(spectral_centroids))
t_centroid = librosa.frames_to_time(frames, sr=sr)  # Time for centroid plots



# Plot the audio waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4, color="green")  # Waveform in blue
plt.plot(t_centroid, spectral_centroids, color='r')  # Spectral centroid in red
plt.fill_between(t_centroid, spectral_centroids - spectral_bandwidth / 2, spectral_centroids +
spectral_bandwidth / 2, color='g', alpha=0.5)  # Bandwidth range in blue
plt.title('Spectral Centroid and Bandwidth')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / Frequency (Hz)')
plt.show()
```

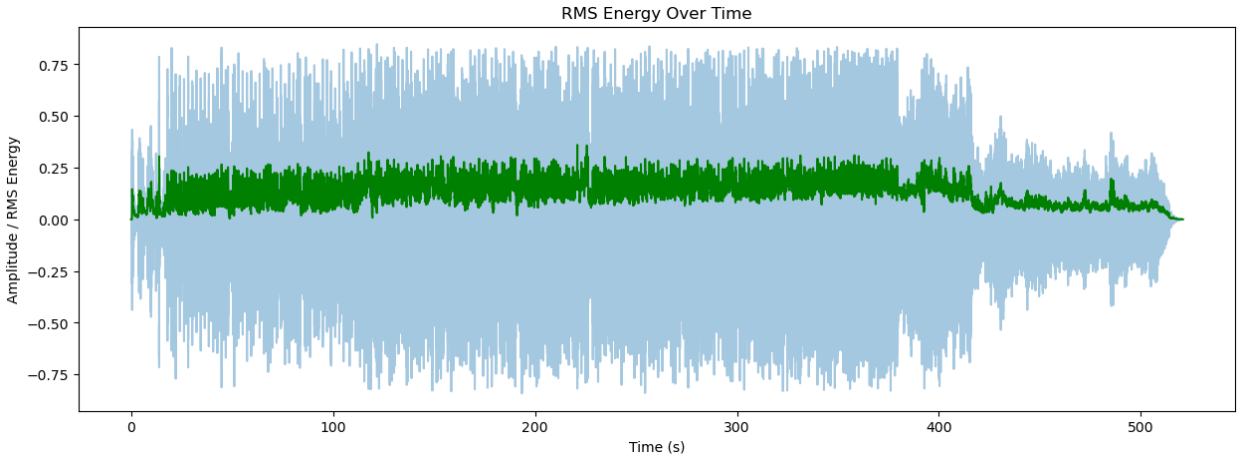


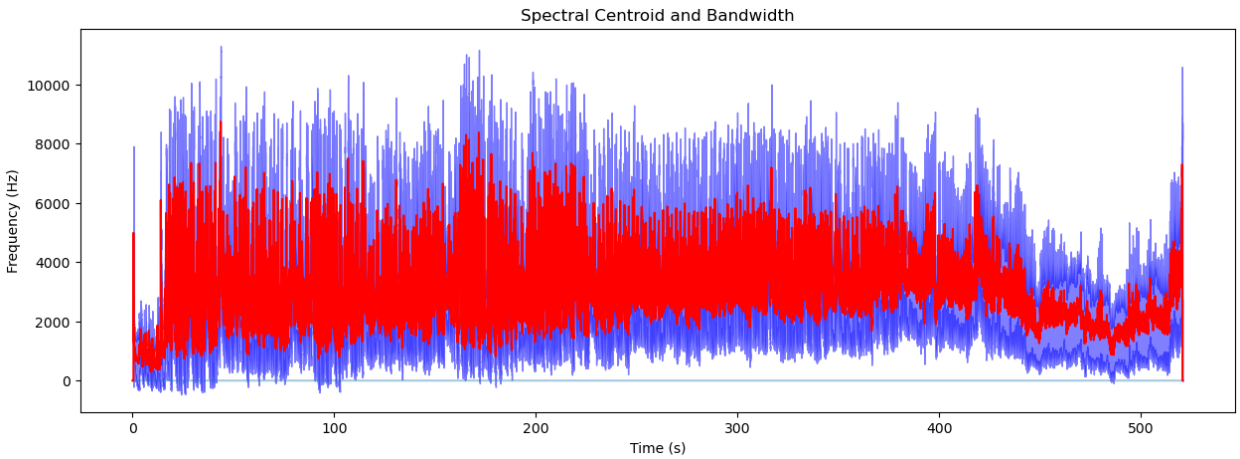
In [ ]:

```
# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=None)

# Time axis for the audio file
t = np.linspace(0, len(y) / sr, num=len(y))

# Plot the audio waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.title('Audio Waveform')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

# Detect onsets
onset_frames = librosa.onset.onset_detect(y=y, sr=sr)
onset_times = librosa.frames_to_time(onset_frames, sr=sr)

# Plotting the onsets over the waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.vlines(onset_times, ymin=min(y), ymax=max(y), color='r')
plt.title('Onsets Over Time')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

# Calculate the RMS energy
rms_energy = librosa.feature.rms(y=y)[0]
rms_times = librosa.frames_to_time(range(len(rms_energy)), sr=sr)

# Plotting the RMS energy over time
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.plot(rms_times, rms_energy, color='g')
plt.title('RMS Energy Over Time')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / RMS Energy')
plt.show()
```

```
# Calculate the spectral centroid and bandwidth
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]
centroid_times = librosa.frames_to_time(range(len(spectral_centroids)), sr=sr)

# Plotting the Spectral Centroid and Bandwidth over the waveform
plt.figure(figsize=(15, 5))
plt.plot(t, y, alpha=0.4)
plt.plot(centroid_times, spectral_centroids, color='r')
plt.fill_between(centroid_times, spectral_centroids - spectral_bandwidth / 2, spectral_centroids +
spectral_bandwidth / 2, color='b', alpha=0.5)
plt.title('Spectral Centroid and Bandwidth')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')
plt.show()
```

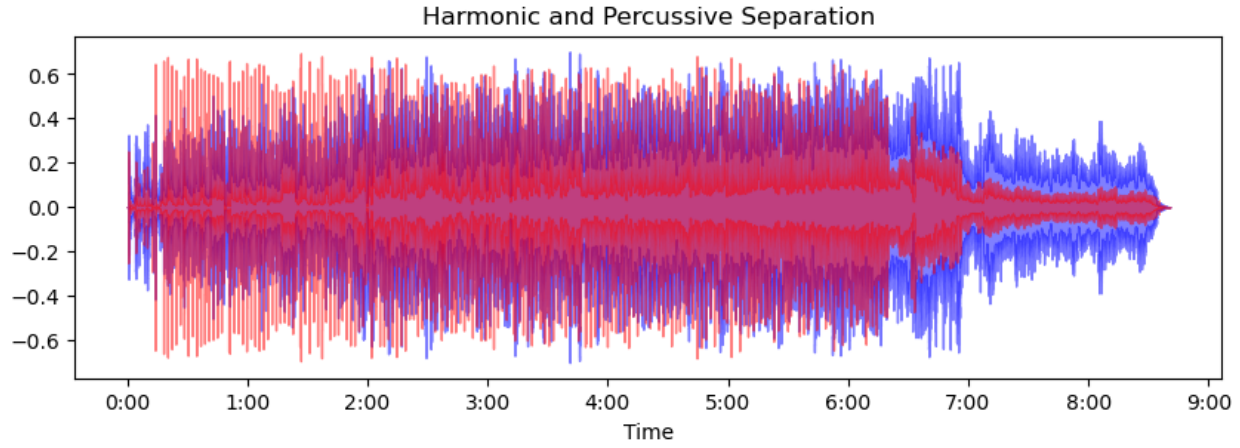






In [ ]:

```
# Harmonic & Percussove Separation:
y_harm, y_perc = librosa.effects.hpss(y)
plt.figure(figsize=(10, 3))
librosa.display.waveshow(y_harm, sr=sr, alpha=0.5, color='b')
librosa.display.waveshow(y_perc, sr=sr, alpha=0.5, color='r')
plt.title('Harmonic and Percussive Separation')
plt.show()
```



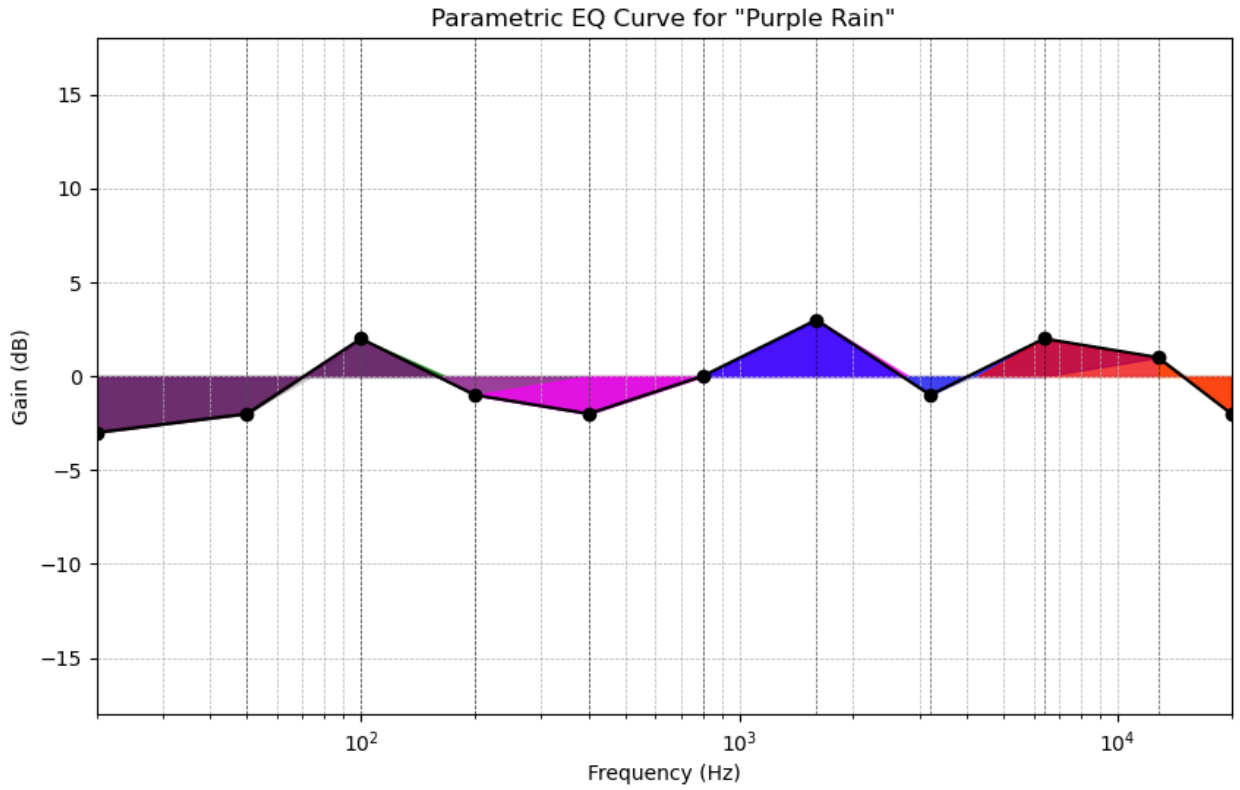
In [ ]:

```
# # Calculate the RMS energy
```

```
# rms_energy = librosa.feature.rms(y=y)[0]

# # Plotting the RMS along the waveform
# plt.figure(figsize=(15, 5))
# librosa.display.waveshow(y, sr=sr, alpha=0.4, color="navy")
# plt.plot(t, rms_energy, color='g') # RMS Energy
# plt.title('RMS Energy Over Time')
# plt.show()
```

In [ ]:

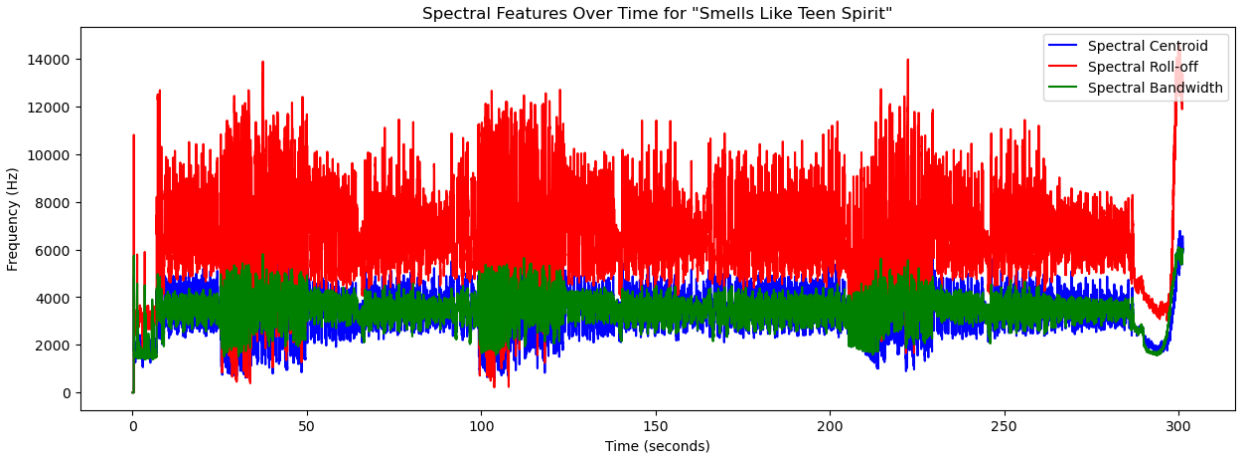
```
# Frequency bands in Hz (mock values)
frequencies = np.array([20, 50, 100, 200, 400, 800, 1600, 3200, 6400, 12800, 20000])
# Gain values for each band in dB (mock values)
gains = np.array([-3, -2, 2, -1, -2, 0, 3, -1, 2, 1, -2])

# Create the plot
plt.figure(figsize=(10, 6))

# Plotting the parametric EQ curve
plt.semilogx(frequencies, gains, marker='o', linestyle='-', color='black')

# Adding color coded areas for different filters
plt.fill_between(frequencies, gains, where=gains>0, interpolate=True, color='grey', alpha=0.3)
plt.fill_between(frequencies, gains, where=gains<=0, interpolate=True, color='grey', alpha=0.3)

