In [39]:
```
#pip install librosa
```

Designed specifically for Python and does not have official support for other programming languages. It's a Python library built on top of Python's NumPy, SciPy, and Matplotlib libraries, which means it relies heavily on Python's ecosystem of data science and audio processing tools.

In [1]:
```
#Loading example audio - Ashiel Mystery, A Detective Story, Ch. 2, LibriSpee
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import IPython.display as ipd

#y is an np.ndarray [shape=(n,) or (…, n)], audio time series. Multi-channel
# sr is a number > 0 [scalar], sampling rate of y

y, sr = librosa.load(librosa.ex('libri1'), duration=5.0)
```

# Spectral Features

## 1. Chroma STFT - Compute a chromagram from a waveform or power spectrogram.

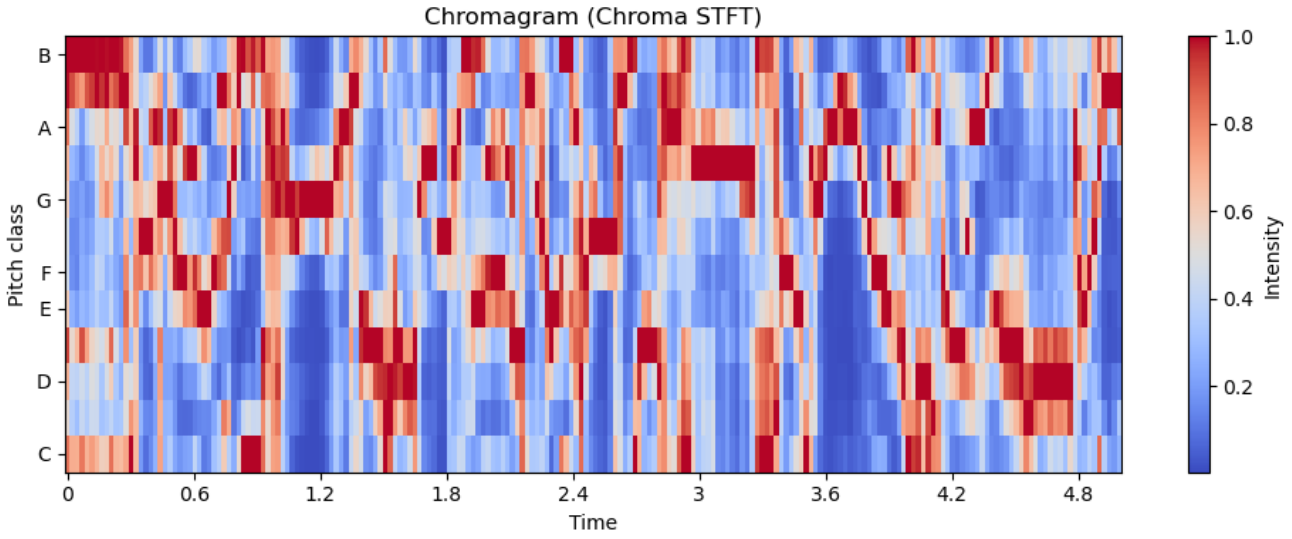
Chromagram that captures pitch and harmony by showing the energy of 12 pitch classes

The intensity (color brightness) of each row tells you how prominent that pitch class is at a specific time. Darker colors typically indicate weaker intensity, while brighter colors show strong prominence for a given pitch. For example, if a song is playing a G major chord (G, B, D) at a given time, you'll see brighter areas in those rows corresponding to G, B, and D.

Useful for chord recognition, detecting key changes, musical genre classification, analyzing song structures to detect choruses/bridges etc.

In [41]:
```
chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)

plt.figure(figsize=(10, 4))
librosa.display.specshow(chroma_stft, y_axis='chroma', x_axis='time', sr=sr,
plt.colorbar(label='Intensity')
```

```
plt.title('Chromagram (Chroma STFT)')
plt.tight_layout()
plt.show()
```

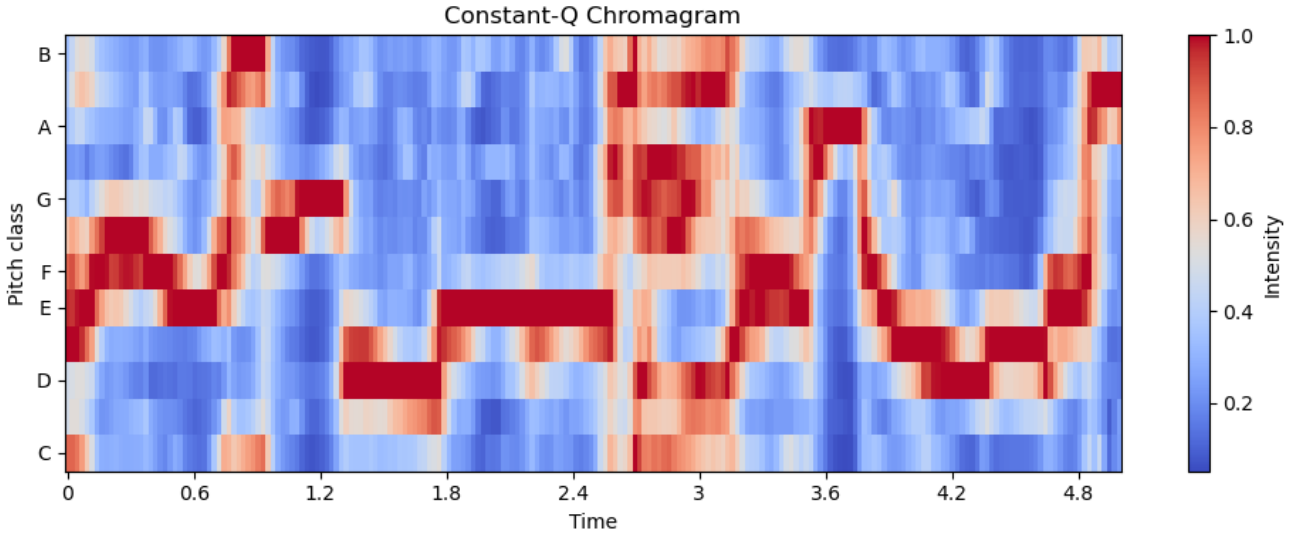



# 2. Chroma CQT - Constant-Q chromagram

Chromagram that captures pitch and harmony by showing the energy of 12 pitch classes similar to the previous feature chroma_stft but is useful because it provides a more musically relevant frequency scale than a standard STFT, as it uses logarithmically spaced frequency bins, which align better with how we perceive pitch.

Useful for chrord recognition, detecting key changes and pitch and harmony analysis

In [42]:
```
chroma_cqt = librosa.feature.chroma_cqt(y=y, sr=sr)

plt.figure(figsize=(10, 4))
librosa.display.specshow(chroma_cqt, y_axis='chroma', x_axis='time', sr=sr,
plt.colorbar(label='Intensity')
plt.title('Constant-Q Chromagram')
plt.tight_layout()
plt.show()

#The intensity (color brightness) of each row indicates the prominence of a
```

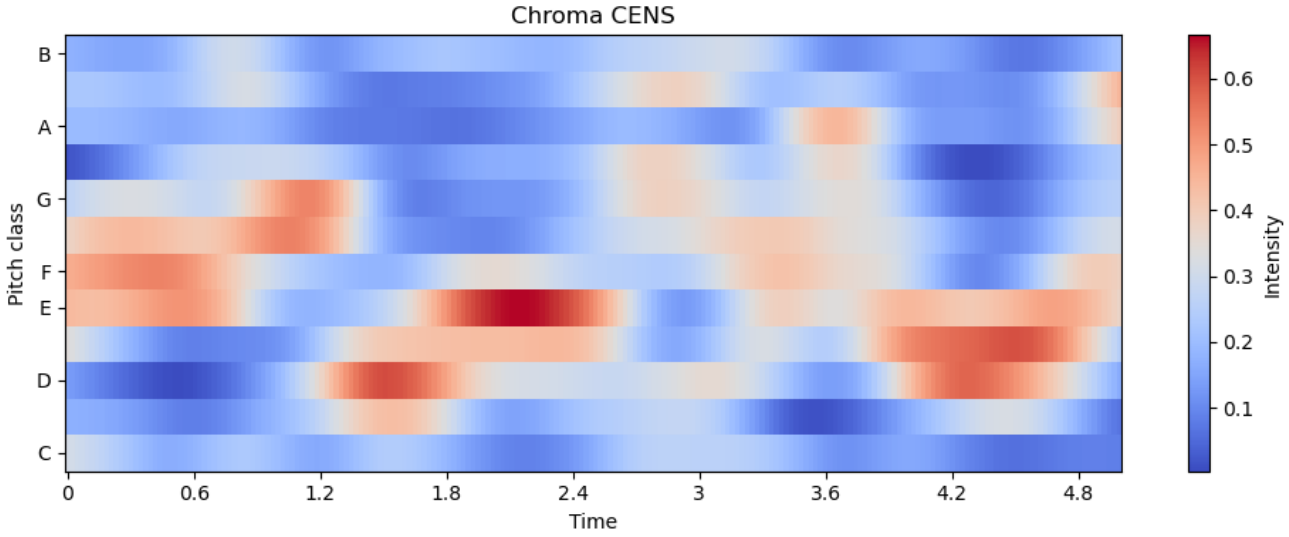



# 3. Chroma CENS - Compute the chroma variant "Chroma Energy Normalized" (CENS)

Chromagram that captures pitch and harmony by displaying the energy of 12 pitch classes. Different from the chroma features derived from STFT as CENS enhances musical relevance by using energy normalization and temporal smoothing, making it less sensitive to transient noise and variations in timing.