# Highlight specific EQ bands with colors
colors = ['green', 'yellow', 'purple', 'fuchsia', 'blue', 'red']
filters = [(20, 100), (10000, 20000), (20, 400), (400, 1600), (1600, 6400), (6400, 20000)]
for (low_cut, high_cut), color in zip(filters, colors):
    plt.fill_between(frequencies, gains, where=(frequencies >= low_cut) & (frequencies <= high_cut),
interpolate=True, color=color, alpha=0.7)

# Setting the x-axis limits to the audible range
plt.xlim(20, 20000)

# Setting the y-axis limits to the dB range of the EQ
plt.ylim(-18, 18)

# Labels and grid
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
plt.title('Parametric EQ Curve for "Purple Rain"')
plt.grid(True, which="both", ls="--", linewidth=0.5)

# Adding key frequency markers
for freq in frequencies:
    plt.axvline(x=freq, color='k', linestyle='--', linewidth=0.5, alpha=0.7)

# Show the plot
plt.show()
```



In [ ]:

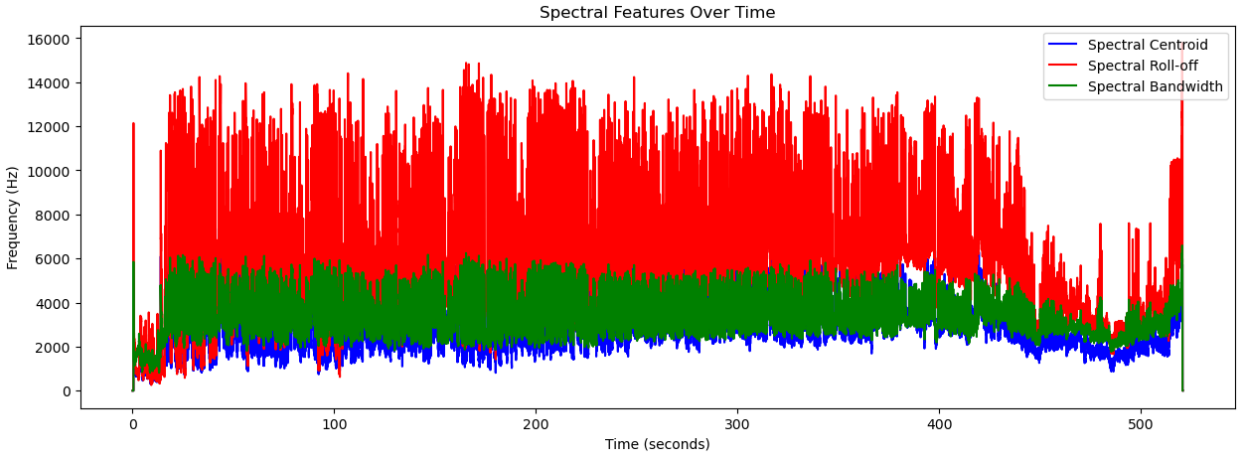
```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

# Load the audio file
y, sr = librosa.load('01 Smells Like Teen Spirit.wav', sr=None)

# Calculate the Short-Time Fourier Transform (STFT)
S = np.abs(librosa.stft(y))

# Compute spectral centroid, roll-off, and bandwidth
centroid = librosa.feature.spectral_centroid(S=S, sr=sr)
rolloff = librosa.feature.spectral_rolloff(S=S, sr=sr)
bandwidth = librosa.feature.spectral_bandwidth(S=S, sr=sr)

# Convert time samples into time (seconds)
frames = range(len(centroid[0]))
t = librosa.frames_to_time(frames, sr=sr)

# Plotting
plt.figure(figsize=(15, 5))
plt.plot(t, centroid[0], label='Spectral Centroid', color='b')
plt.plot(t, rolloff[0], label='Spectral Roll-off', color='r')
plt.plot(t, bandwidth[0], label='Spectral Bandwidth', color='g')
plt.title('Spectral Features Over Time for "Smells Like Teen Spirit"')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency (Hz)')
```

```
plt.legend(loc='upper right')
plt.show()
```



In [ ]:

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

# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=None)

# Calculate the Short-Time Fourier Transform (STFT)
S = np.abs(librosa.stft(y))

# Compute spectral centroid, roll-off, and bandwidth
centroid = librosa.feature.spectral_centroid(S=S, sr=sr)
rolloff = librosa.feature.spectral_rolloff(S=S, sr=sr)
bandwidth = librosa.feature.spectral_bandwidth(S=S, sr=sr)

# Convert time samples into time (seconds)
frames = range(len(centroid[0]))
t = librosa.frames_to_time(frames, sr=sr)

# Plotting
plt.figure(figsize=(15, 5))
plt.plot(t, centroid[0], label='Spectral Centroid', color='b')
plt.plot(t, rolloff[0], label='Spectral Roll-off', color='r')
plt.plot(t, bandwidth[0], label='Spectral Bandwidth', color='g')
plt.title('Spectral Features Over Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency (Hz)')
plt.legend(loc='upper right')
plt.show()
```



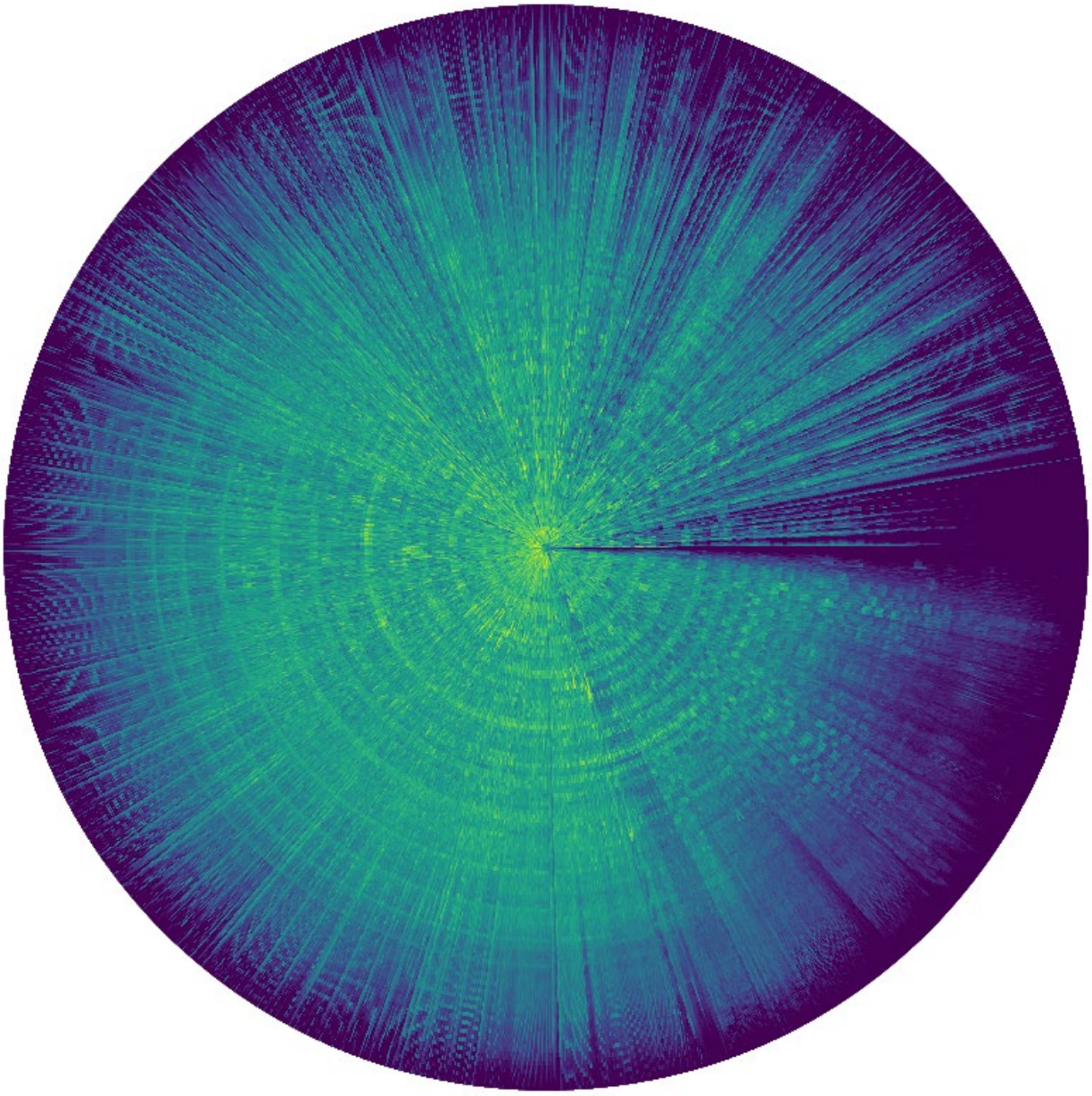
In [ ]:

```
zero_crossings = librosa.zero_crossings(y, pad=False)
print("Total Zero Crossings:", sum(zero_crossings))
```

Total Zero Crossings: 1518424

In [ ]:

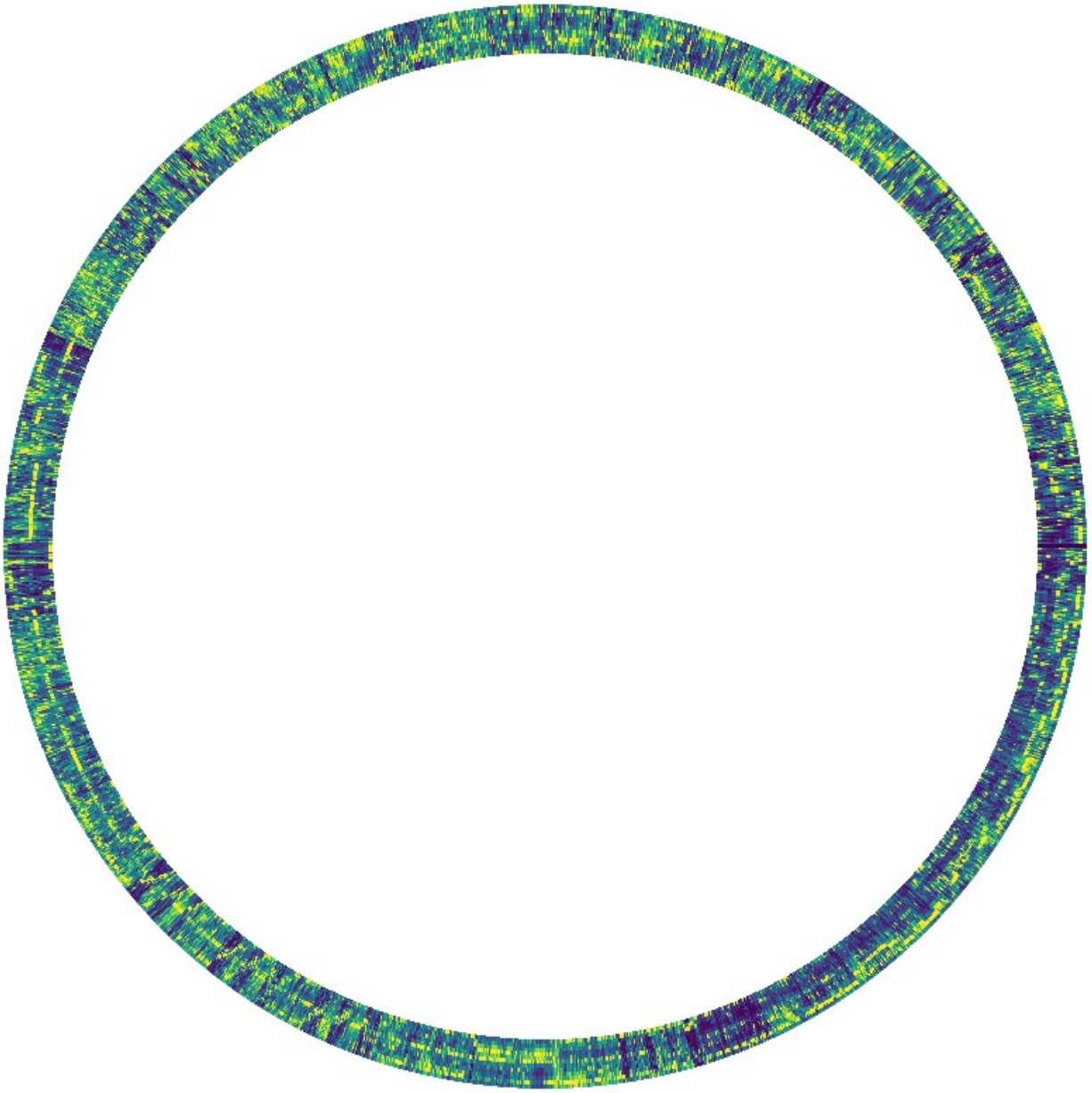
```
# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=None)

# Extract features
spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)

# Convert to decibels
spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Create the meshgrid for the spectrogram plot
# Ensure theta and r have one more point than the spectrogram_db dimensions
theta = np.linspace(0, 2*np.pi, spectrogram_db.shape[1] + 1)
r = np.linspace(0, 1, spectrogram_db.shape[0] + 1)
theta, r = np.meshgrid(theta, r)

# Plot the spectrogram
ax.pcolormesh(theta, r, spectrogram_db, shading='flat')

plt.show()
```



In [ ]:

```
# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=None)

# Extract features
chromagram = librosa.feature.chroma_stft(y=y, sr=sr)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Create the meshgrid for the chromagram plot
# Ensure theta and r have one more point than the chromagram dimensions
chroma_theta = np.linspace(0, 2*np.pi, chromagram.shape[1] + 1)
chroma_r = np.linspace(1, 1.1, chromagram.shape[0] + 1)
```

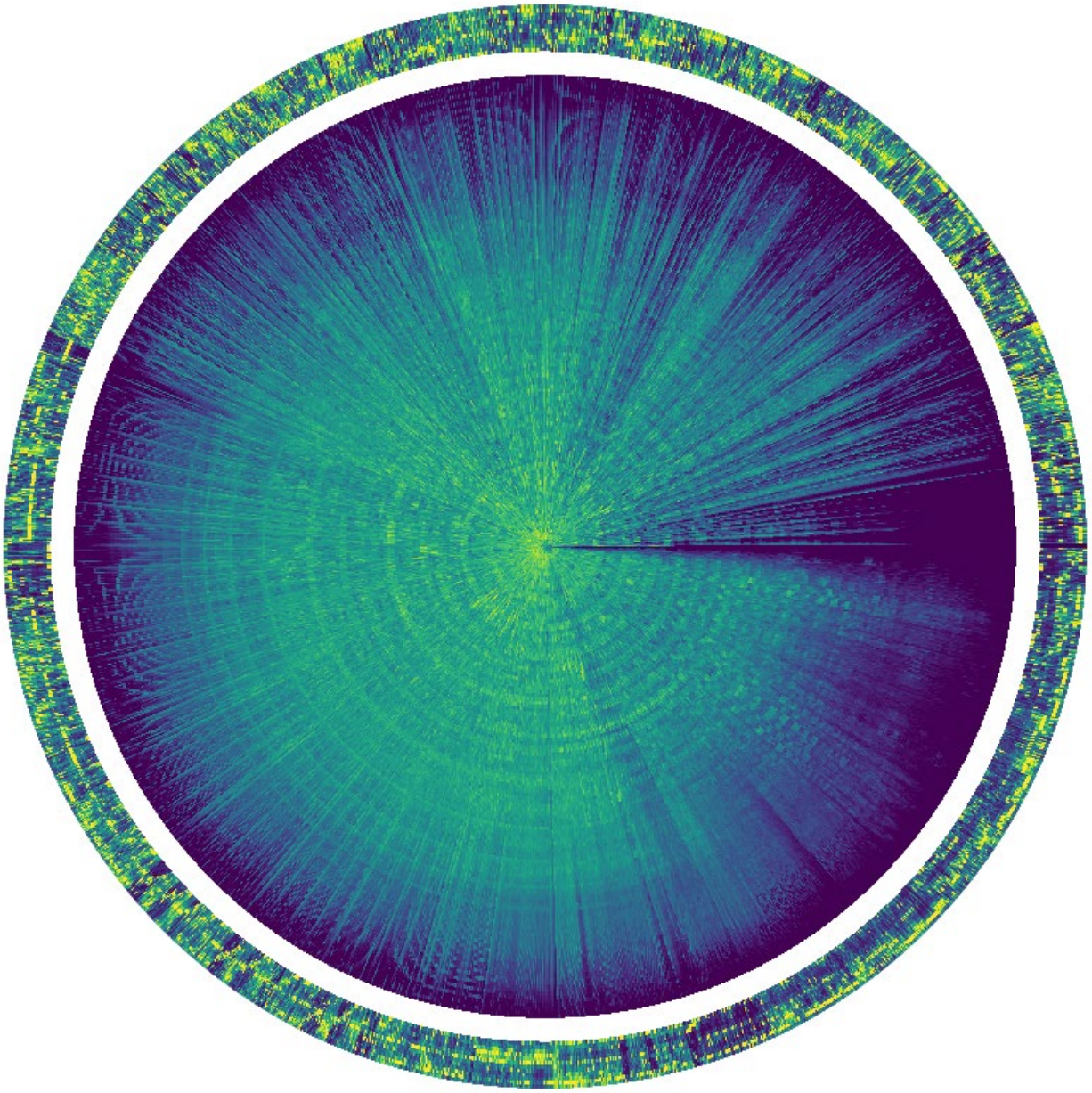
```
chroma_theta, chroma_r = np.meshgrid(chroma_theta, chroma_r)

# Plot the chromagram
ax.pcolormesh(chroma_theta, chroma_r, chromagram, shading='flat')

plt.show()
```



In [ ]:

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

# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=None)

# Extract features
```

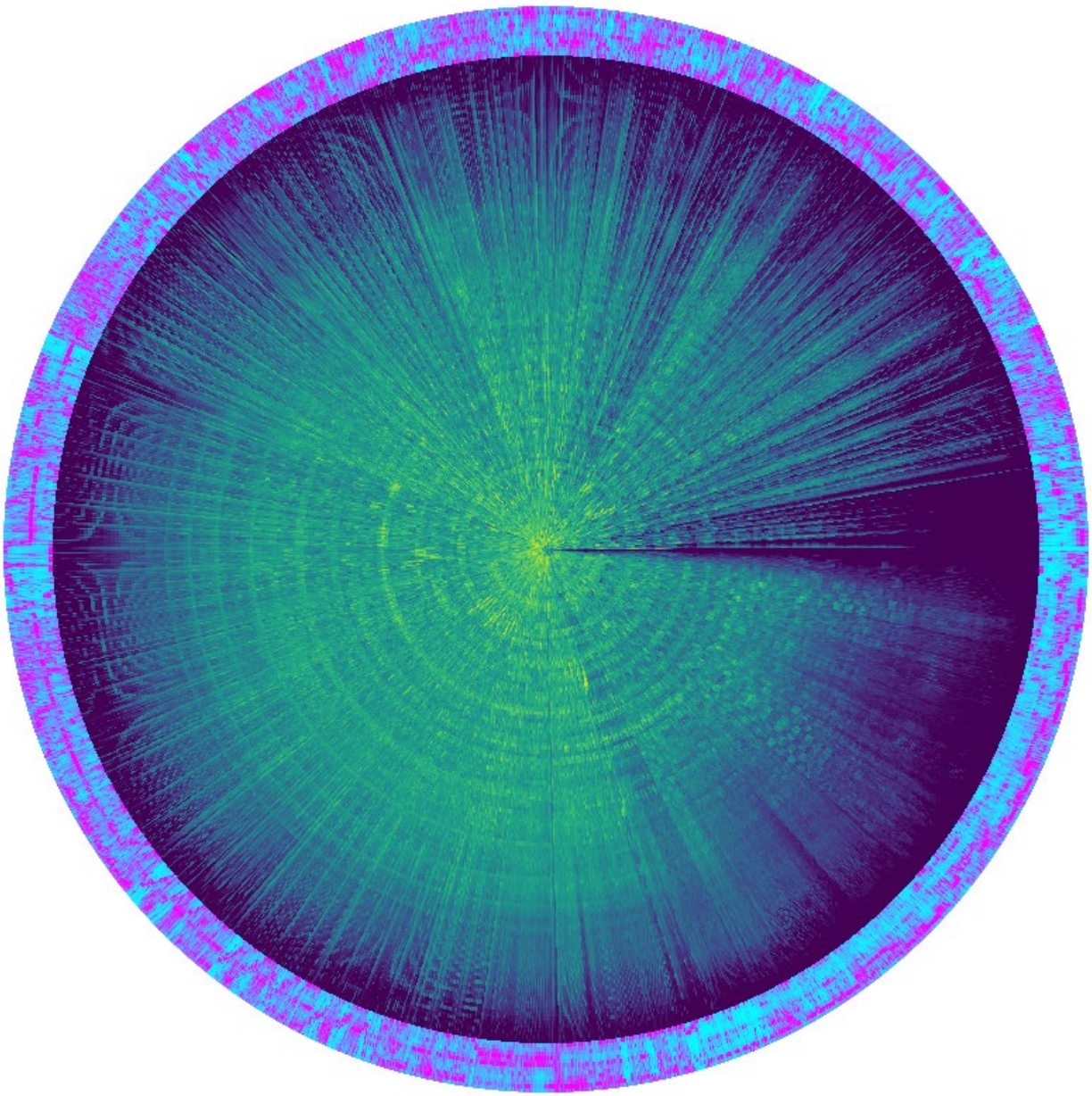
```
spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr)

# Convert spectrogram to decibels
spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.axis('off')

# Spectrogram grid
theta = np.linspace(0, 2 * np.pi, spectrogram_db.shape[1] + 1)
r = np.linspace(0, 1, spectrogram_db.shape[0] + 1)
theta_grid, r_grid = np.meshgrid(theta, r)

# Chromagram grid
chroma_theta = np.linspace(0, 2 * np.pi, chromagram.shape[1] + 1)
chroma_r = np.linspace(1.05, 1.15, chromagram.shape[0] + 1)  # Adjusted to sit just outside the spectrogram
chroma_theta_grid, chroma_r_grid = np.meshgrid(chroma_theta, chroma_r)

# Plot the spectrogram
ax.pcolormesh(theta_grid, r_grid, spectrogram_db, shading='flat')

# Plot the chromagram
ax.pcolormesh(chroma_theta_grid, chroma_r_grid, chromagram, shading='flat', cmap='viridis')  # Optionally set a different colormap

plt.show()
```



In [ ]:

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

# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=None)

# Extract features
spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr)

# Setup for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 10), subplot_kw={'projection': 'polar'})
```

```
ax.axis('off')

# Setup mesh for spectrogram
theta = np.linspace(0, 2*np.pi, spectrogram_db.shape[1] + 1)
r = np.linspace(0, 1, spectrogram_db.shape[0] + 1)
theta, r = np.meshgrid(theta, r)
ax.pcolormesh(theta, r, spectrogram_db, shading='flat')

# Setup mesh for chromagram
chroma_theta = np.linspace(0, 2*np.pi, chromagram.shape[1] + 1)
chroma_r = np.linspace(1, 1.1, chromagram.shape[0] + 1)
chroma_theta, chroma_r = np.meshgrid(chroma_theta, chroma_r)
ax.pcolormesh(chroma_theta, chroma_r, chromagram, shading='flat', cmap='cool')

plt.show()
```



In [ ]:

```
# Load the audio file at a lower sampling rate
y, sr = librosa.load('09 Purple Rain.wav', sr=22050)  # Downsample to 22050 Hz if higher
```

In [ ]:

```
# Load the full audio but trim to first 30 seconds
y, sr = librosa.load('09 Purple Rain.wav', sr=None, duration=30)  # Only load first 30 seconds
```

In [ ]:

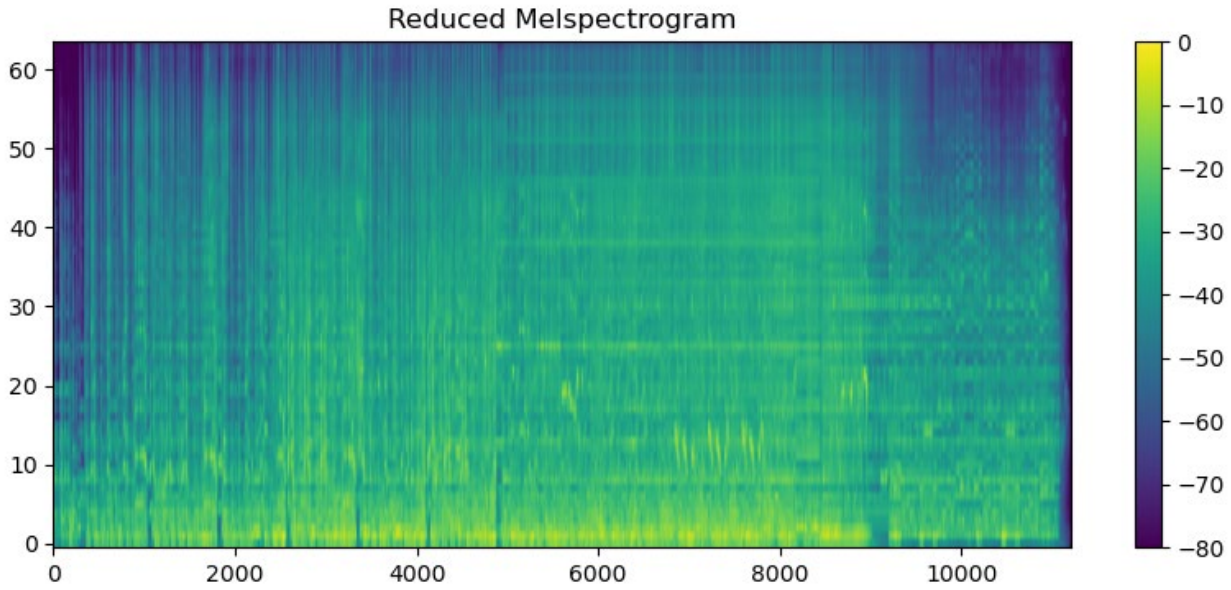
```
# Use fewer mel bands and a larger hop length
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=64, hop_length=1024)
```

In [ ]:

```
# Convert data to float16 for processing to save space
melspectrogram = melspectrogram.astype(np.float16)
```

In [ ]:

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

# Load the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050)  # Reduced sampling rate

# Extract features
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=64, hop_length=1024)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=1024)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=1024)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)

# Convert to decibels for the melspectrogram
melspectrogram_db = librosa.power_to_db(melspectrogram, ref=np.max)

# Normalize and prepare for padding
features = [melspectrogram_db, chromagram, spectral_contrast, tonnetz, harmonic]
features_padded = []

for f in features:
    # Check if the feature is 1D or 2D and apply appropriate padding
    if f.ndim == 1:
        # It's a 1D array
        padded = np.pad(f, (0, max(0, 500 - f.shape[0])), mode='constant', constant_values=0)
        padded = padded[np.newaxis, :]  # Make it 2D by adding an axis
    else:
        # It's a 2D array
        padded = np.pad(f, ((0, 0), (0, max(0, 500 - f.shape[1]))), mode='constant', constant_values=0)

    features_padded.append(padded)

# Visualizing a portion to manage memory
plt.figure(figsize=(10, 4))
plt.imshow(features_padded[0], aspect='auto', origin='lower')
plt.title('Reduced Melspectrogram')
plt.colorbar()
plt.show()
```



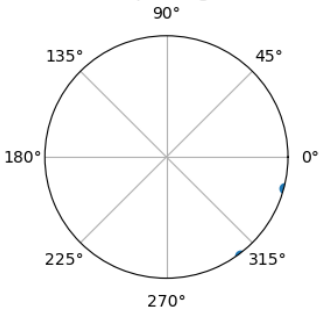
Goal Make Scatter Plot on Polar Axis

In [ ]:

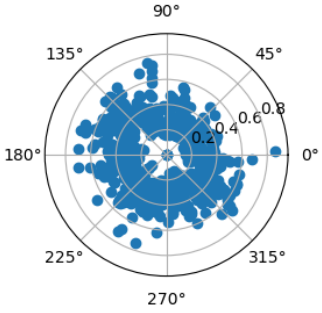
```
import librosa
import numpy as np
import matplotlib.pyplot as plt