Useful for identifying similar sounding musical segments even with different dynamics, chord recognition, key change detection

In [43]:
```
chroma_cens = librosa.feature.chroma_cens(y=y, sr=sr)

plt.figure(figsize=(10, 4))
librosa.display.specshow(chroma_cens, y_axis='chroma', x_axis='time', sr=sr,
plt.colorbar(label='Intensity')
plt.title('Chroma CENS')
plt.tight_layout()
plt.show()

#The intensity (color brightness) of each row indicates the prominence of a
```

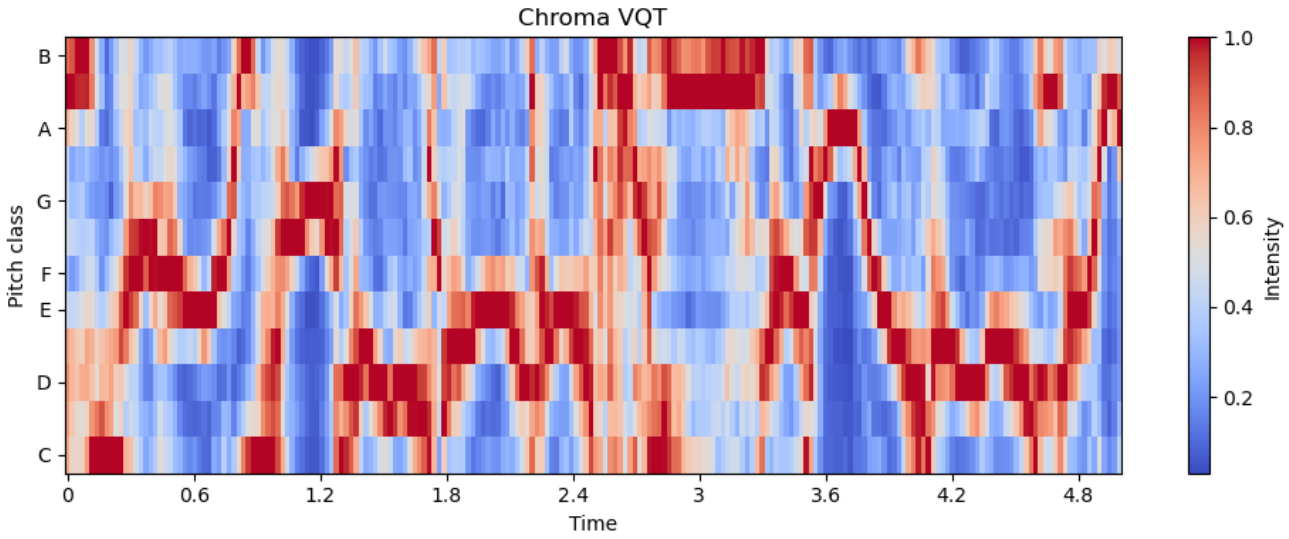



# 4. Chroma VQT - Compute the chroma variant using the Variable-Q Transform (VQT)

Chromagram that captures pitch and harmony by displaying the energy of 12 pitch classes. Unlike traditional chroma features derived from STFT, the VQT adapts to the local spectral characteristics of the audio signal, allowing for a more detailed representation of tonal content across varying frequencies.

Useful for chord recognition, key change detection, tonal analysis -particularly useful for analyzing music and other tonal signals

In [44]:
```
chroma_vqt = librosa.feature.chroma_vqt(y=y, sr=sr, intervals='equal')

plt.figure(figsize=(10, 4))
librosa.display.specshow(chroma_vqt, sr=sr, x_axis='time', y_axis='chroma',
plt.colorbar(label='Intensity')
plt.title('Chroma VQT')
plt.tight_layout()
plt.show()

#The intensity (color brightness) of each row indicates the prominence of a
```



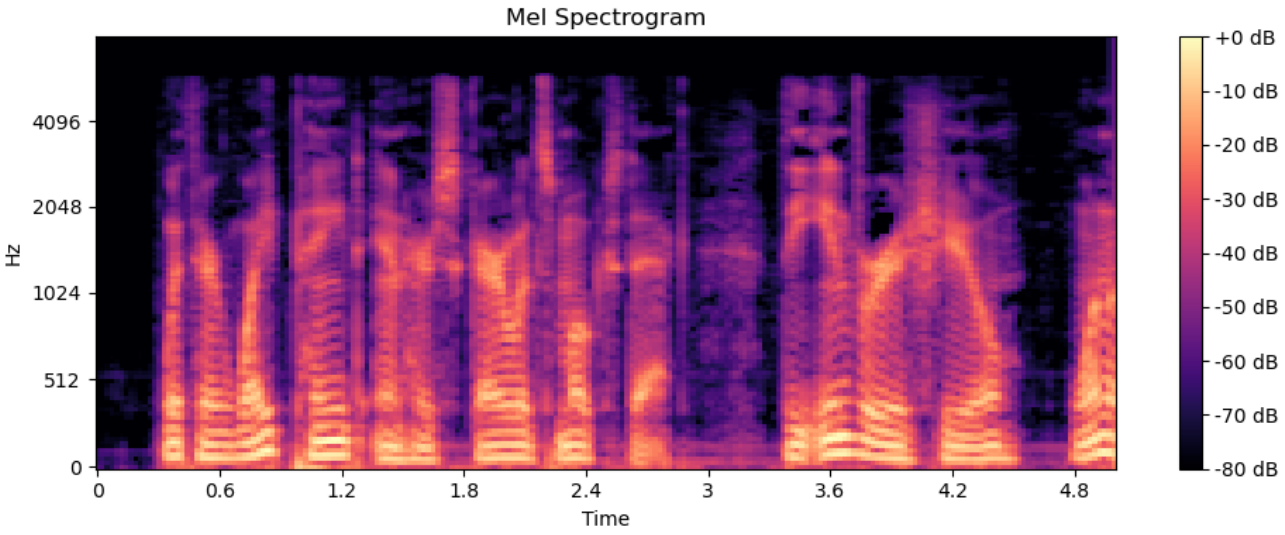

# 5. Mel Spectrogram

The Mel Spectrogram is a time-frequency representation that visualizes the energy distribution of audio signals across Mel frequency bins. This representation leverages the Mel scale, which approximates the way humans perceive pitch.

The intensity (color brightness) in the Mel spectrogram indicates the amplitude of each frequency bin at a given time. Darker colors typically represent lower energy levels, while brighter colors indicate higher energy, revealing the presence and strength of different frequencies throughout the audio. The negative dB values indicate that the measured intensity is below a defined reference level, typically the threshold of hearing.

Useful for speech and audio recognition, detecting pitches and timbres that align with human hearing.

In [45]:
```
melspec = librosa.feature.melspectrogram(y=y, sr=sr)
melspec_db = librosa.power_to_db(melspec, ref=np.max)

plt.figure(figsize=(10, 4))
librosa.display.specshow(melspec_db, sr=sr, x_axis='time', y_axis='mel', fma
plt.colorbar(format='%+2.0f dB')
plt.title('Mel Spectrogram')
plt.tight_layout()
plt.show()
```



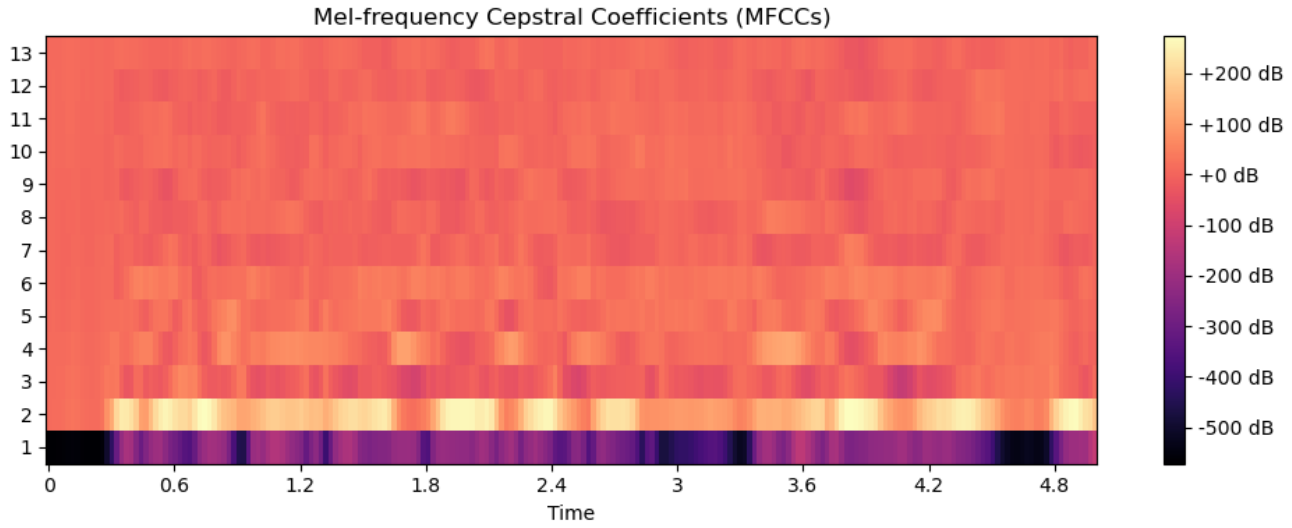

# 6. MFCC - Mel-frequency cepstral coefficients

MFCCs are a representation of the short-term power spectrum of sound, capturing the timbral aspects of audio signals. MFCCs are derived from the Mel spectrogram, transforming the frequency domain into a cepstral domain, which emphasizes the perceptually relevant characteristics of sounds.

The y-axis indicates the MFCC coefficients, with typically 13 coefficients used to represent the audio signal's key features. The color intensity indicates the amplitude of each coefficient at a specific time, where brighter colors represent stronger features and darker colors indicate weaker features.

Useful for Speech recognition, Speaker Identification, Emotion recognition, captures timbral characteristics that are unique to speech.

In [46]:
```
mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

plt.figure(figsize=(10, 4))
librosa.display.specshow(mfcc, sr=sr, x_axis='time', cmap='magma')
plt.yticks(np.arange(0, 13, step=1), labels=np.arange(1, 14))
plt.colorbar(format='%+2.0f dB')
plt.title('Mel-frequency Cepstral Coefficients (MFCCs)')
plt.tight_layout()
plt.show()
```



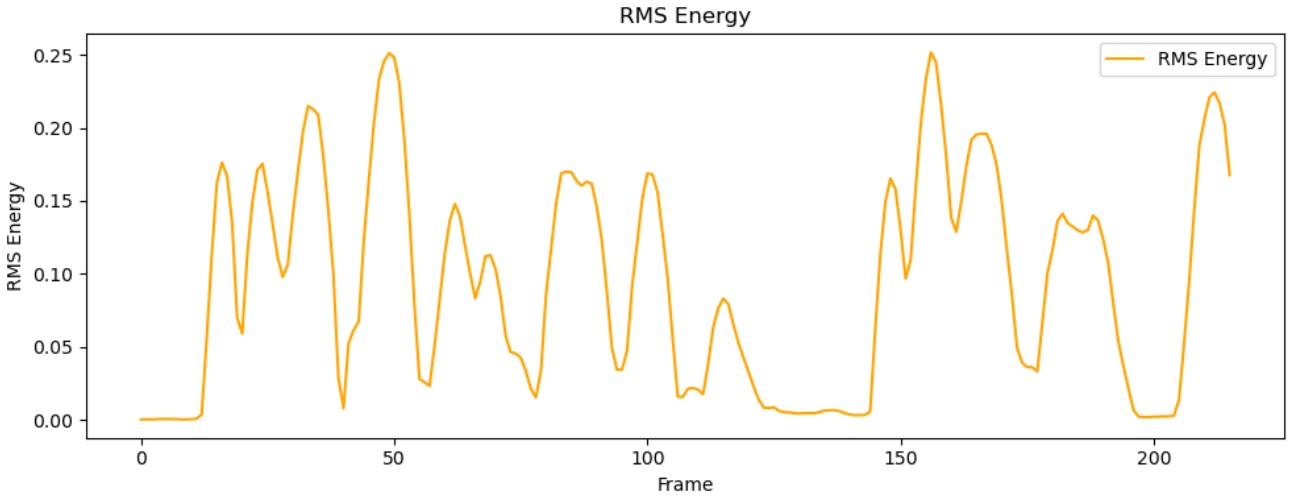

# 7. RMS Energy - Compute the Root Mean Square (RMS) energy of an audio signal

The RMS energy represents the average power of an audio signal over time, providing a measure of its loudness.