# Load a segment of the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050, duration=60)  # Load only the first 60 seconds

# Extract features with a larger hop_length to reduce data size
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=32, hop_length=2048)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, n_chroma=12, hop_length=2048)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=2048)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)

# Convert to decibels for the melspectrogram
melspectrogram_db = librosa.power_to_db(melspectrogram, ref=np.max)

# Normalize and prepare for visualization
features = [melspectrogram_db, chromagram, spectral_contrast, tonnetz, harmonic]
feature_names = ['Mel Spectrogram', 'Chromagram', 'Spectral Contrast', 'Tonnetz', 'Harmonic']

# Create a figure with subplots for each feature
fig, axes = plt.subplots(len(features), 1, figsize=(8, 15), subplot_kw={'projection': 'polar'})

# Plot each feature in a separate subplot
for ax, feature, name in zip(axes, features, feature_names):
    if feature.ndim > 1:
        # Average the feature over its rows to reduce complexity
        feature = np.mean(feature, axis=0)
```

```
    theta = np.linspace(0, 2 * np.pi, num=feature.shape[0])
    r = feature  # Magnitude as radius

    ax.scatter(theta, r)
    ax.set_title(name)
    ax.set_ylim(0, np.max(r) + 0.1)  # Adjust the limits

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

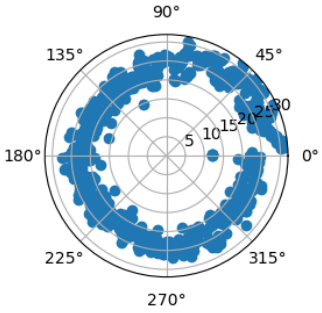
Mel Spectrogram



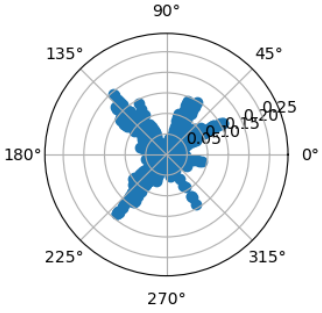
Chromagram



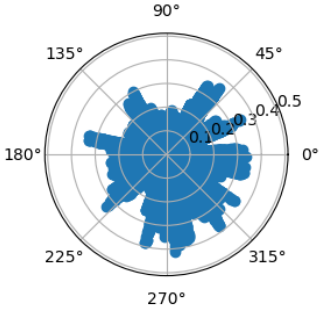
Spectral Contrast



Tonnetz

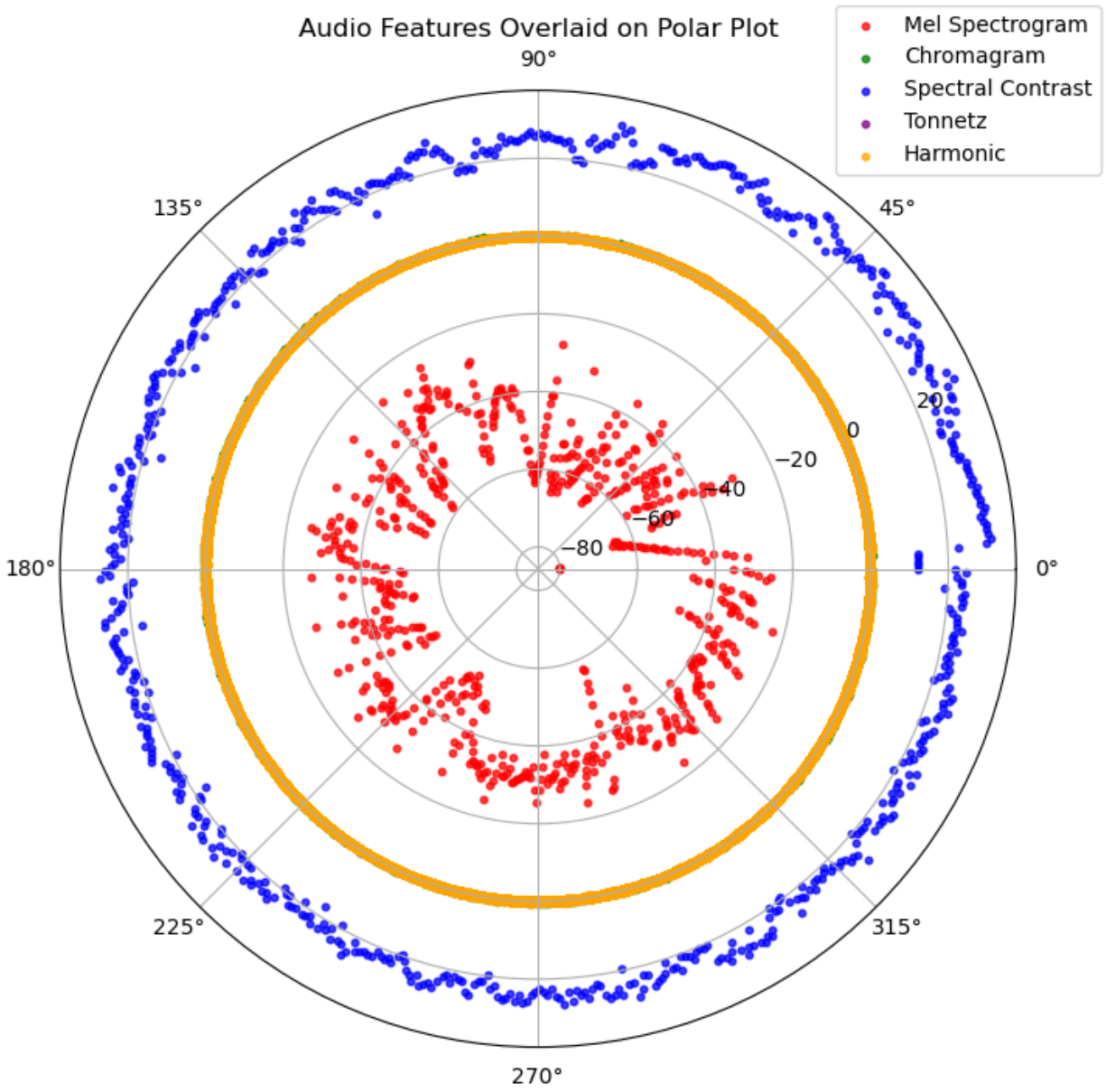


Harmonic



In [ ]:

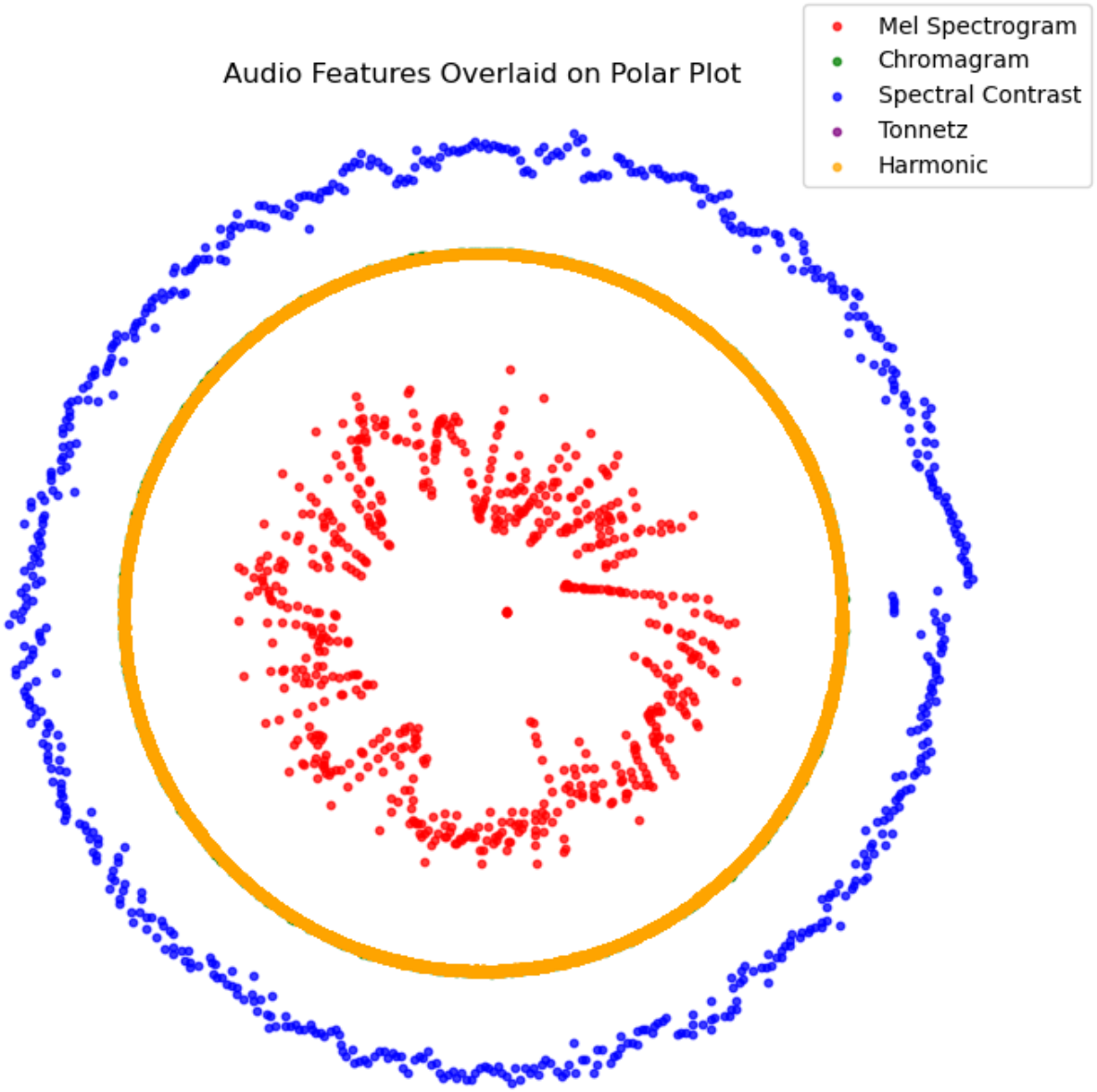
```
import librosa
import numpy as np
import matplotlib.pyplot as plt

# Load a segment of the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050, duration=60)  # Load only the first 60 seconds

# Extract features with a larger hop_length to reduce data size
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=32, hop_length=2048)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, n_chroma=12, hop_length=2048)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=2048)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)

# Convert to decibels for the melspectrogram
melspectrogram_db = librosa.power_to_db(melspectrogram, ref=np.max)

# Normalize and prepare for visualization
features = [melspectrogram_db, chromagram, spectral_contrast, tonnetz, harmonic]
feature_names = ['Mel Spectrogram', 'Chromagram', 'Spectral Contrast', 'Tonnetz', 'Harmonic']
colors = ['red', 'green', 'blue', 'purple', 'orange']  # Assign a color to each feature

# Create a polar plot
fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': 'polar'})

# Plot each feature in the same subplot with different colors
for feature, name, color in zip(features, feature_names, colors):
    if feature.ndim > 1:
        # Average the feature over its rows to reduce complexity
        feature = np.mean(feature, axis=0)

    theta = np.linspace(0, 2 * np.pi, num=feature.shape[0])
    r = feature  # Magnitude as radius

    ax.scatter(theta, r, label=name, color=color, alpha=0.75, s=10)  # Use smaller dots with some transparency

ax.set_title('Audio Features Overlaid on Polar Plot')
ax.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))  # Adjust legend position outside the plot

plt.show()
```



In [ ]:

```
# Load a segment of the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050, duration=60)  # Load only the first 60 seconds

# Extract features with a larger hop_length to reduce data size
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=32, hop_length=2048)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, n_chroma=12, hop_length=2048)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=2048)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)

# Convert to decibels for the melspectrogram
melspectrogram_db = librosa.power_to_db(melspectrogram, ref=np.max)

# Normalize and prepare for visualization
```

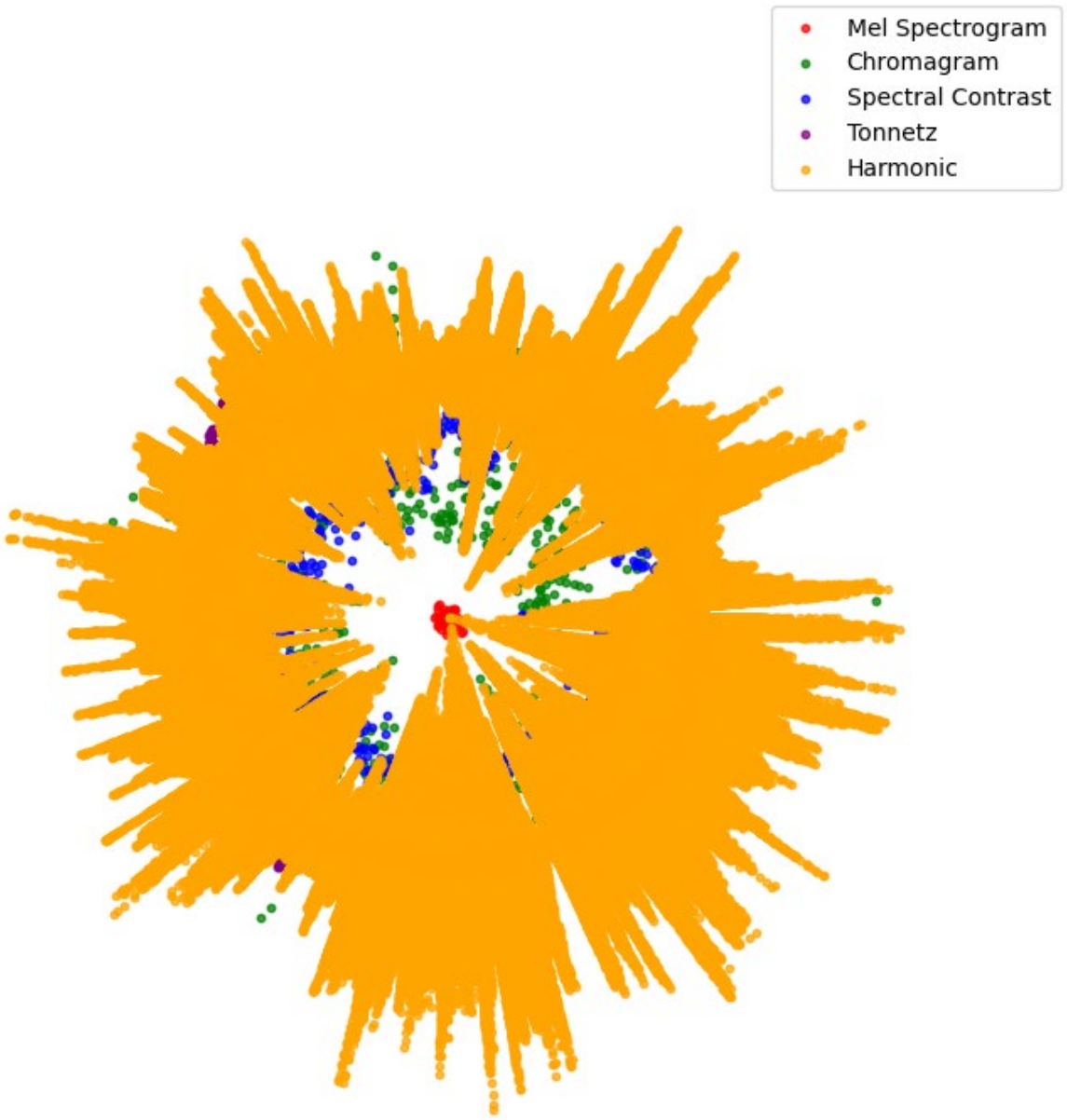
```
features = [melspectrogram_db, chromagram, spectral_contrast, tonnetz, harmonic]
feature_names = ['Mel Spectrogram', 'Chromagram', 'Spectral Contrast', 'Tonnetz', 'Harmonic']
colors = ['red', 'green', 'blue', 'purple', 'orange']  # Assign a color to each feature

# Create a polar plot
fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': 'polar'})

# Plot each feature in the same subplot with different colors
for feature, name, color in zip(features, feature_names, colors):
    if feature.ndim > 1:
        # Average the feature over its rows to reduce complexity
        feature = np.mean(feature, axis=0)

    theta = np.linspace(0, 2 * np.pi, num=feature.shape[0])
    r = feature  # Magnitude as radius

    ax.scatter(theta, r, label=name, color=color, alpha=0.75, s=10)  # Use smaller dots with some transparency

ax.set_title('Audio Features Overlaid on Polar Plot')
ax.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))  # Adjust legend position outside the plot
# Remove the polar labels
ax.set_xticklabels([])

# Remove the radial labels
ax.set_yticklabels([])

# Remove the grid
ax.grid(False)

# Remove the outer circle (spine)
ax.spines['polar'].set_visible(False)

# Optionally, if you want to remove the radial ticks as well:
ax.yaxis.set_ticks([])

# Show the plot without the polar labels and grid
plt.show()
```



In [ ]:

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

# Load a segment of the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050, duration=60)  # Load only the first 60 seconds

# Extract features
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=32, hop_length=2048)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=2048)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=2048)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)
```

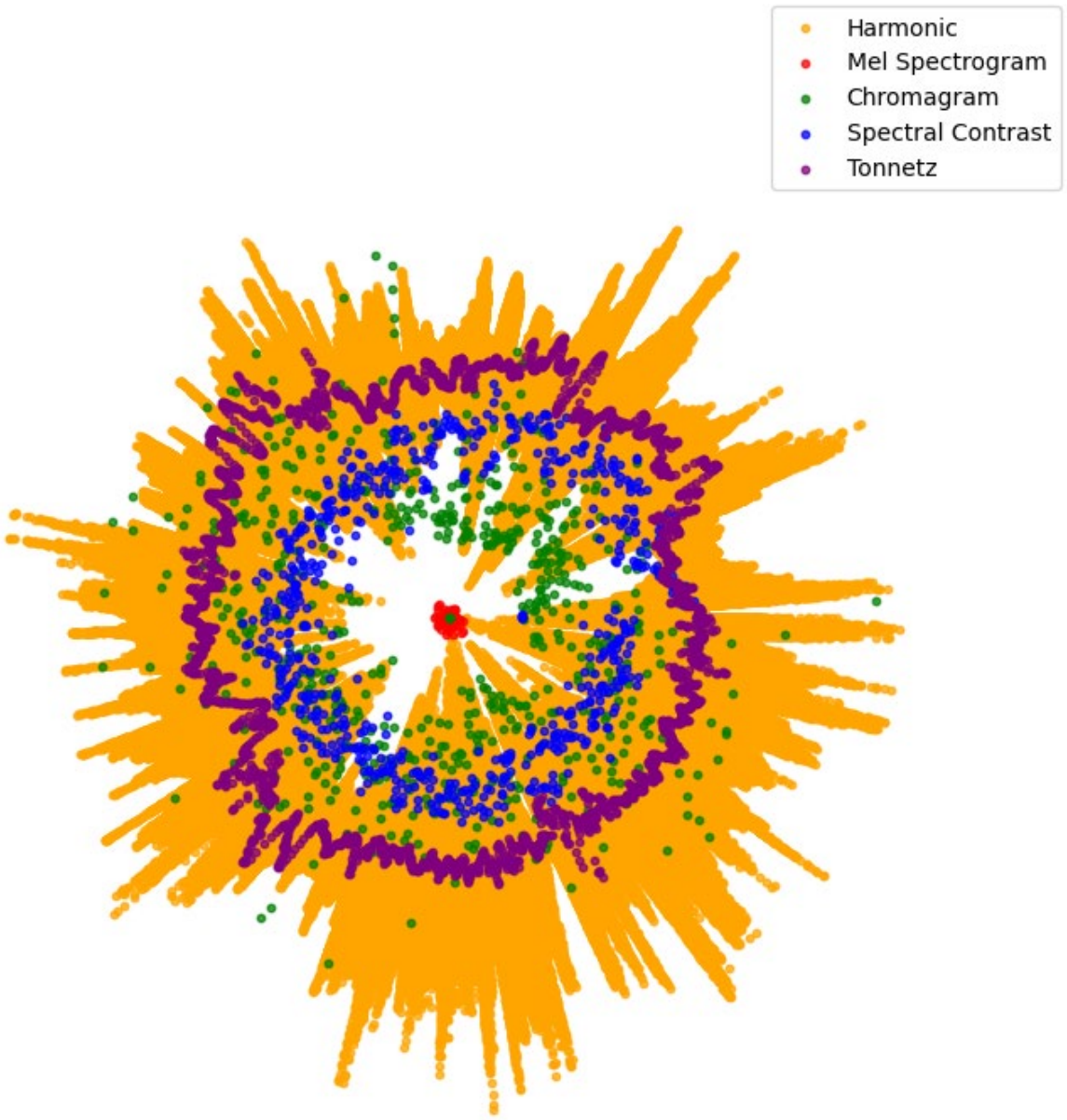
```
# Normalize the features to a common scale
features = [melspectrogram, chromagram, spectral_contrast, tonnetz, harmonic]
normalized_features = []

for feature in features:
    # Scale features to be between 0 and 1
    min_val = np.min(feature)
    max_val = np.max(feature)
    scaled_feature = (feature - min_val) / (max_val - min_val)
    normalized_features.append(scaled_feature)

# Create a polar plot
fig, ax = plt.subplots(figsize=(10, 8), subplot_kw={'projection': 'polar'})
feature_names = ['Mel Spectrogram', 'Chromagram', 'Spectral Contrast', 'Tonnetz', 'Harmonic']
colors = ['red', 'green', 'blue', 'purple', 'orange']  # Assign a color to each feature

# Plot each normalized feature in the same subplot with different colors
for feature, name, color in zip(normalized_features, feature_names, colors):
    # Collapse feature dimensions if necessary
    if feature.ndim > 1:
        feature = np.mean(feature, axis=0)

    # Map to polar coordinates
    theta = np.linspace(0, 2 * np.pi, feature.size)
    r = feature

    ax.scatter(theta, r, alpha=0.75, s=10, label=name, color=color)  # Use smaller dots with some transparency

# Remove the polar labels
ax.set_xticklabels([])

# Remove the radial labels and grids
ax.set_yticklabels([])
ax.grid(False)

# Remove the outer circle (spine)
ax.spines['polar'].set_visible(False)

# Optionally, if you want to remove the radial ticks as well:
ax.yaxis.set_ticks([])

# Add a legend
ax.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))

# Show the plot
plt.show()
```



In [ ]:

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

# Load a segment of the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050, duration=60)  # Load only the first 60 seconds

# Extract features
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=32, hop_length=2048)
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=2048)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=2048)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)
```

```
# Normalize the features to a common scale
features = [harmonic, melspectrogram, chromagram, spectral_contrast, tonnetz]  # Place harmonic first
feature_names = ['Harmonic', 'Mel Spectrogram', 'Chromagram', 'Spectral Contrast', 'Tonnetz']
colors = ['orange', 'red', 'green', 'blue', 'purple']  # Corresponding colors, with harmonic's color first

normalized_features = []

for feature in features:
    # Scale features to be between 0 and 1
    min_val = np.min(feature)
    max_val = np.max(feature)
    scaled_feature = (feature - min_val) / (max_val - min_val)
    normalized_features.append(scaled_feature)

# Create a polar plot
fig, ax = plt.subplots(figsize=(10, 8), subplot_kw={'projection': 'polar'})

# Plot each normalized feature in the same subplot with different colors
for feature, name, color in zip(normalized_features, feature_names, colors):
    # Collapse feature dimensions if necessary
    if feature.ndim > 1:
        feature = np.mean(feature, axis=0)

    # Map to polar coordinates
    theta = np.linspace(0, 2 * np.pi, feature.size)
    r = feature

    ax.scatter(theta, r, alpha=0.75, s=10, label=name, color=color)  # Use smaller dots with some transparency

# Customize the plot - removing labels, ticks, and spines
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.grid(False)
ax.spines['polar'].set_visible(False)
ax.yaxis.set_ticks([])

# Add a legend
ax.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))

# Show the plot
plt.show()
```



Goal Make Scatterplot Polar Plot

In [ ]:

```
import plotly.graph_objects as go
import numpy as np
import librosa

# Load a segment of the audio file
y, sr = librosa.load('09 Purple Rain.wav', sr=22050, duration=60)  # Load only the first 60 seconds

# Extract features
melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=32, hop_length=2048)
```

```
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=2048)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr, hop_length=2048)
tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
harmonic, percussive = librosa.effects.hpss(y)

# Normalize the features to a common scale
features = [harmonic, melspectrogram, chromagram, spectral_contrast, tonnetz]
feature_names = ['Harmonic', 'Mel Spectrogram', 'Chromagram', 'Spectral Contrast', 'Tonnetz']
colors = ['orange', 'red', 'green', 'blue', 'purple']  # Colors for each feature

normalized_features = []
for feature in features:
    # Scale features to be between 0 and 1
    min_val = np.min(feature)
    max_val = np.max(feature)
    scaled_feature = (feature - min_val) / (max_val - min_val)
    normalized_features.append(scaled_feature)

# Create a Plotly figure
fig = go.Figure()

# Add traces for each feature
for feature, name, color in zip(normalized_features, feature_names, colors):
    if feature.ndim > 1:
        feature = np.mean(feature, axis=0)

    theta = np.linspace(0, 360, feature.size)  # Plotly uses degrees instead of radians
    r = feature

    fig.add_trace(go.Scatterpolar(
        r=r,
        theta=theta,
        mode='markers',
        name=name,
        marker=dict(color=color)
    ))

# Update layout to clean up the plot
fig.update_layout(
    polar=dict(
        radialaxis=dict(showticklabels=False, ticks=''),
        angularaxis=dict(showticklabels=False, ticks='')
    ),
    showlegend=True
)

# Export to HTML
fig.write_html("interactive_plot.html")
print("Interactive plot created successfully.")
```

```
Interactive plot created successfully.
```

In [ ]:

```
y, sr = librosa.load('14 Free Fallin.wav')
print('y:', y, '\n')
```

```
y: [0. 0. 0. ... 0. 0. 0.]
```

madonna pearl jam tom petty nirvana prince

In [ ]:

```
print('y shape:', np.shape(y), '\n')
```

```
y shape: (5637344,)
```

In [ ]:

```
print('Sample Rate (KHz):', sr, '\n')
```

```
Sample Rate (KHz): 22050
```

In [ ]:

```
print('Check Length of the audio in second:', 5637344/22050)
```

```
Check Length of the audio in second: 255.66185941043085
```

In [ ]:

```
# Trim leading and trailing silence from an audio signal (silence before and after the actual audio)
audio_file, _ = librosa.effects.trim(y)

# the result is an numpy ndarray
print('Audio File:', audio_file, '\n')
```

```
Audio File: [-9.0949470e-13 -1.3642421e-12  9.0949470e-12 ...  8.3927749e-05
  1.1652079e-05 -2.1102940e-05]
```

In [ ]:

```
print('Audio File shape:', np.shape(audio_file))
```

```
Audio File shape: (5571584,)
```