The x-axis represents time in frames, indicating different segments of the audio signal and the y-axis indicates the RMS energy level, which reflects the loudness of the audio at each time frame. Higher values indicate louder segments of audio, while lower values represent quieter segment

Useful for loudness estimation, segmentation, voice activity detection.

In [47]:
```
rms = librosa.feature.rms(y=y)

plt.figure(figsize=(10, 4))
plt.plot(rms[0], label='RMS Energy', color='orange')
plt.title('RMS Energy')
plt.xlabel('Frame')
plt.ylabel('RMS Energy')
plt.tight_layout()
plt.legend()
plt.show()
```



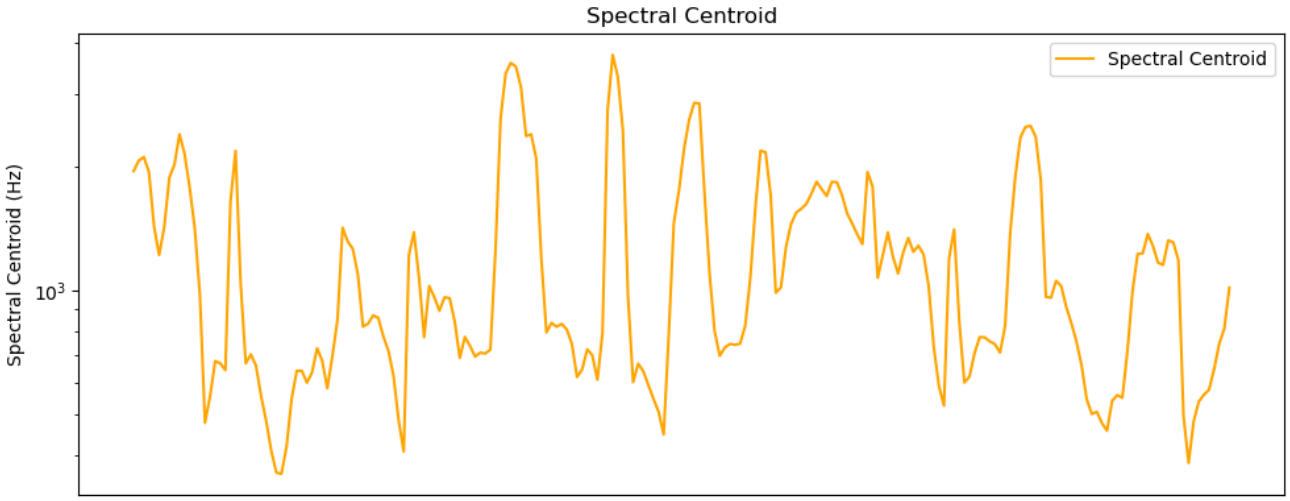

# 8. Spectral Centroid - Compute the spectral centroid of an audio signal

The spectral centroid indicates the "center of mass" of the spectrum, effectively representing the balance point of the spectral energy distribution. It provides insight into the brightness or sharpness of the sound. Higher spectral centroid values generally indicate a brighter sound, while lower values suggest a darker tone.

The x-axis represents time in frames, indicating different segments of the audio signal and the y-axis indicates the spectral centroid in Hertz, reflecting the frequency at which most of the spectral energy is concentrated.

Useful for perceived brightness of sound, identifying speech vs music, genre classification, instrument recognition

In [48]:
```
spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr)

plt.figure(figsize=(10, 4))
plt.semilogy(spectral_centroid[0], label='Spectral Centroid', color='orange'
plt.ylabel('Spectral Centroid (Hz)')
plt.xticks([])
plt.title('Spectral Centroid')
plt.tight_layout()
plt.legend()
plt.show()
```



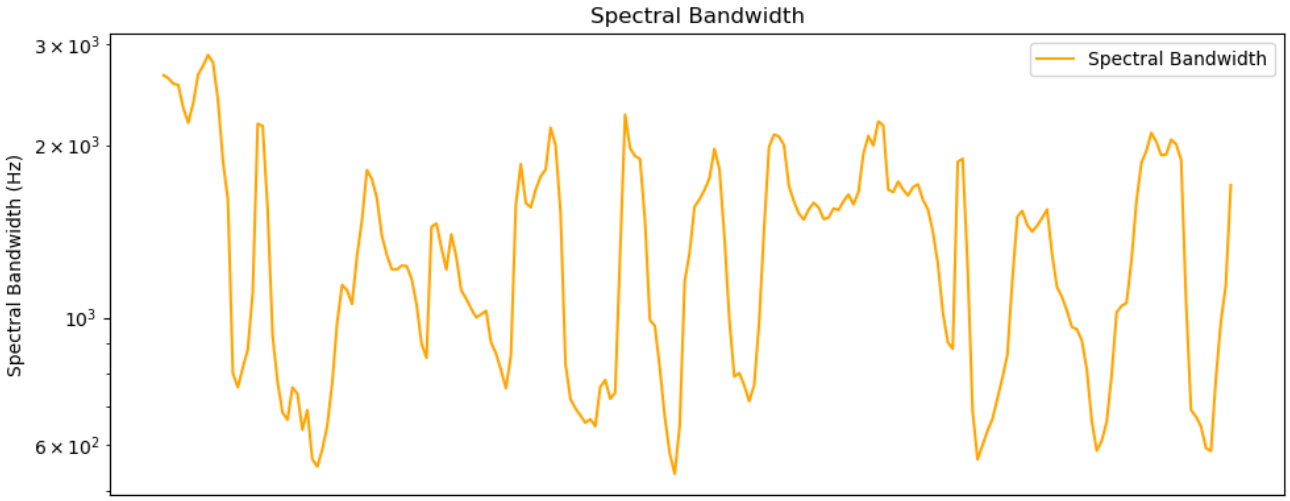

# 9. Spectral Bandwidth - Compute the spectral bandwidth of an audio signal

The spectral bandwidth measures the width of the spectrum around the spectral centroid, indicating how spread out the frequencies are in a given audio signal. This gives insights into the degree of tonal variation, where a higher spectral bandwidth suggests a richer and more complex sound texture, while a lower bandwidth indicates a more tonal and focused sound.

The x-axis represents time in frames, indicating different segments of the audio signal and the y-axis indicates the spectral bandwidth in Hertz, reflecting the extent of frequency dispersion around the spectral centroid.

Useful for describing the timbre of sounds, often used to classify different instruments or voices, audio quality evaluation (excessive bandwidth=noise)

In [49]:
```
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)

plt.figure(figsize=(10, 4))
plt.semilogy(spectral_bandwidth[0], label='Spectral Bandwidth', color='orang
plt.ylabel('Spectral Bandwidth (Hz)')
plt.xticks([])
plt.title('Spectral Bandwidth')
plt.tight_layout()
plt.legend()
plt.show()
```



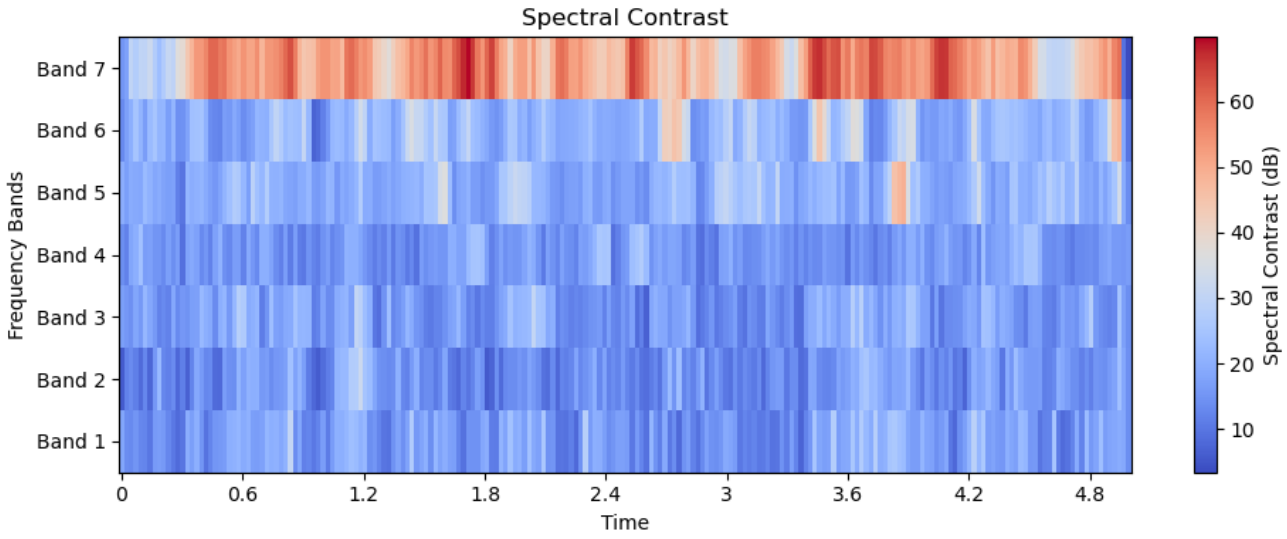

# 10. Spectral Contrast - Compute the spectral contrast of an audio signal

Spectral contrast measures the difference in amplitude between peaks and valleys in the sound spectrum, offering a representation of the timbral texture of an audio signal.

The x-axis represents time and the y-axis represents different frequency bands, while the color intensity indicates the level of spectral contrast measured in decibels.