In [ ]:

```
plt.figure(figsize = (16, 6))
```

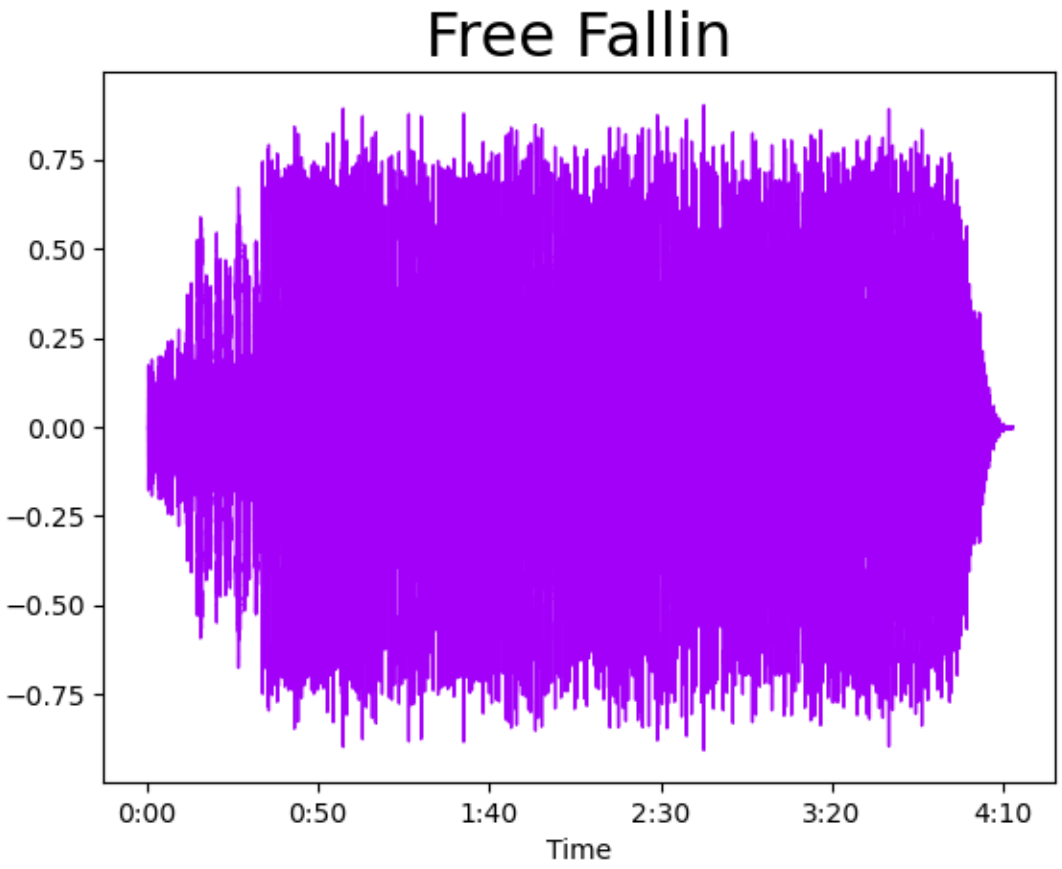
Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
```

<Figure size 1600x600 with 0 Axes>

In [ ]:

```
librosa.display.waveshow(y = audio_file, sr = sr, color = "#A300F9");
plt.title("Free Fallin", fontsize = 23);
plt.show()
```



In [ ]:

```
# Default FFT window size
n_fft = 2048 # FFT window size
hop_length = 512 # number audio of frames between STFT columns (looks like a good default)

# Short-time Fourier transform (STFT)
D = np.abs(librosa.stft(audio_file, n_fft = n_fft, hop_length = hop_length))

print('Shape of D object:', np.shape(D))
```

Shape of D object: (1025, 10883)

In [ ]:

```
plt.figure(figsize = (16, 6))
```

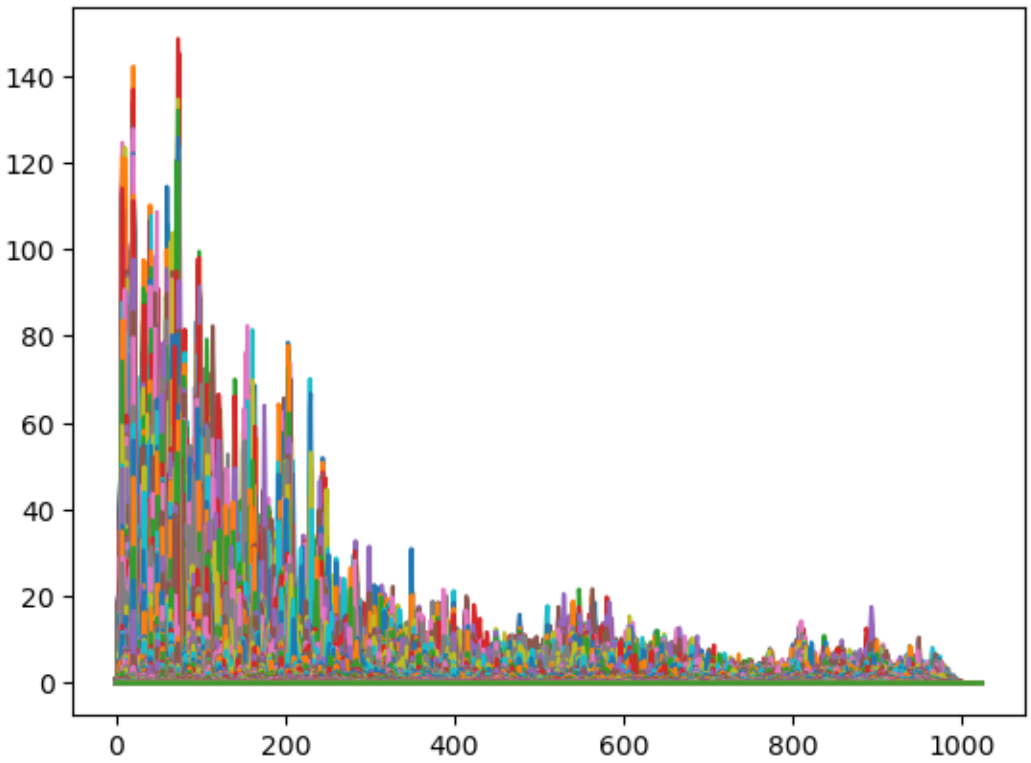
Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
```

<Figure size 1600x600 with 0 Axes>

In [ ]:

```
plt.plot(D);
plt.show()
```



In [ ]:

```
# Convert an amplitude spectrogram to Decibels-scaled spectrogram.
DB = librosa.amplitude_to_db(D, ref = np.max)

# Creating the Spectogram
plt.figure(figsize = (16, 6))
```

Out[ ]:

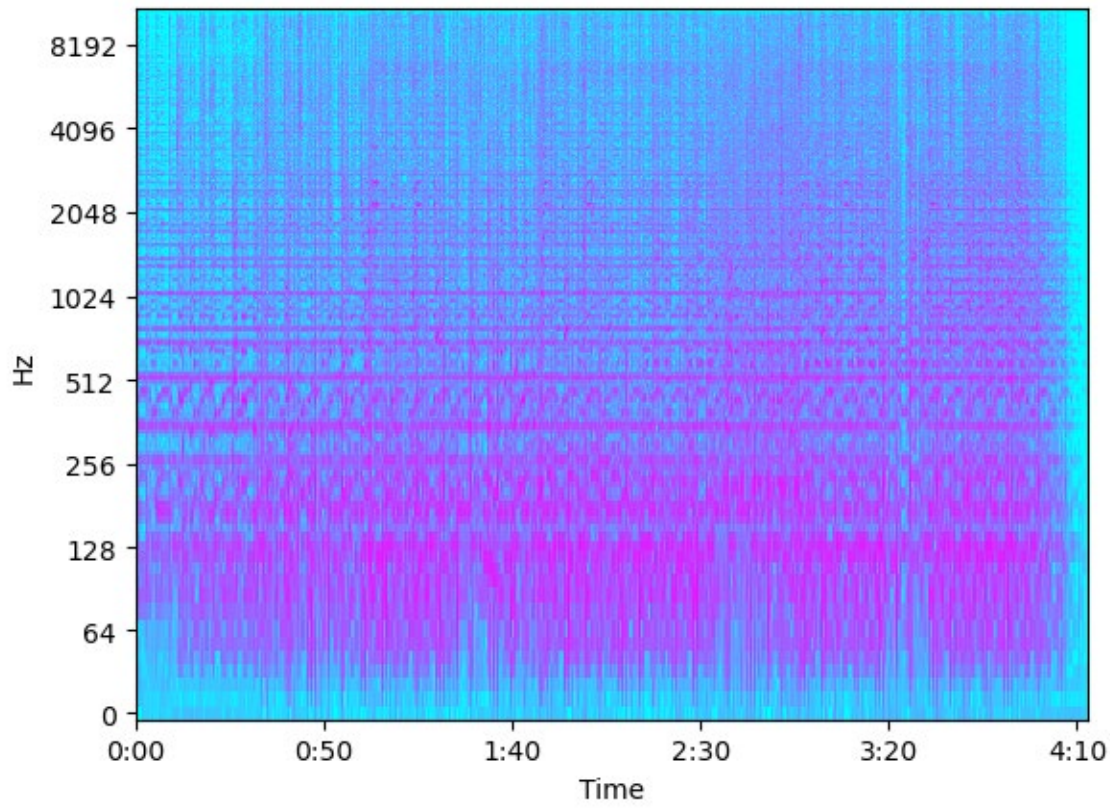
```
<Figure size 1600x600 with 0 Axes>
```
<Figure size 1600x600 with 0 Axes>

In [ ]:

```
librosa.display.specshow(DB, sr = sr, hop_length = hop_length, x_axis = 'time', y_axis = 'log', cmap = 'cool')
```

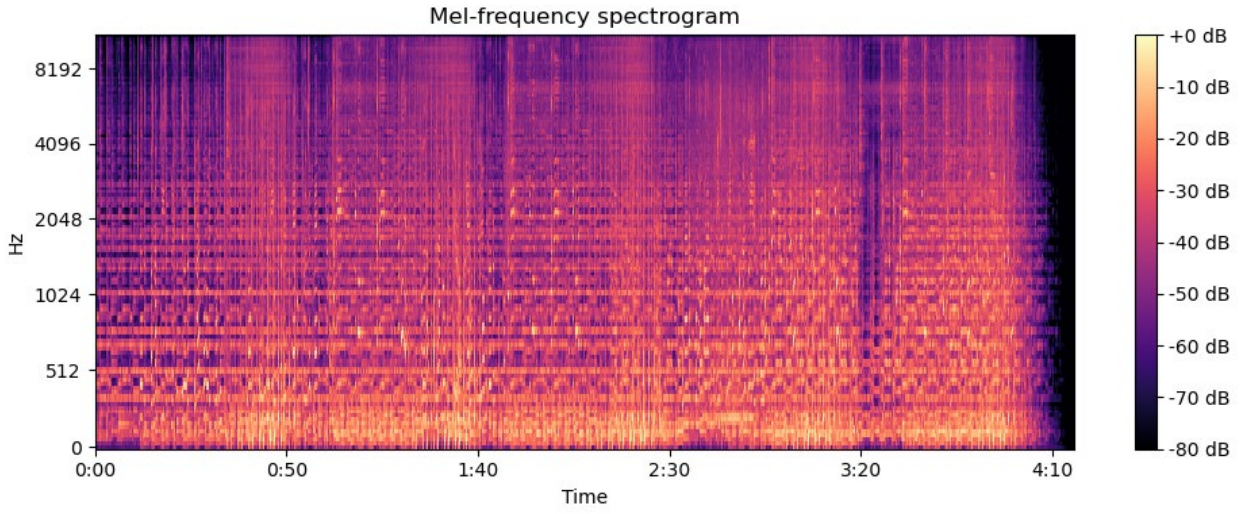
Out[ ]:

```
<matplotlib.collections.QuadMesh at 0x2d5a2e73a50>
```



In [ ]:

```
# Load audio file
y, sr = librosa.load('14 Free Fallin.wav')

# Compute the Mel-scaled spectrogram
S = librosa.feature.melspectrogram(y=y, sr=sr)

# Convert the amplitude to decibels
S_DB = librosa.power_to_db(S, ref=np.max)

# Plot the spectrogram
plt.figure(figsize=(10, 4))
mappable = librosa.display.specshow(S_DB, sr=sr, x_axis='time', y_axis='mel')

# Create colorbar
plt.colorbar(mappable, format='%+2.0f dB')
plt.title('Mel-frequency spectrogram')
plt.tight_layout()
plt.show()
```



In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 1)),  # Adjust input shape to match your data
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')  # Adjust the final layer based on the number of genres
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 1)),  # Adjust input shape to match your data
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')  # Adjust the final layer based on the number of genres
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [ ]:

```
print(model)
```

```
<Sequential name=sequential_2, built=True>
```

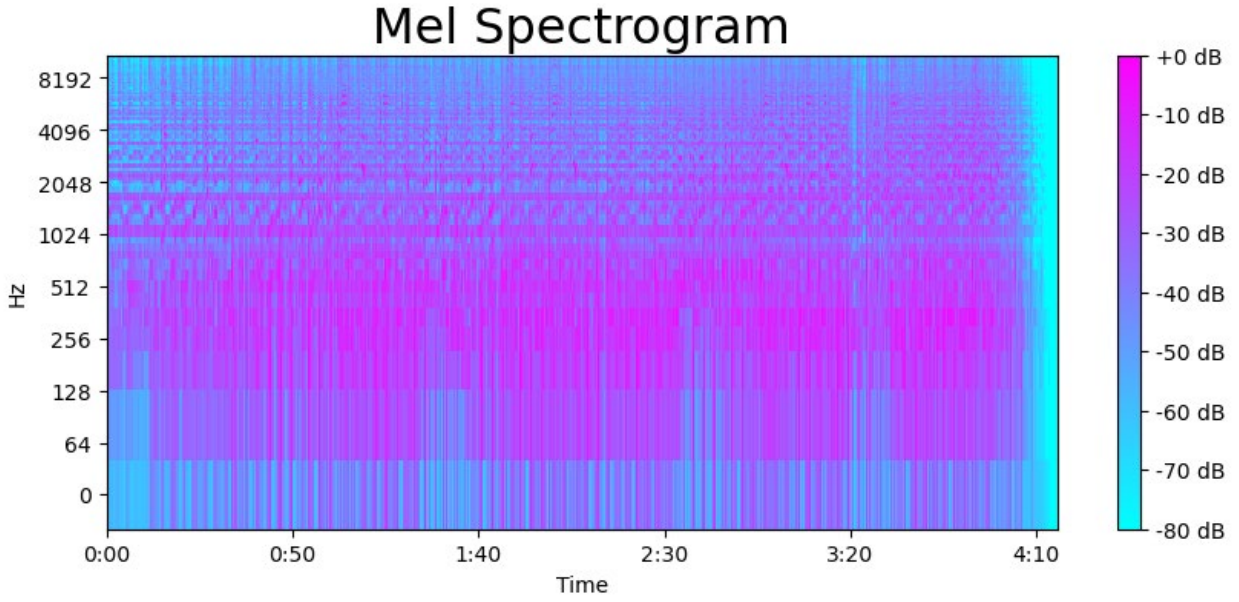
In [ ]:

```
y, sr = librosa.load('14 Free Fallin.wav')
y, _ = librosa.effects.trim(y)
```