Useful for genre classification, timbre analysis, audio event detection, classification of tonal vs noise audio

In [50]:
```
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)

plt.figure(figsize=(10, 4))
librosa.display.specshow(spectral_contrast, sr=sr, x_axis='time', cmap='cool
plt.colorbar(label='Spectral Contrast (dB)')
plt.title('Spectral Contrast')
num_bands = spectral_contrast.shape[0]
frequency_labels = [f'Band {i+1}' for i in range(num_bands)]
plt.yticks(ticks=np.arange(num_bands), labels=frequency_labels)
plt.ylabel('Frequency Bands')
plt.tight_layout()
plt.show()
```




# 11. Spectral Flatness - Compute the spectral flatness of an audio signal

Spectral flatness (or tonality coefficient) is a measure to quantify how much noise-like a sound is, as opposed to being tone-like. A high spectral flatness (closer to 1.0) indicates the spectrum is similar to white noise.

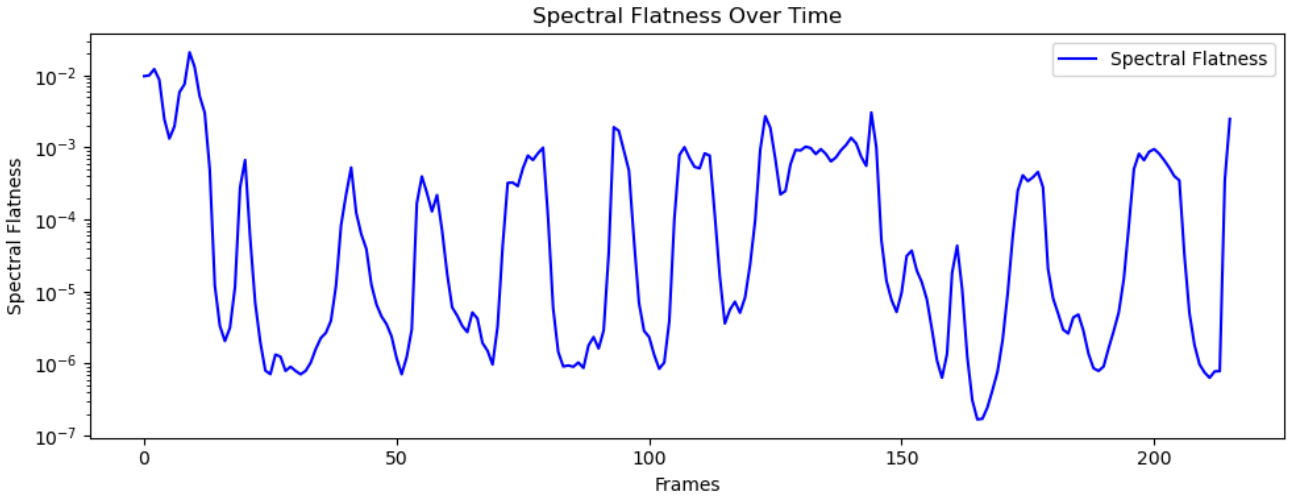
Higher spectral flatness values indicate a noisier spectrum (more aperiodic content), while lower values suggest a more tonal spectrum (more periodic content).

For example, distinguishing between harmonic sounds (like musical instruments) and noise-like sounds (like static or wind).

The x-axis represents time in frames and the y-axis indicates spectral flatness, reflecting the noisiness or tonal quality of the sound.

Useful for differentiating between tonal and noise-like sounds, audio quality assessment, environmental sound recognition (artificial vs real)

In [51]:
```
spectral_flatness = librosa.feature.spectral_flatness(y=y)

plt.figure(figsize=(10, 4))
plt.semilogy(spectral_flatness[0], label='Spectral Flatness', color='blue')
plt.ylabel('Spectral Flatness')
plt.title('Spectral Flatness Over Time')
plt.xlabel('Frames')
plt.tight_layout()
plt.legend()
plt.show()
```



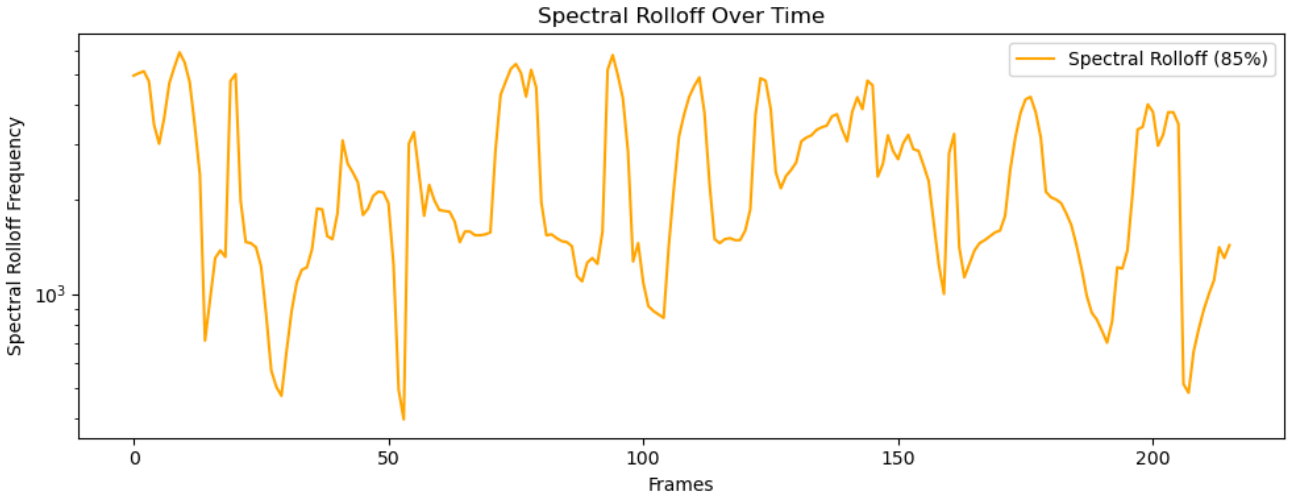

# 12. Spectral Rolloff - Compute the spectral rolloff of an audio signal

Spectral rolloff measures the frequency below which a specified percentage of the total spectral energy is contained. It is a useful feature for distinguishing between harmonic (tonal) and non-harmonic (noise-like) sounds. A lower rolloff frequency often indicates a more tonal sound, while a higher rolloff frequency suggests the presence of noise or complex sounds.

The x-axis represents time in frames and the y-axis indicates the spectral rolloff frequency, reflecting the cutoff point for the specified percentage of spectral energy.

Useful for distinguishing noise, genre classification, audio feature extraction

In [52]:
```
spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, roll_percent

plt.figure(figsize=(10, 4))
plt.semilogy(spectral_rolloff[0], label='Spectral Rolloff (85%)', color='ora
plt.ylabel('Spectral Rolloff Frequency')
plt.title('Spectral Rolloff Over Time')
plt.xlabel('Frames')
plt.tight_layout()
plt.legend()
plt.show()
```




# 13. Polynomial Features - Get coefficients of fitting an nth-order polynomial to the columns of a spectrogram

Polynomial features provide a way to represent audio data by capturing interactions and non-linear relationships between the frequencies in a spectrogram. Can extract complex patterns that can be useful for various audio analysis tasks.

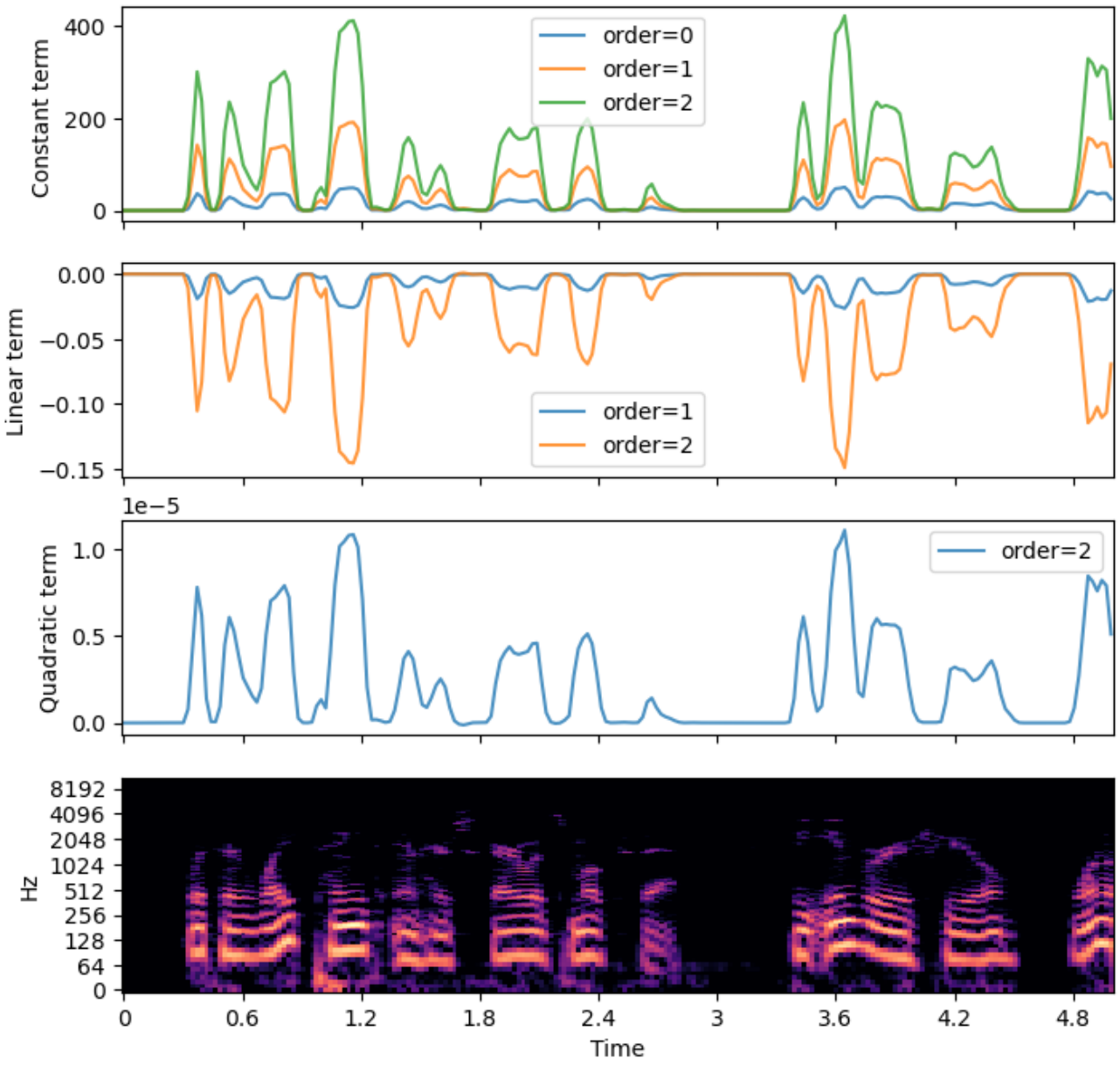
The x-axis represents time and the y-axis represents the polynomial feature values, indicating the intensity of the computed polynomial terms at different time frames.

Useful for capturing shape characteristics of spectral content, useful in speech and music analytics, feature engineering

In [53]:
```
D = librosa.stft(y) #Compute stft
S = np.abs(D)**2 #Convert to a power spectrogram
poly_features0 = librosa.feature.poly_features(S=S, order=0)
poly_features1 = librosa.feature.poly_features(S=S, order=1)
poly_features2 = librosa.feature.poly_features(S=S, order=2)

fig, ax = plt.subplots(nrows=4, sharex=True, figsize=(8, 8))
times = librosa.times_like(poly_features0)
ax[0].plot(times, poly_features0[0], label='order=0', alpha=0.8)
ax[0].plot(times, poly_features1[1], label='order=1', alpha=0.8)
ax[0].plot(times, poly_features2[2], label='order=2', alpha=0.8)
ax[0].legend()
ax[0].label_outer()
ax[0].set(ylabel='Constant term ')
ax[1].plot(times, poly_features1[0], label='order=1', alpha=0.8)
ax[1].plot(times, poly_features2[1], label='order=2', alpha=0.8)
ax[1].set(ylabel='Linear term')
```

```
ax[1].label_outer()
ax[1].legend()
ax[2].plot(times, poly_features2[0], label='order=2', alpha=0.8)
ax[2].set(ylabel='Quadratic term')
ax[2].legend()
librosa.display.specshow(librosa.amplitude_to_db(S, ref=np.max), y_axis='log
```

Out[53]: <matplotlib.collections.QuadMesh at 0x157c28f90>




# 14. Tonnetz - Computes the tonal centroid features

Link: https://en.wikipedia.org/wiki/Tonnetz

The tonnetz features represent the harmonic structure of a musical piece by mapping its

tonal centroids in a geometric space. It captures the relationships between different pitch classes and their harmonic relations, allowing for a visual representation of the tonal characteristics.

The function projects chroma features onto a 6-dimensional basis representing the perfect fifth, minor third, and major third each as two-dimensional coordinates

Useful for capturing harmonic and tonal characteristics, useful for chord recognition, genre classification

In [54]:
```
chroma = librosa.feature.chroma_stft(y=y, sr=sr)
tonnetz = librosa.feature.tonnetz(chroma=chroma)

tonnetz_df = pd.DataFrame(tonnetz.T, columns=[
    '1st Dimension (Major 3rd)',
    '2nd Dimension (Minor 3rd)',
    '3rd Dimension (Perfect 5th)',
    '4th Dimension (Tonality)',
    '5th Dimension (Harmonic Interval)',
    '6th Dimension (Tonal Cluster)'
])

print("Tonnetz Values (Sampled over Time):")
print(tonnetz_df.head())
```

```
Tonnetz Values (Sampled over Time):
   1st Dimension (Major 3rd)  2nd Dimension (Minor 3rd)  \
0                  -0.014303                   0.015269
1                  -0.034723                   0.010189
2                  -0.059316                   0.011999
3                  -0.101023                   0.026460
4                  -0.085225                   0.009279

   3rd Dimension (Perfect 5th)  4th Dimension (Tonality)  \
0                     0.078889                 -0.007698
1                     0.129418                 -0.041982
2                     0.195727                 -0.043660
3                     0.220098                 -0.030217
4                     0.167940                 -0.058201

   5th Dimension (Harmonic Interval)  6th Dimension (Tonal Cluster)
0                          -0.005199                       0.017926
1                          -0.010188                       0.006849
2                          -0.023046                       0.025103
3                          -0.013795                       0.035317
4                          -0.013648                       0.020599
```

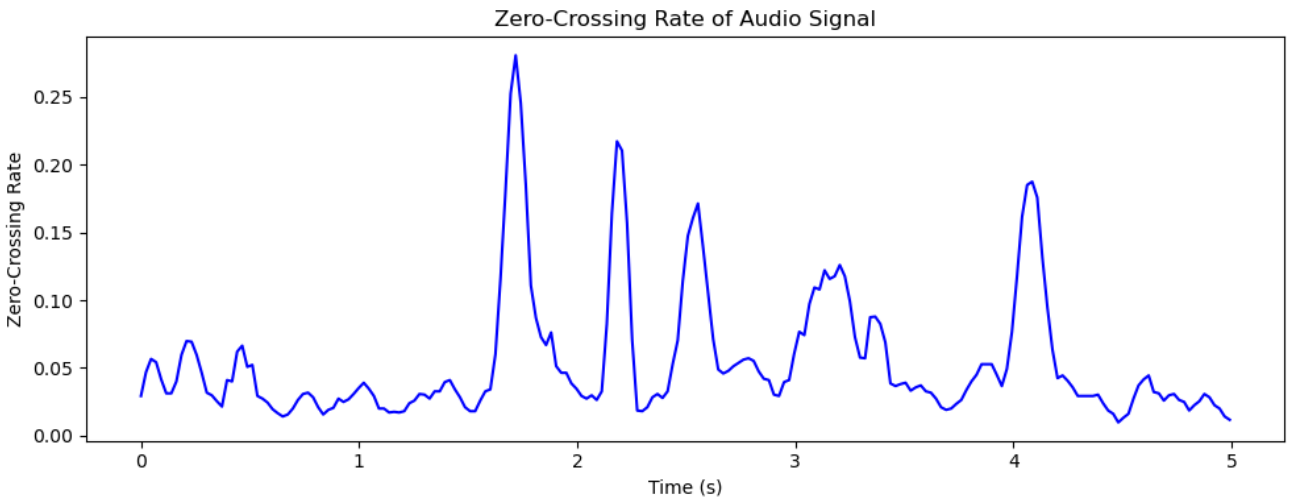
## 15. Zero-Crossing Rate - Compute the zero-crossing rate of an audio time series

ZCR indicates the rate of sign changes in the waveform, which corresponds to how frequently the signal crosses the zero amplitude line. This metric is especially helpful for analyzing sound textures, distinguishing percussive or noisy sounds from smoother, harmonic content.

The x-axis represents time and the y-axis represents the zero-crossing rate, which reflects the frequency of zero crossings within each frame.

Useful for Detecting voiced vs unvoiced parts of speech(higher zcr on unvoiced), common in speech segmentation, Noise Identification (higher zcr).

In [55]:
```
zcr = librosa.feature.zero_crossing_rate(y)

plt.figure(figsize=(10, 4))
plt.plot(librosa.times_like(zcr), zcr[0], color='b')
plt.title('Zero-Crossing Rate of Audio Signal')
plt.xlabel('Time (s)')
plt.ylabel('Zero-Crossing Rate')
plt.tight_layout()
plt.show()
```




# Rhythm Features

## 15. Tempo - Estimate the tempo (beats per minute)

This analysis measures the overall speed and rhythmic strength of the audio signal.

The estimated tempo represents the beats per minute (BPM) and reflects the speed of the audio.

Useful for Music analysis, determining beats-per-minute and rhythm detection, Genre classification.

In [56]:
```
tempo, _ = librosa.beat.beat_track(y=y, sr=sr)
print(f"Estimated Tempo: {tempo} BPM")
```

```
Estimated Tempo: [151.99908088] BPM
```

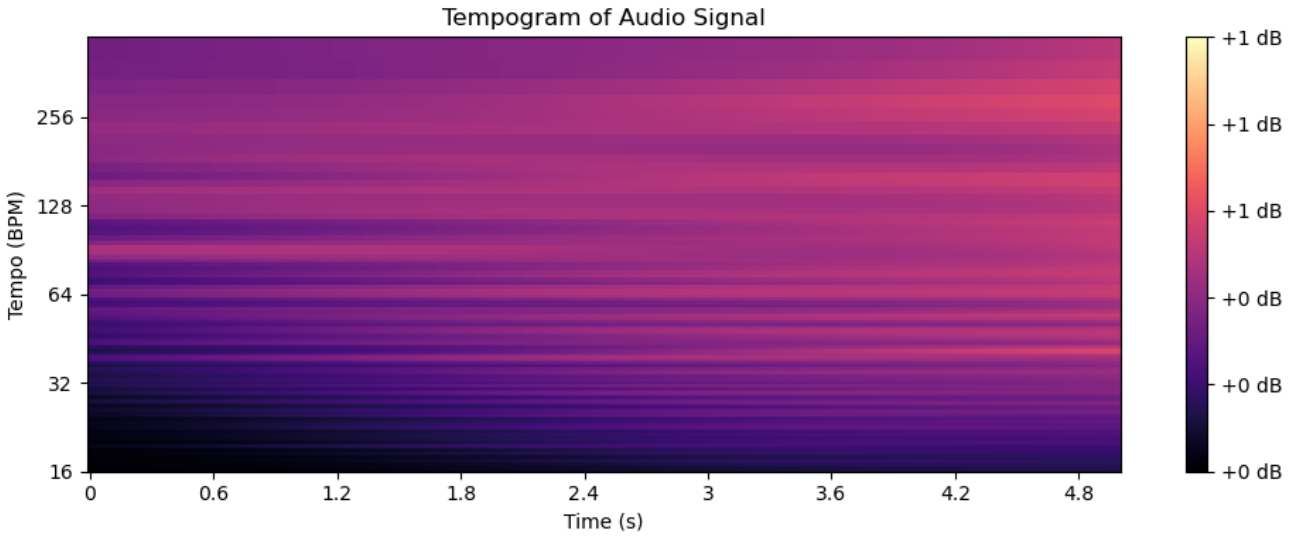
## 16. Tempogram - A representation of the tempo of an audio signal over time, capturing rhythmic changes and variations.

The tempogram displays the estimated tempo of the audio signal at each time frame, illustrating how the rhythmic intensity and speed may fluctuate throughout the piece. Axes: The x-axis represents time and the y-axis represents the tempo, showing the estimated beats per minute corresponding to the rhythmic activity at each moment.

The color intensity in the plot indicates the strength of the detected tempo at each time frame, with brighter colors typically representing stronger rhythmic activity

Useful for visualizing changes in tempo over time, Beat Tracking, Genre Classification

In [57]:
```
onset_env = librosa.onset.onset_strength(y=y, sr=sr)
tempogram = librosa.feature.tempogram(onset_envelope=onset_env, sr=sr)

plt.figure(figsize=(10, 4))
librosa.display.specshow(tempogram, sr=sr, x_axis='time', y_axis='tempo', cm
plt.colorbar(format='%+2.0f dB')
plt.title('Tempogram of Audio Signal')
plt.xlabel('Time (s)')
plt.ylabel('Tempo (BPM)')
plt.tight_layout()
plt.show()
```




# 17. Fourier Tempogram - A representation of the tempo of an audio signal over time using the Short-Time Fourier Transform (STFT)

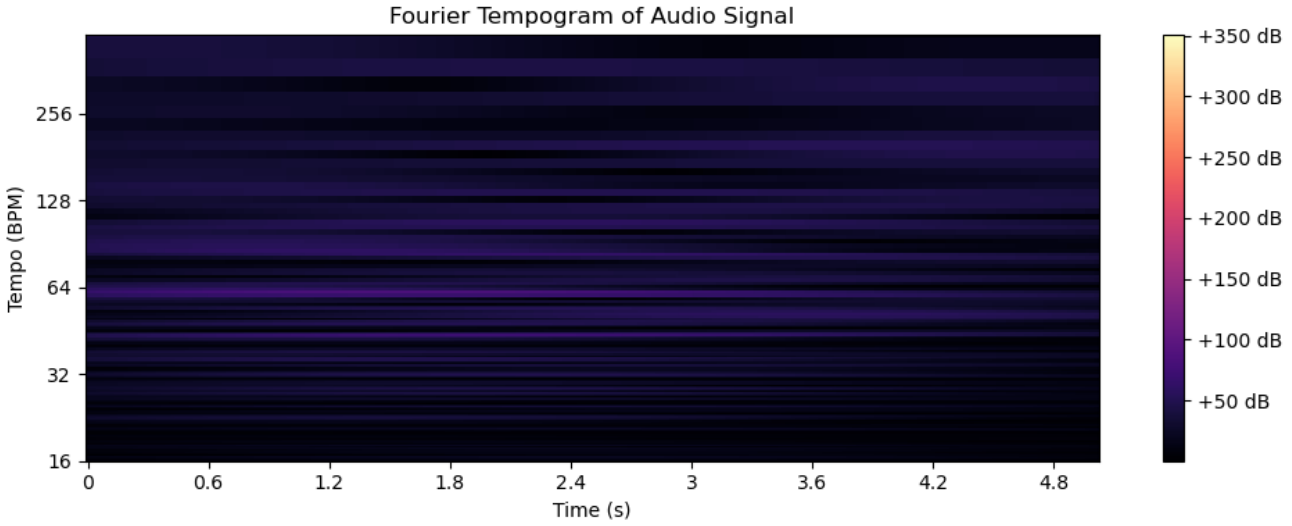
The Fourier tempogram displays the estimated tempo of the audio signal at each time frame, derived from the spectral content of the signal. This feature captures how the rhythmic intensity and speed vary throughout the piece, providing a more detailed analysis of tempo changes compared to standard tempograms.

Look for clearer and more consistent lines for stable rhythms, while irregular patterns suggest complex or variable tempo.

Useful for Tempo Variation Detection, Beat Tracking

In [58]:
```
fourier_tempogram = librosa.feature.fourier_tempogram(onset_envelope=onset_e

plt.figure(figsize=(10, 4))
librosa.display.specshow(fourier_tempogram, sr=sr, x_axis='time', y_axis='te
plt.colorbar(format='%+2.0f dB')
plt.title('Fourier Tempogram of Audio Signal')
plt.xlabel('Time (s)')
plt.ylabel('Tempo (BPM)')
plt.tight_layout()
plt.show()
```

```
/Users/sudeepthirebbalapalli/anaconda3/lib/python3.11/site-packages/librosa/
core/spectrum.py:266: UserWarning: n_fft=384 is too large for input signal o
f length=216
  warnings.warn(
/var/folders/vj/_93sqqfs3ddf9f18b8n51sfc0000gn/T/ipykernel_1266/77800296.py:
4: UserWarning: Trying to display complex-valued input. Showing magnitude in
stead.
  librosa.display.specshow(fourier_tempogram, sr=sr, x_axis='time', y_axis='
tempo', cmap='magma')
```




# 18. Tempogram Ratio

The tempogram ratio provides a comparative analysis of rhythmic energy distribution at different tempo levels, allowing for the identification of changes in tempo throughout the audio. This feature helps analyze sections with distinct rhythmic characteristics and shifts.

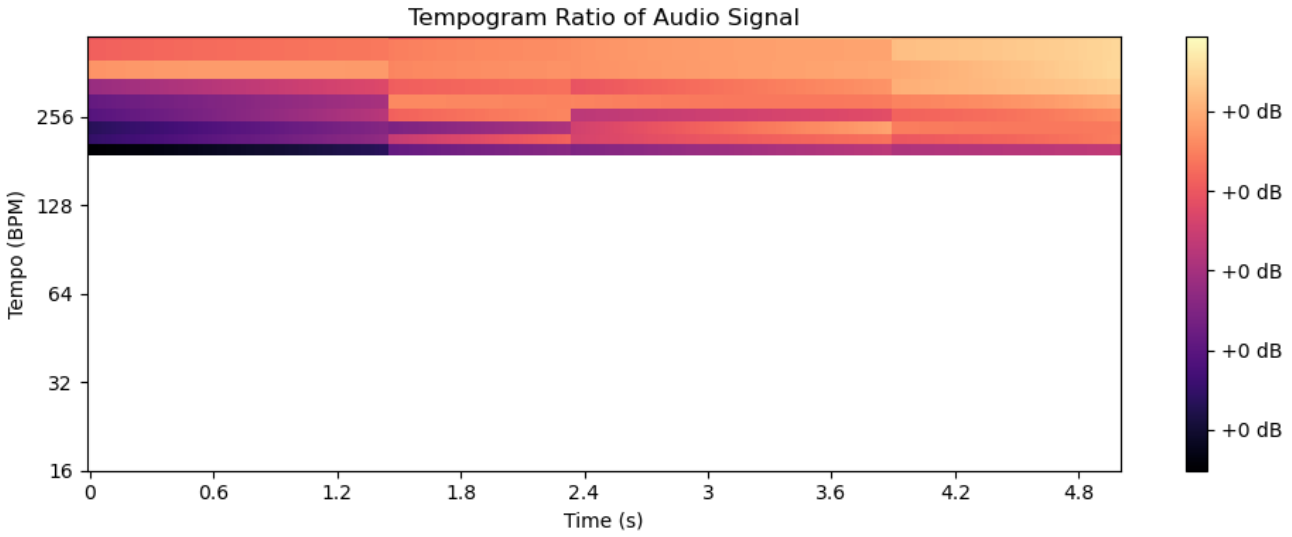
The color intensity in the plot indicates the strength of the detected tempo at each time frame, with brighter colors generally representing stronger rhythmic activity.

Brightness and density in specific areas can indicate regular tempo structures.

Useful for analyzing rhythmic structure through spectral patterns, Beat Tracking, Style Detection

In [59]:
```
tempogram_ratio = librosa.feature.tempogram_ratio(tg=tempogram, sr=sr)

plt.figure(figsize=(10, 4))
librosa.display.specshow(tempogram_ratio, sr=sr, x_axis='time', y_axis='temp
plt.colorbar(format='%+2.0f dB')
plt.title('Tempogram Ratio of Audio Signal')
```

```
plt.xlabel('Time (s)')
plt.ylabel('Tempo (BPM)')
plt.tight_layout()
plt.show()
```




# Feature Manipulation

## 19. Delta features - Computes the rate of change (delta) of a feature over time

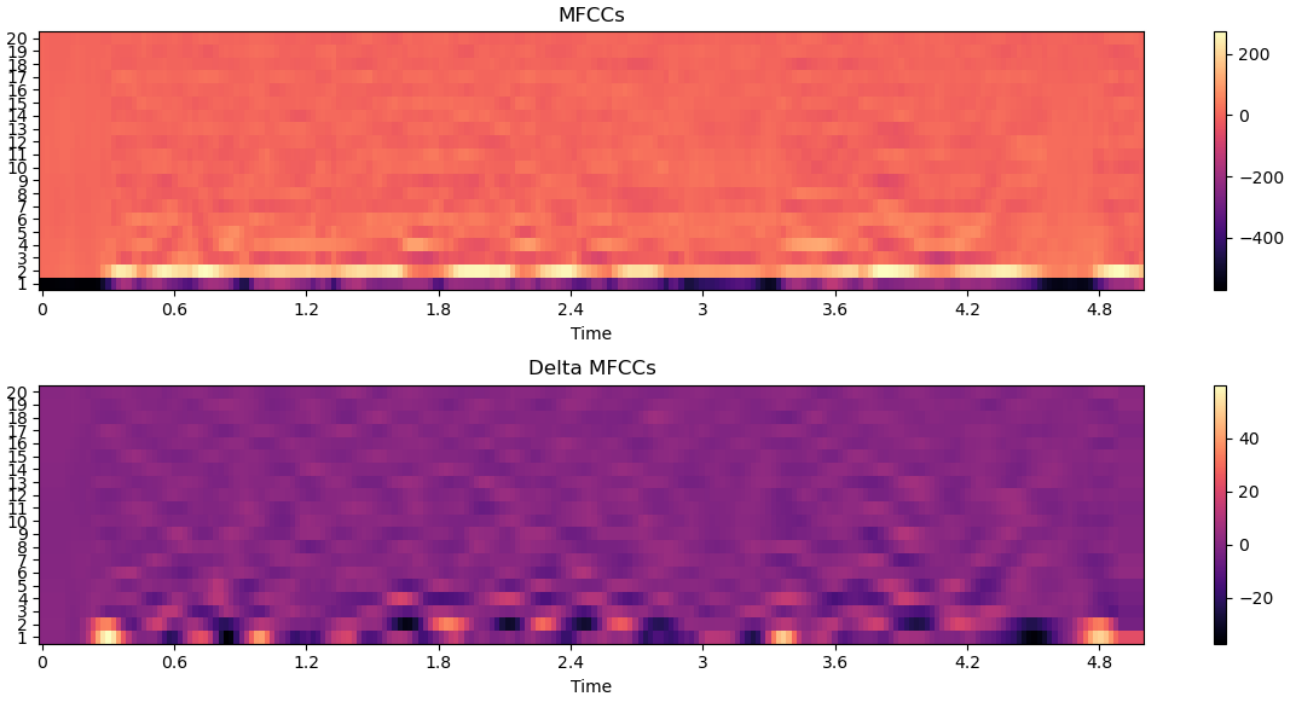
The delta features capture how the computed feature (like MFCC) changes over time. For example, if the MFCC values represent the spectral properties of an audio signal, the delta values show how quickly these properties are changing, indicating dynamic characteristics such as articulation or expressiveness in speech or music.

Observe areas with high variation for significant tonal or timbral changes.

Useful for capturing dynamics or changes in MFCCs over time, aiding in genre or speech emotion classification.

In [60]:
```
mfccs = librosa.feature.mfcc(y=y, sr=sr)
mfcc_delta = librosa.feature.delta(mfccs)

delta_mfccs = librosa.feature.delta(mfccs)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
librosa.display.specshow(mfccs, x_axis='time', sr=sr, cmap='magma')
```

```
plt.yticks(np.arange(0, 20, step=1), labels=np.arange(1, 21))
plt.colorbar()
plt.title('MFCCs')

plt.subplot(2, 1, 2)
librosa.display.specshow(delta_mfccs, x_axis='time', sr=sr, cmap='magma')
plt.yticks(np.arange(0, 20, step=1), labels=np.arange(1, 21))
plt.colorbar()
plt.title('Delta MFCCs')

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



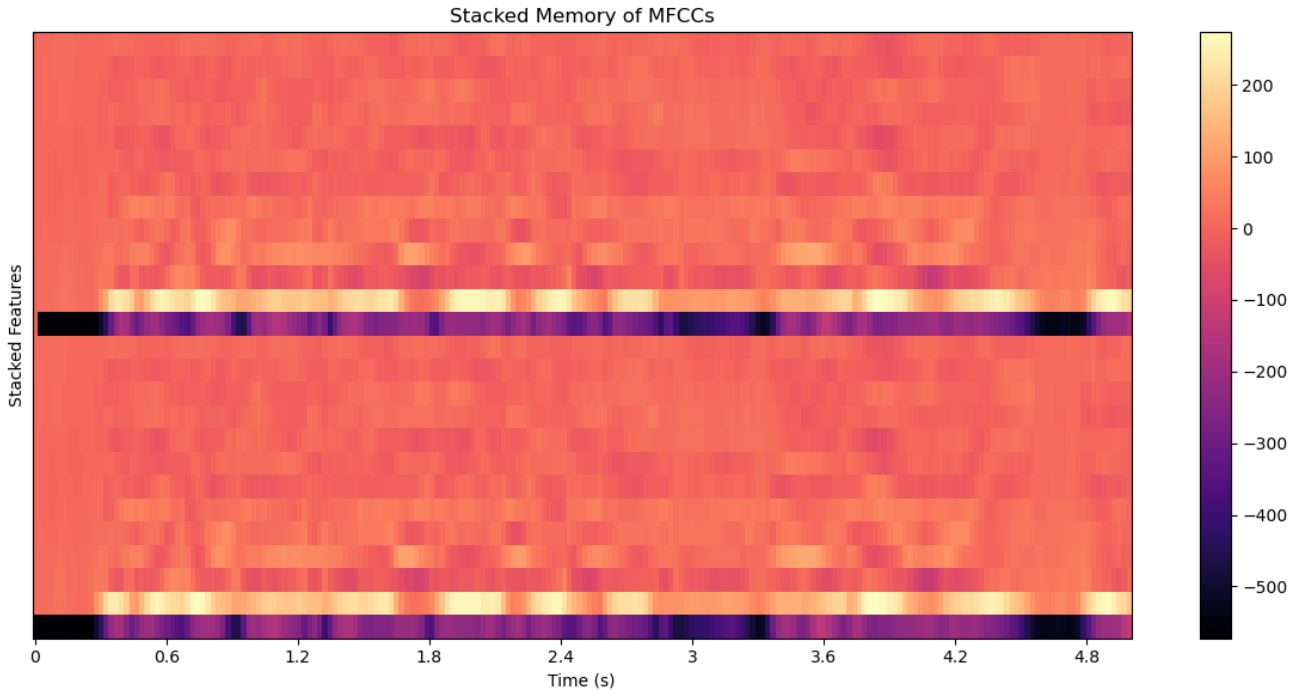

# 20. Stack Memory -Enables modeling of temporal dependencies by creating a history of past feature values

The output of stack memory effectively creates a new representation of the feature data where each column contains the original features as well as the additional stacked features. This gives a more comprehensive view of how the features evolve over time.

Look for trends and patterns in stacked features to analyze how audio evolves over time.

Useful for Temporal Context(account several frames of audio features), Machine Learning (stacked features can improve the performance of models)

In [61]:
```
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

stacked_mfccs = librosa.feature.stack_memory(mfccs, n_steps=2)

plt.figure(figsize=(12, 6))
librosa.display.specshow(stacked_mfccs, x_axis='time', sr=sr, cmap='magma')
plt.colorbar()
plt.title('Stacked Memory of MFCCs')
plt.xlabel('Time (s)')
plt.ylabel('Stacked Features')
plt.tight_layout()
plt.show()
#Plotting 2 step stacked features one above the other
```




# Feature Inversion

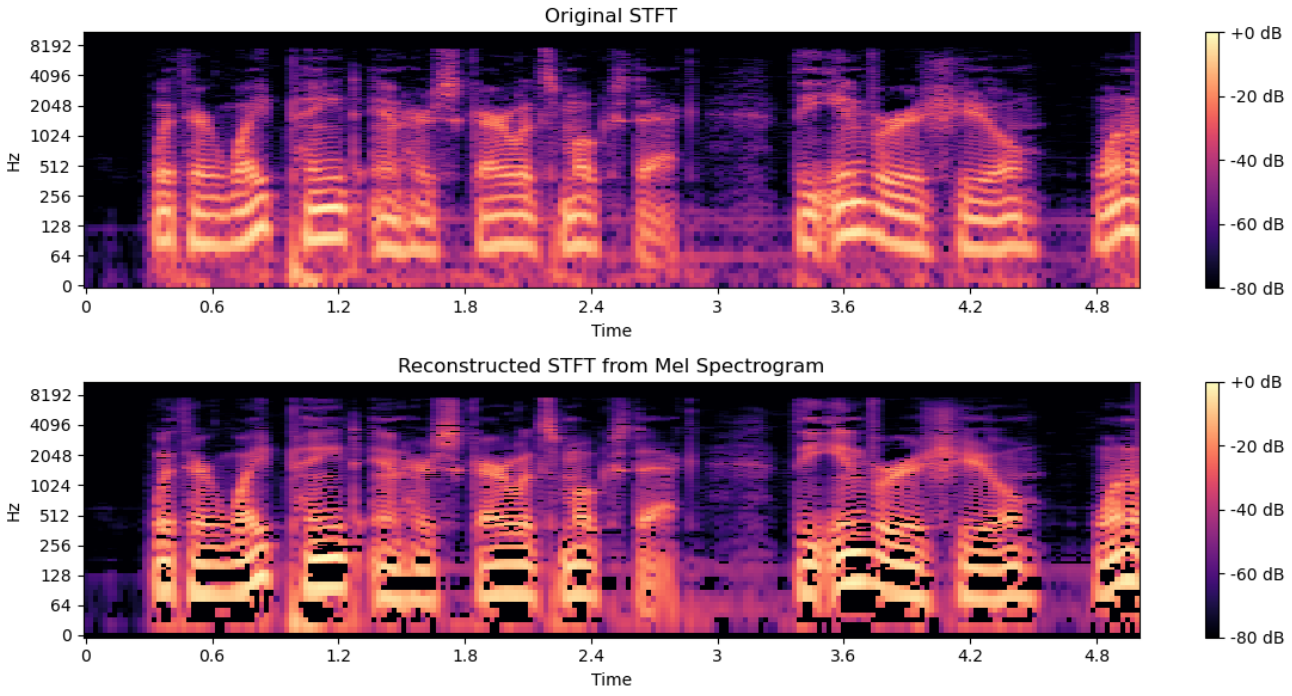
## 21. Inverse Mel to STFT - Approximate STFT magnitude from a Mel power spectrogram

The reconstructed STFT should ideally resemble the original STFT used to compute the Mel spectrogram, showing how well the Mel representation retains the frequency content of the original signal.

Useful Audio Reconstruction, Audio Manipulation, for reconstructing audio details from

the Mel spectrogram. Brightness and sharpness in reconstructed STFT indicate how well the inversion captures original signal characteristics.

In [64]:
```
stft = librosa.stft(y)
mel_spec = librosa.feature.melspectrogram(y=y, sr=sr)
stft_approx = librosa.feature.inverse.mel_to_stft(mel_spec)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
librosa.display.specshow(librosa.amplitude_to_db(np.abs(stft), ref=np.max),
plt.colorbar(format='%+2.0f dB')
plt.title('Original STFT')

plt.subplot(2, 1, 2)
librosa.display.specshow(librosa.amplitude_to_db(np.abs(stft_approx), ref=np
plt.colorbar(format='%+2.0f dB')
plt.title('Reconstructed STFT from Mel Spectrogram')

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



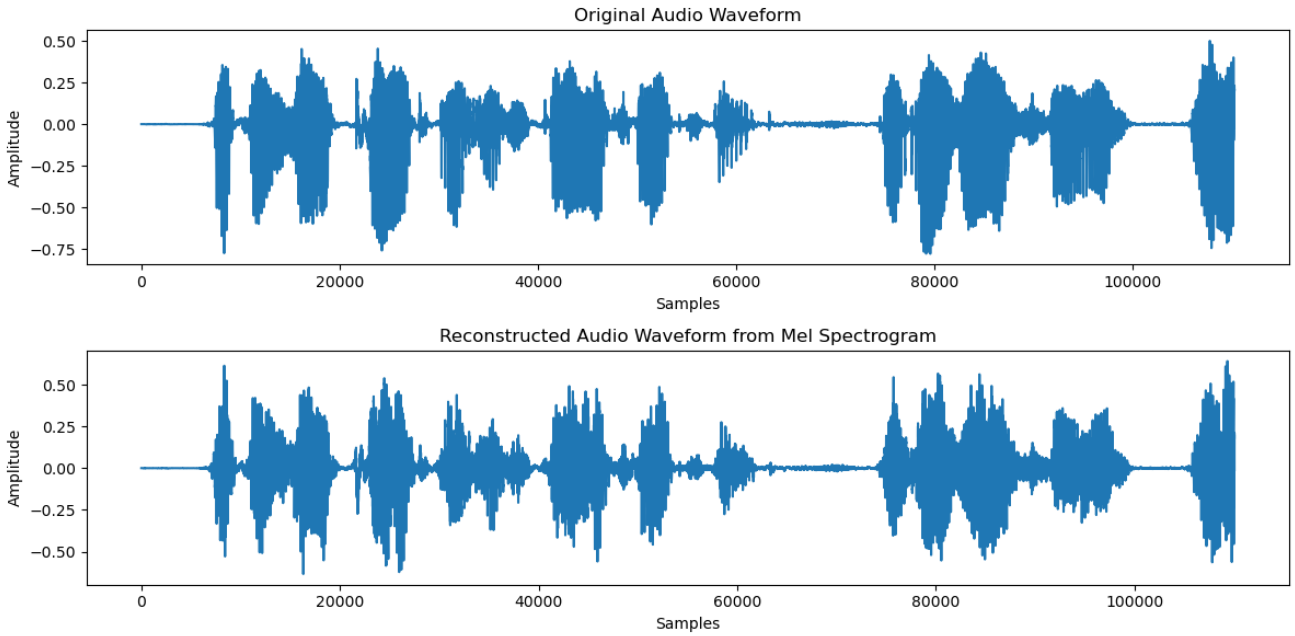

# 22. Inverse Mel to Audio

The reconstructed audio waveform should ideally resemble the original audio waveform used to compute the Mel spectrogram, demonstrating the effectiveness of the Mel representation in retaining the key characteristics of the original audio signal.

Listen for clarity and resemblance to original audio to evaluate reconstruction accuracy.

Useful to assess the quality of Mel-based audio reconstructions, Audio Synthesis, Speech Processing, Machine Learning

In [68]:
```
audio_reconstructed = librosa.feature.inverse.mel_to_audio(mel_spec, sr=sr,

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(y)
plt.title('Original Audio Waveform')
plt.xlabel('Samples')
plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)
plt.plot(audio_reconstructed)
plt.title('Reconstructed Audio Waveform from Mel Spectrogram')
plt.xlabel('Samples')
plt.ylabel('Amplitude')

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

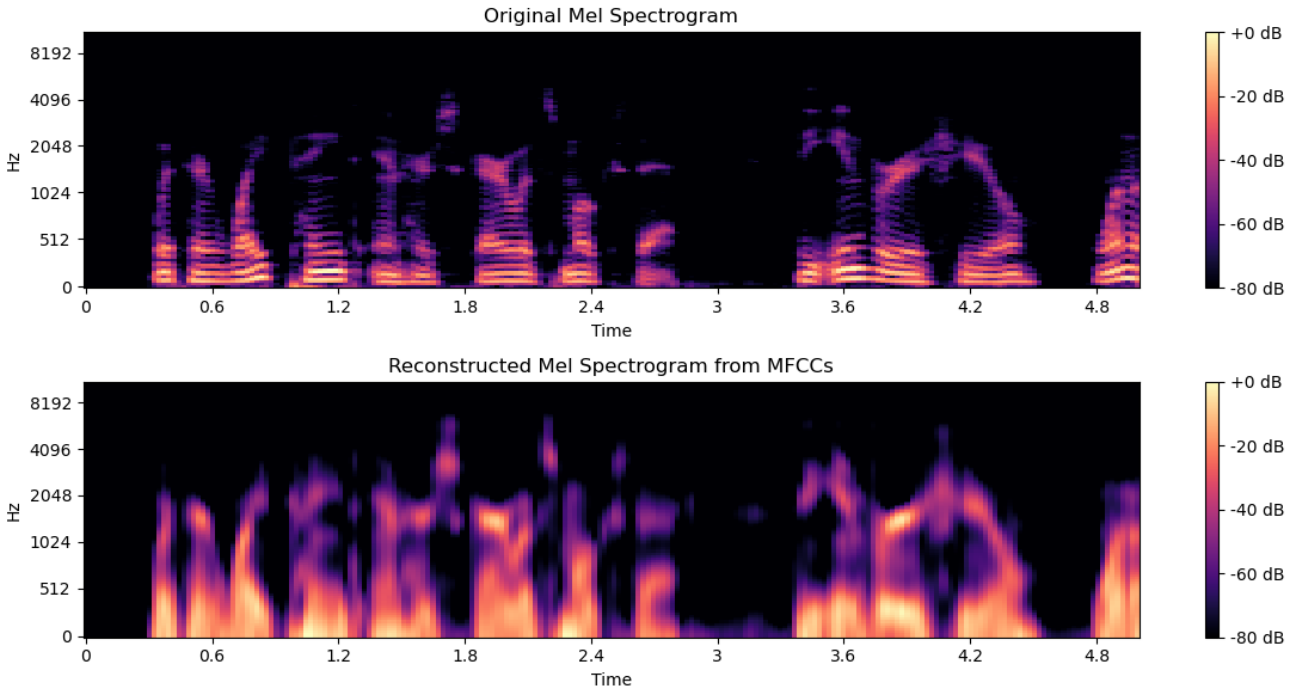



# 23. Inverse MFCC to Mel Spectrogram - Invert MFCC to approximate a Mel power spectrogram

The reconstructed Mel spectrogram should ideally resemble the original Mel spectrogram used to compute the MFCCs.

Useful for obtaining a Mel spectrogram from MFCCs when direct Mel information is unavailable, Speech Processing, Feature Transformation

In [69]:
```
mel_from_mfcc = librosa.feature.inverse.mfcc_to_mel(mfcc)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
mel = librosa.feature.melspectrogram(y=y, sr=sr)
librosa.display.specshow(librosa.amplitude_to_db(mel, ref=np.max), y_axis='m
plt.colorbar(format='%+2.0f dB')
plt.title('Original Mel Spectrogram')

plt.subplot(2, 1, 2)
librosa.display.specshow(librosa.amplitude_to_db(mel_from_mfcc, ref=np.max),
plt.colorbar(format='%+2.0f dB')
plt.title('Reconstructed Mel Spectrogram from MFCCs')

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




# 24. Inverse MFCC to Audio - Convert MFCC to a time-domain audio signal

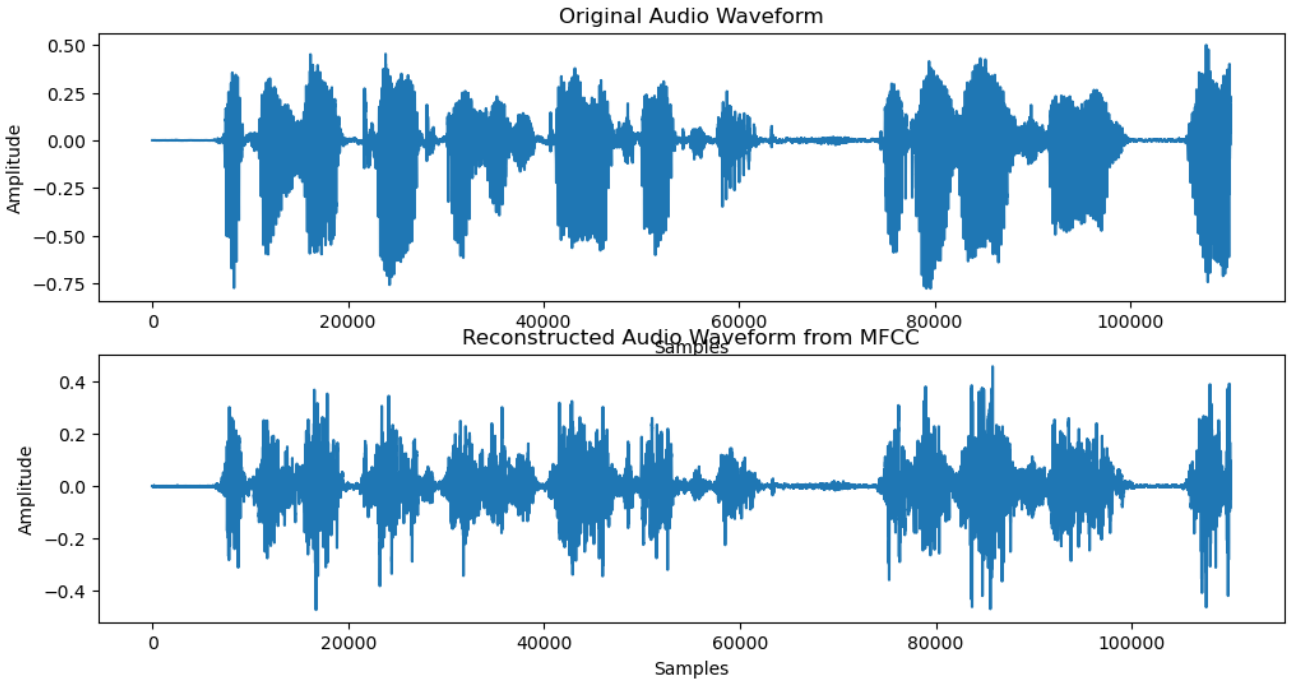
he reconstructed audio waveform should ideally resemble the original audio waveform used to compute the MFCCs, demonstrating how well the MFCC representation retains the original audio characteristics

Useful for approximating original audio from MFCCs, Audio Synthesis, Speech Processing, Machine Learning

In [85]:
```
audio_reconstructed= librosa.feature.inverse.mfcc_to_audio(mfcc)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(y)
plt.title('Original Audio Waveform')
plt.xlabel('Samples')
plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)
plt.plot(audio_reconstructed)
plt.title('Reconstructed Audio Waveform from MFCC')
plt.xlabel('Samples')
plt.ylabel('Amplitude')
# Optionally, listen to the original and reconstructed audio

print("Original Audio")
ipd.Audio(y, rate=sr)

print("Reconstructed Audio")
ipd.Audio(audio_reconstructed, rate=sr)
```

```
Original Audio
Reconstructed Audio
```

Out[85]: 0:00 0:04



In [5]:
```
melspec = librosa.feature.melspectrogram(y=y, sr=sr)
melspec_db = librosa.power_to_db(melspec, ref=np.max)

print(melspec_db)
```

```
[[-60.780678 -61.458168 -71.75807  ... -26.751677 -29.189425 -23.326138]
 [-65.70865  -63.457405 -66.07936  ... -32.420715 -30.399462 -21.965286]
 [-67.82845  -65.46873  -69.30775  ... -30.494272 -22.012863 -14.480339]
 ...
 [-80.       -80.       -80.       ... -80.       -70.86289  -59.469376]
 [-80.       -80.       -80.       ... -80.       -70.895355 -59.50184 ]
 [-80.       -80.       -80.       ... -80.       -70.91782  -59.524303]]
```