In [ ]:

```
import librosa
```

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

# Load an audio file
y, sr = librosa.load('14 Free Fallin.wav')

# Compute Mel spectrogram using keyword arguments explicitly
S = librosa.feature.melspectrogram(y=y, sr=sr)  # Notice y=y and sr=sr are now explicitly named

# Convert to log scale (dB)
S_DB = librosa.power_to_db(S, ref=np.max)

# Display the spectrogram
plt.figure(figsize=(10, 4))
librosa.display.specshow(S_DB, sr=sr, hop_length=512, x_axis='time', y_axis='log', cmap='cool')
plt.colorbar(format='%+2.0f dB')
plt.title("Mel Spectrogram", fontsize=23)
plt.show()
```



In [ ]:

```
y_harm, y_perc = librosa.effects.hpss(audio_file)

plt.figure(figsize = (16, 6))
```

Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
<Figure size 1600x600 with 0 Axes>
```

In [ ]:

```
# Total zero_crossings in our 1 song
zero_crossings = librosa.zero_crossings(audio_file, pad=False)
print(sum(zero_crossings))
```

```
594987
```

In [ ]:

```
import scipy.signal

# Create a Hann window
```

```
window = scipy.signal.windows.hann(5)
```

In [ ]:

```
import numpy as np
import scipy.signal.windows

# Creating a Hann window
window_length = 512  # Size of the window
window = scipy.signal.windows.hann(window_length)
```

In [ ]:

```
# def patched_beat_track(y, sr):
#     S, phase = librosa.magphase(librosa.stft(y))
#     onset_env = librosa.onset.onset_strength(S=S)
#     tempo, beats = librosa.beat.beat_track(onset_envelope=onset_env, sr=sr)
#     return tempo

# # Using the patched function
# tempo = patched_beat_track(y=y, sr=sr)
# print("Detected tempo:", tempo)
```

In [ ]:

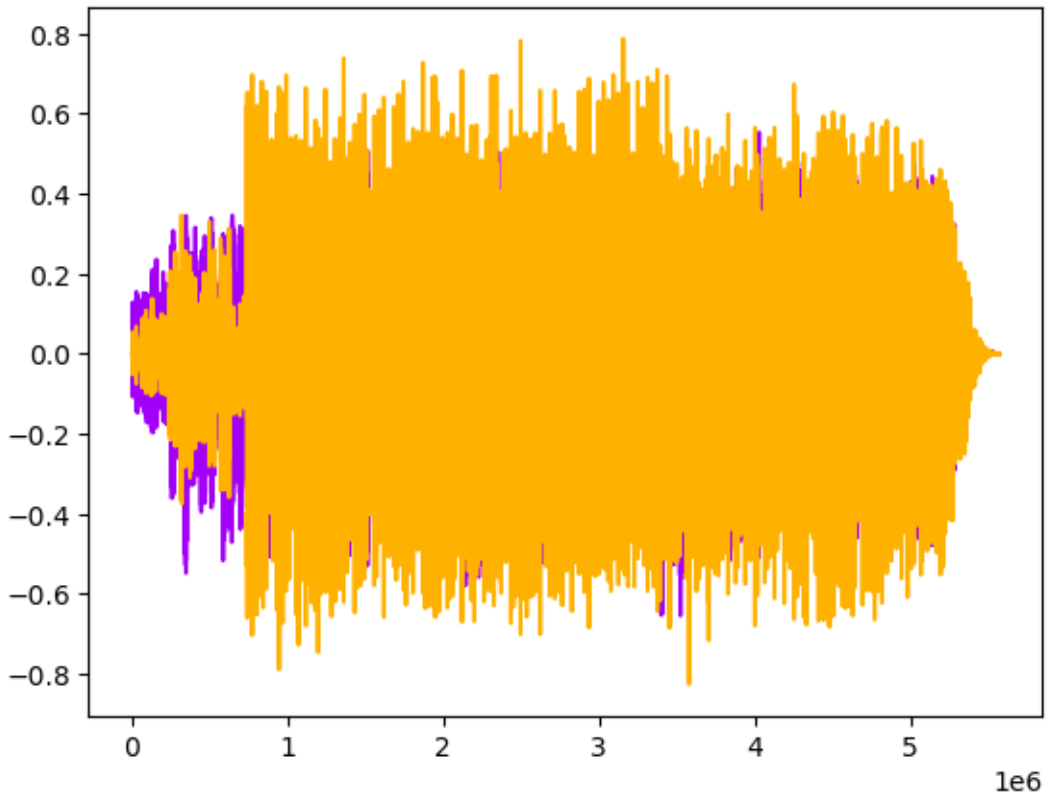
```
y_harm, y_perc = librosa.effects.hpss(audio_file)

plt.figure(figsize = (16, 6))
```

Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
<Figure size 1600x600 with 0 Axes>
```

In [ ]:

```
plt.plot(y_harm, color = '#A300F9');
plt.plot(y_perc, color = '#FFB100');
plt.show()
```



In [ ]:

```
# import librosa

# # Load audio data
# y, sr = librosa.load('14 Free Fallin.wav')

# # Correctly calling beat_track with keyword arguments
# tempo, _ = librosa.beat.beat_track(y=y, sr=sr)

# print("Detected tempo:", tempo)
```

Spectral Centroid

This variable represents brightness of a sound by calculating the center of sound spectrum (where the sound signal is at its peak). We can also plot it into a wave form.

In [ ]:

```
import librosa

# Load audio data
y, sr = librosa.load('14 Free Fallin.wav')
# Calculate the Spectral Centroids using keyword arguments
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]

# Output the shape and data
print('Centroids:', spectral_centroids)
Centroids: [0. 0. 0. ... 0. 0. 0.]
```

In [ ]:

```
print('Shape of Spectral Centroids:', spectral_centroids.shape, '\n')
```

```
Shape of Spectral Centroids: (11011,)
```

In [ ]:

```
frames = range(len(spectral_centroids))

# Converts frame counts to time (seconds)
t = librosa.frames_to_time(frames)

print('frames:', frames, '\n')
```

```
frames: range(0, 11011)
```

In [ ]:

```
print('t:', t)
```

```
t: [0.00000000e+00 2.32199546e-02 4.64399093e-02 ... 2.55605261e+02
 2.55628481e+02 2.55651701e+02]
```

In [ ]:

```
def normalize(x, axis=0):
   return sklearn.preprocessing.minmax_scale(x, axis=axis)
```

In [ ]:

```
#Plotting the Spectral Centroid along the waveform
plt.figure(figsize = (16, 6))
```

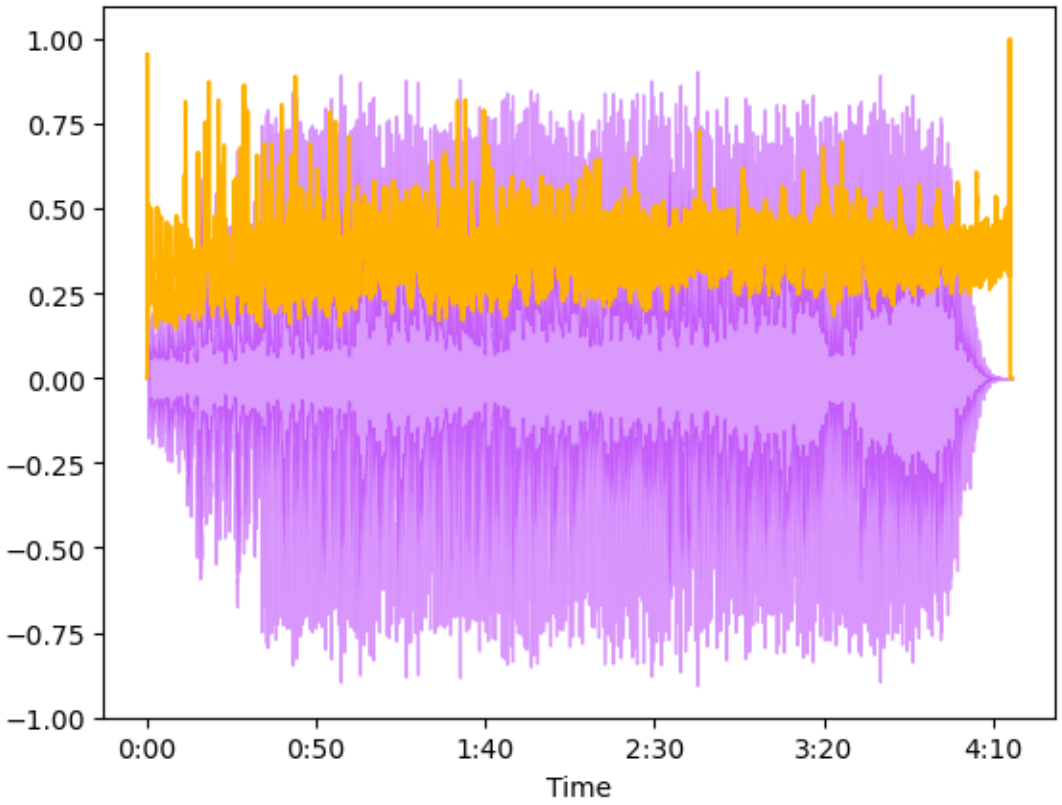
Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
```

<Figure size 1600x600 with 0 Axes>

In [ ]:

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
from sklearn.preprocessing import minmax_scale  # Import the specific function

# Load your audio file (replace 'path_to_your_audio_file.wav' with your file's path)
y, sr = librosa.load('14 Free Fallin.wav')

# Calculate spectral centroids
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]

# Time variable for plotting
t = librosa.frames_to_time(range(len(spectral_centroids)), sr=sr)

# Normalize the spectral centroids for plotting
normalized_centroids = minmax_scale(spectral_centroids, axis=0)

# Plotting the waveform and the normalized spectral centroids
librosa.display.waveshow(y, sr=sr, alpha=0.4, color='#A300F9')
plt.plot(t, normalized_centroids, color='#FFB100')
plt.show()
```

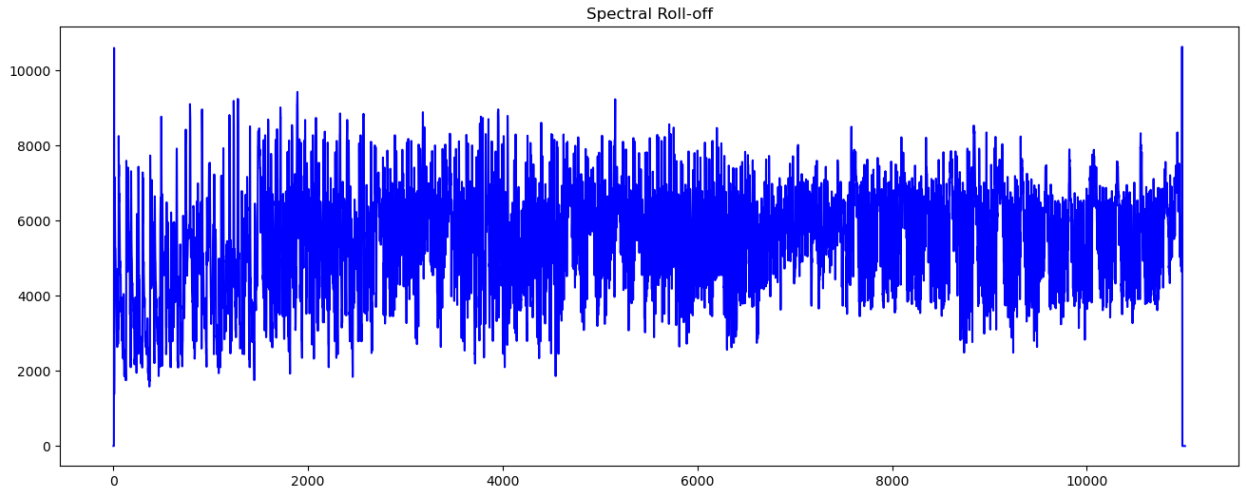


Spectral Rolloff

Spectral Rolloff is a frequency below a specified percentage of the total spectral energy. It is like we have a cut-point, and we visualize the sound wave that is below that cut-point. Let's just call it as another characteristic of a sound.

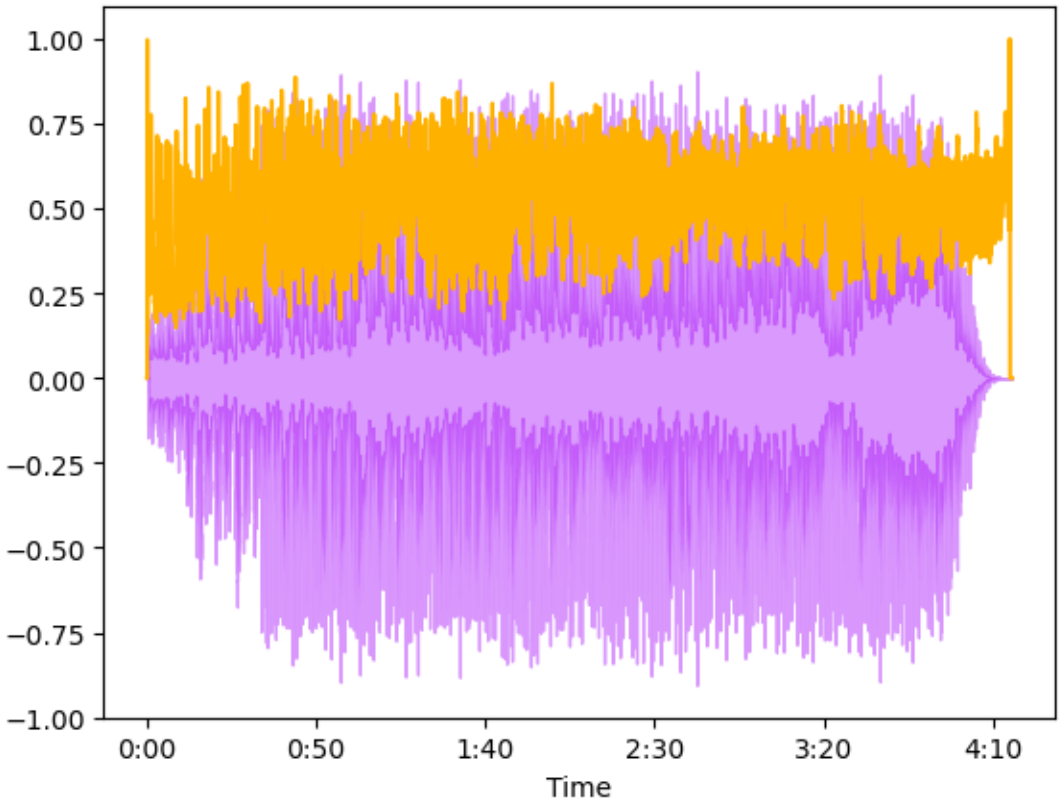
In [ ]:

```
import librosa
import matplotlib.pyplot as plt

# Assuming 'audio_file' is your audio time series and 'sr' is the sample rate
# Make sure you've loaded your audio data correctly with librosa.load
y, sr = librosa.load('14 Free Fallin.wav')

# Calculate the Spectral Roll-off using keyword arguments
spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)[0]

# The plot
plt.figure(figsize=(16, 6))
plt.plot(spectral_rolloff, color='b')
plt.title('Spectral Roll-off')
plt.show()
```



In [ ]:

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
from sklearn.preprocessing import minmax_scale  # Importing the minmax_scale function

# Load your audio file
y, sr = librosa.load('14 Free Fallin.wav')

# Calculate spectral rolloff
spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)[0]

# Time variable for plotting, assuming the correct definition of 't'
t = librosa.frames_to_time(range(len(spectral_rolloff)), sr=sr)

# Define a normalization function that uses sklearn's minmax_scale
def normalize(x, axis=0):
    return minmax_scale(x, axis=axis)

# Normalize the spectral rolloff for plotting
normalized_rolloff = normalize(spectral_rolloff)

# Display the waveform and the normalized spectral rolloff
librosa.display.waveshow(y, sr=sr, alpha=0.4, color='#A300F9')
plt.plot(t, normalized_rolloff, color='#FFB100')
plt.show()
```



Mel-Frequency Cepstral Coefficients

The Mel frequency Cepstral coefficients (MFCCs) of a signal are a small set of features that describes the overall shape of a spectral envelope. It imitates characteristics of human voice.

In [ ]:

```
import librosa

# Assuming you have loaded your audio file 'audio_file' and have its sample rate 'sr'
y, sr = librosa.load('14 Free Fallin.wav')

# Calculate MFCCs using keyword arguments
mfccs = librosa.feature.mfcc(y=y, sr=sr)

# Print the shape of the MFCCs array to verify
print('MFCCs shape:', mfccs.shape)
```

MFCCs shape: (20, 11011)

In [ ]:

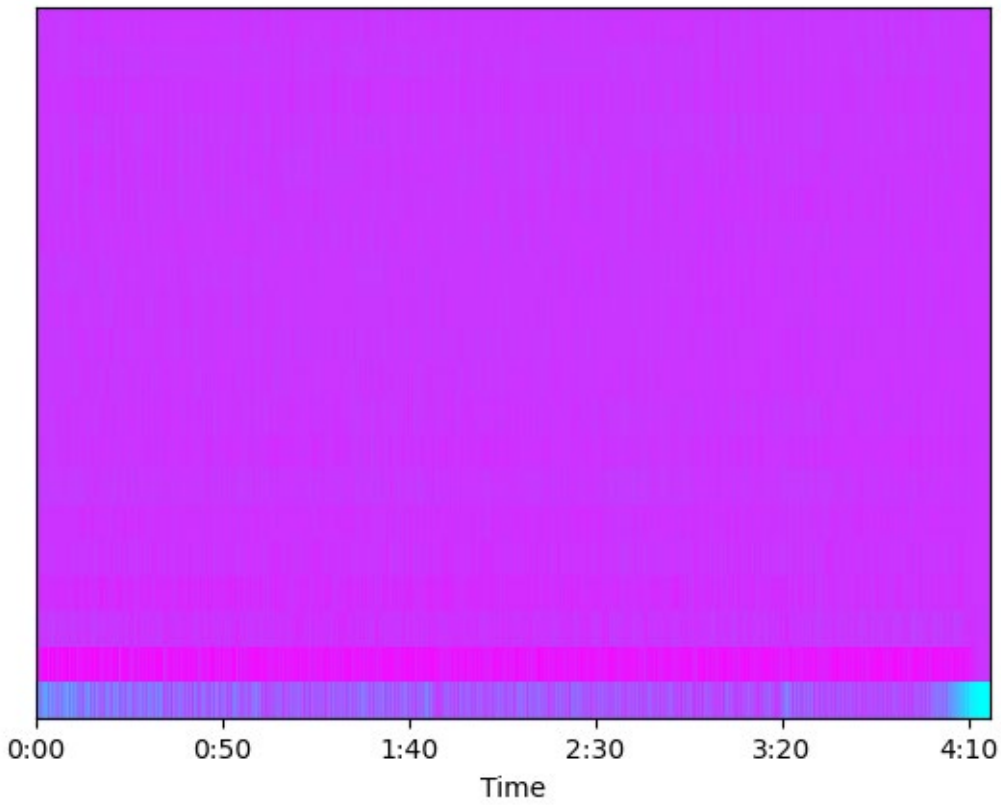
```
plt.figure(figsize = (16, 6))
```

Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
```
<Figure size 1600x600 with 0 Axes>

In [ ]:

```
librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
plt.show()
```



In [ ]:

```
from sklearn.preprocessing import scale
import librosa
from sklearn.preprocessing import scale  # Importing the scale function

# Assuming you've loaded your audio file and calculated mfccs
y, sr = librosa.load('14 Free Fallin.wav')
mfccs = librosa.feature.mfcc(y=y, sr=sr)

# Perform feature scaling
mfccs_scaled = scale(mfccs, axis=1)  # Scale MFCCs across the feature axis

# Print the shape of the scaled MFCCs array to verify
print('Scaled MFCCs shape:', mfccs_scaled.shape)
```

```
Scaled MFCCs shape: (20, 11011)
c:\ProgramData\Anaconda3\envs\ml\Lib\site-packages\sklearn\preprocessing\_data.py:261:
UserWarning:

Numerical issues were encountered when centering the data and might not be solved. Dataset may contain too large values. You may need to prescale your features.

c:\ProgramData\Anaconda3\envs\ml\Lib\site-packages\sklearn\preprocessing\_data.py:280:
UserWarning:

Numerical issues were encountered when scaling the data and might not be solved. The standard deviation of the data is probably very close to 0.
```

In [ ]:

```
print('Mean:', mfccs.mean(), '\n')
```
```
Mean: -1.8536327
```

In [ ]:

```
print('Var:', mfccs.var())
```
```
Var: 1683.6346
```

In [ ]:

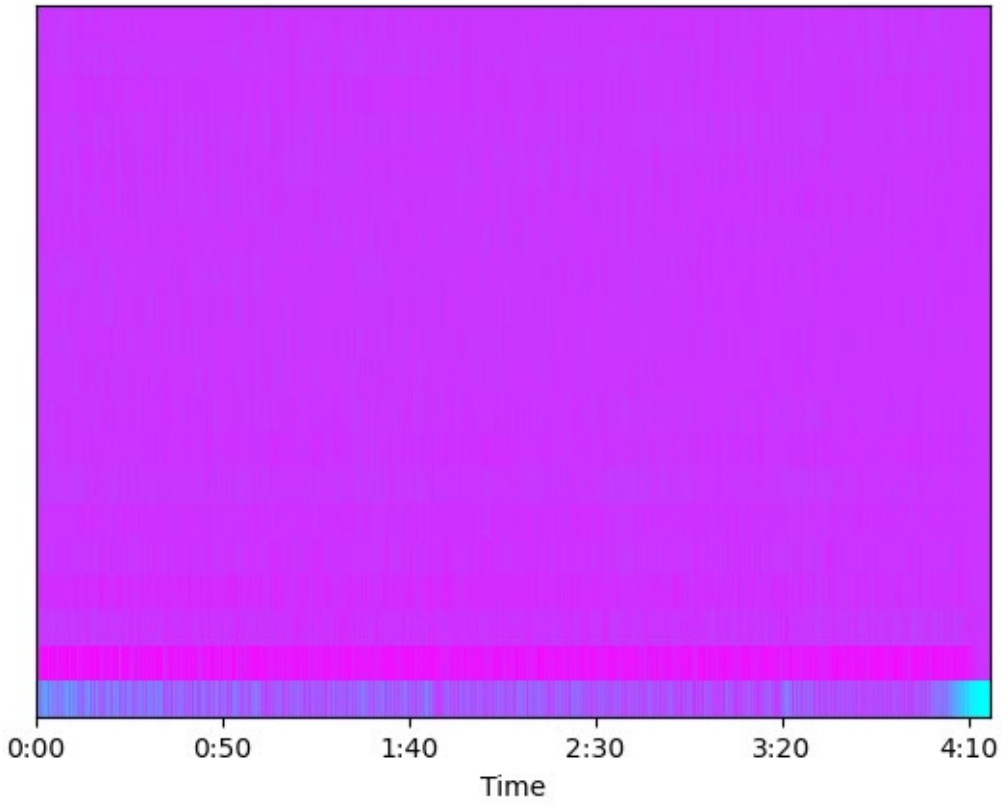
```
plt.figure(figsize = (16, 6))
```

Out[ ]:

```
<Figure size 1600x600 with 0 Axes>
```
<Figure size 1600x600 with 0 Axes>

In [ ]:

```
librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
plt.show()
```



Chroma Frequencies

Chroma feature represents the tone of music or sound by projecting its sound spectrum into a space that represents musical octave. This feature is usually used in chord recognition task.

In [ ]:

```
import librosa

# Assuming 'audio_file' contains the audio data loaded using librosa.load()
y, sr = librosa.load('14 Free Fallin.wav')
hop_length = 5000
```

```
# Calculate the Chromagram using keyword arguments
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=hop_length)

# Print the shape of the Chromagram to verify
print('Chromogram shape:', chromagram.shape)
```
Chromogram shape: (12, 1128)

In [ ]:

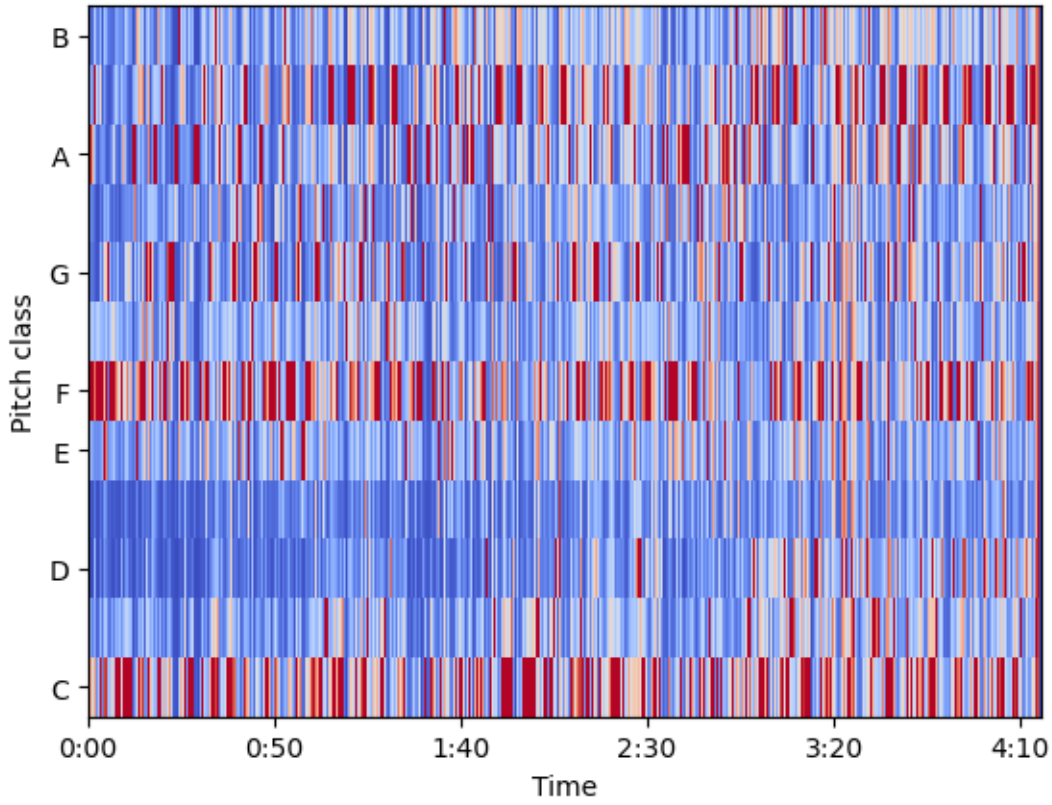
```
plt.figure(figsize=(16, 6))
```

Out[ ]:

<Figure size 1600x600 with 0 Axes>
<Figure size 1600x600 with 0 Axes>

In [ ]:

```
librosa.display.specshow(chromagram, x_axis='time', y_axis='chroma', hop_length=hop_length,
cmap='coolwarm');
plt.show()
```



Exploratory Data Analysis

We will perform an exploratory data analysis with the features_30_sec.csv data that contains the mean and variance of the features discussed above for all audio file in the data bank. We have 10 genres of music, each genre has 100 audio files. That makes the total of 1000 songs that we have. There are 60 features in total for each song.

In [ ]: