**import** librosa

**import** librosa.display

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

*# Load "Purple Rain"*

file\_path\_purple\_rain **=** '09 Purple Rain.wav'

audio\_purple\_rain, sample\_rate\_purple\_rain **=** librosa**.**load(file\_path\_purple\_rain, sr**=None**)

*# Perform a Short-Time Fourier Transform (STFT) to analyze the frequency content over time*

stft\_purple\_rain **=** librosa**.**stft(audio\_purple\_rain)

db\_stft\_purple\_rain **=** librosa**.**amplitude\_to\_db(abs(stft\_purple\_rain))

*# Plot the spectrogram for a visual representation of the spectrum over time*

plt**.**figure(figsize**=**(14, 5))

librosa**.**display**.**specshow(db\_stft\_purple\_rain, sr**=**sample\_rate\_purple\_rain, x\_axis**=**'time', y\_axis**=**'log', cmap**=**'coolwarm')

plt**.**colorbar(format**=**'%+2.0f dB')

plt**.**title('Spectral Analysis of Purple Rain')

plt**.**show()

*# Calculate the spectral centroid to measure the "center of mass" for the sound spectrum, indicating brightness*

spectral\_centroids\_purple\_rain **=** librosa**.**feature**.**spectral\_centroid(y**=**audio\_purple\_rain, sr**=**sample\_rate\_purple\_rain)[0]

*# Calculate the spectral bandwidth, which reflects the width of the sound spectrum around the centroid*

spectral\_bandwidth\_purple\_rain **=** librosa**.**feature**.**spectral\_bandwidth(y**=**audio\_purple\_rain, sr**=**sample\_rate\_purple\_rain)[0]

*# Convert frame counts to time to plot these spectral features*

frames **=** range(len(spectral\_centroids\_purple\_rain))

t\_purple\_rain **=** librosa**.**frames\_to\_time(frames, sr**=**sample\_rate\_purple\_rain)

*# Plot spectral features over the waveform for detailed analysis*

plt**.**figure(figsize**=**(15, 5))

librosa**.**display**.**waveshow(audio\_purple\_rain, sr**=**sample\_rate\_purple\_rain, alpha**=**0.4, color**=**"blue")

plt**.**plot(t\_purple\_rain, spectral\_centroids\_purple\_rain, color**=**'r', label**=**'Spectral Centroid')

plt**.**fill\_between(t\_purple\_rain, spectral\_centroids\_purple\_rain **-** spectral\_bandwidth\_purple\_rain **/** 2,

spectral\_centroids\_purple\_rain **+** spectral\_bandwidth\_purple\_rain **/** 2, color**=**'b', alpha**=**0.5, label**=**'Spectral Bandwidth')

plt**.**legend(loc**=**'upper right')

plt**.**title('Spectral Features over Waveform of Purple Rain')

plt**.**show()

*# Harmonic and Percussive component separation*

y\_harm\_purple\_rain, y\_perc\_purple\_rain **=** librosa**.**effects**.**hpss(audio\_purple\_rain)

plt**.**figure(figsize**=**(10, 3))

librosa**.**display**.**waveshow(y\_harm\_purple\_rain, sr**=**sample\_rate\_purple\_rain, alpha**=**0.5, color**=**'b', label**=**'Harmonic')

librosa**.**display**.**waveshow(y\_perc\_purple\_rain, sr**=**sample\_rate\_purple\_rain, alpha**=**0.5, color**=**'r', label**=**'Percussive')

plt**.**legend(loc**=**'upper right')

plt**.**title('Harmonic and Percussive Separation for Purple Rain')

plt**.**show()

*# Chroma Feature Extraction to analyze the musical tones involved*

chroma\_purple\_rain **=** librosa**.**feature**.**chroma\_cqt(y**=**audio\_purple\_rain, sr**=**sample\_rate\_purple\_rain)

plt**.**figure(figsize**=**(15, 5))

librosa**.**display**.**specshow(chroma\_purple\_rain, y\_axis**=**'chroma', x\_axis**=**'time')

plt**.**colorbar()

plt**.**title('Chroma Features of Purple Rain')

plt**.**show()

*# Calculate and plot Mel-Frequency Cepstral Coefficients (MFCCs) to capture timbral aspects*

mfccs\_purple\_rain **=** librosa**.**feature**.**mfcc(y**=**audio\_purple\_rain, sr**=**sample\_rate\_purple\_rain, n\_mfcc**=**13)

plt**.**figure(figsize**=**(14, 5))

librosa**.**display**.**specshow(mfccs\_purple\_rain, sr**=**sample\_rate\_purple\_rain, x\_axis**=**'time')

plt**.**colorbar()

plt**.**title('MFCC of Purple Rain')

plt**.**show()

**import** librosa

**import** librosa.display

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**from** sklearn.preprocessing **import** scale, minmax\_scale

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Conv2D, MaxPooling2D, Flatten, Dense

*# Load the audio file*

y, sr **=** librosa**.**load('09 Purple Rain.wav')

*# Trim silence from the beginning and end*

y, \_ **=** librosa**.**effects**.**trim(y)

*# Harmonic and Percussive separation*

y\_harm, y\_perc **=** librosa**.**effects**.**hpss(y)

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

plt**.**plot(y\_harm, color**=**'#A300F9', label**=**'Harmonic')

plt**.**plot(y\_perc, color**=**'#FFB100', label**=**'Percussive')

plt**.**legend()

plt**.**title('Harmonic and Percussive Components')

plt**.**show()

*# Calculate the Mel-scaled spectrogram*

S **=** librosa**.**feature**.**melspectrogram(y**=**y, sr**=**sr)

S\_DB **=** librosa**.**power\_to\_db(S, ref**=**np**.**max)

*# Plot the Mel-frequency spectrogram*

plt**.**figure(figsize**=**(10, 4))

mappable **=** librosa**.**display**.**specshow(S\_DB, sr**=**sr, x\_axis**=**'time', y\_axis**=**'mel')

plt**.**colorbar(mappable, format**=**'%+2.0f dB')

plt**.**title('Mel-frequency spectrogram')

plt**.**tight\_layout()

plt**.**show()

*# Calculate Spectral Centroid and Spectral Roll-off*

spectral\_centroids **=** librosa**.**feature**.**spectral\_centroid(y**=**y, sr**=**sr)[0]

spectral\_rolloff **=** librosa**.**feature**.**spectral\_rolloff(y**=**y, sr**=**sr)[0]

frames **=** range(len(spectral\_centroids))

t **=** librosa**.**frames\_to\_time(frames, sr**=**sr)

*# Normalize and plot Spectral Centroid and Roll-off*

normalized\_centroids **=** minmax\_scale(spectral\_centroids, axis**=**0)

normalized\_rolloff **=** minmax\_scale(spectral\_rolloff, axis**=**0)

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

librosa**.**display**.**waveshow(y, sr**=**sr, alpha**=**0.4, color**=**'#A300F9')

plt**.**plot(t, normalized\_centroids, color**=**'#FFB100', label**=**'Spectral Centroid')

plt**.**plot(t, normalized\_rolloff, color**=**'b', label**=**'Spectral Roll-off')

plt**.**legend()

plt**.**show()

*# Calculate MFCCs and plot*

mfccs **=** librosa**.**feature**.**mfcc(y**=**y, sr**=**sr)

mfccs\_scaled **=** scale(mfccs, axis**=**1) *# Scale MFCCs across the feature axis*

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

librosa**.**display**.**specshow(mfccs\_scaled, sr**=**sr, x\_axis**=**'time', cmap**=**'cool')

plt**.**colorbar()

plt**.**title('MFCC')

plt**.**show()

*# Define a deep learning model architecture for genre classification*

model **=** Sequential([

Conv2D(32, (3, 3), activation**=**'relu', input\_shape**=**(128, 128, 1)),

MaxPooling2D(2, 2),

Conv2D(64, (3, 3), activation**=**'relu'),

MaxPooling2D(2, 2),

Flatten(),

Dense(64, activation**=**'relu'),

Dense(10, activation**=**'softmax')

])

model**.**compile(optimizer**=**'adam', loss**=**'categorical\_crossentropy', metrics**=**['accuracy'])

print(model**.**summary())

*# Calculate Chromagram and plot*

chromagram **=** librosa**.**feature**.**chroma\_stft(y**=**y, sr**=**sr, hop\_length**=**512)

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

librosa**.**display**.**specshow(chromagram, x\_axis**=**'time', y\_axis**=**'chroma', hop\_length**=**512, cmap**=**'coolwarm')

plt**.**colorbar()

plt**.**title('Chromagram')

plt**.**show()

**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('09 Purple Rain.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()

*# Spectral Rolloff*

*# 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()

*# Load the audio file*

y, sr **=** librosa**.**load('09 Purple Rain.wav')

*# 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()

*# 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 "Free Fallin"')

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()

*# 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()

*# 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()

*# 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')

print('Audio File shape:', np**.**shape(audio\_file))

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

librosa**.**display**.**waveshow(y **=** audio\_file, sr **=** sr, color **=** "#A300F9");

plt**.**title("Free Fallin", fontsize **=** 23);

plt**.**show()

*# 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))

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

plt**.**plot(D);

plt**.**show()

**import** librosa

**import** librosa.display

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

*# Load an audio file*

y, sr **=** librosa**.**load('09 Purple Rain.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()

y\_harm, y\_perc **=** librosa**.**effects**.**hpss(audio\_file)

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

plt**.**plot(y\_harm, color **=** '#A300F9');

plt**.**plot(y\_perc, color **=** '#FFB100');

plt**.**show()

**import** librosa

**import** pandas **as** pd

**import** numpy **as** np

*# Load the audio file*

file\_path **=** '09 Purple Rain.wav'

audio, sr **=** librosa**.**load(file\_path, sr**=None**)

*# Calculate the Spectral Centroid*

spectral\_centroids **=** librosa**.**feature**.**spectral\_centroid(y**=**audio, sr**=**sr)

*# Calculate the Spectral Bandwidth*

spectral\_bandwidth **=** librosa**.**feature**.**spectral\_bandwidth(y**=**audio, sr**=**sr)

*# Calculate Harmonic and Percussive components*

y\_harmonic, y\_percussive **=** librosa**.**effects**.**hpss(audio)

*# Calculate MFCCs*

mfccs **=** librosa**.**feature**.**mfcc(y**=**audio, sr**=**sr, n\_mfcc**=**13)

*# Calculate statistics for each feature*

features **=** {

'Spectral Centroid Mean': np**.**mean(spectral\_centroids),

'Spectral Centroid Variance': np**.**var(spectral\_centroids),

'Spectral Bandwidth Mean': np**.**mean(spectral\_bandwidth),

'Spectral Bandwidth Variance': np**.**var(spectral\_bandwidth),

'Harmonic Mean': np**.**mean(y\_harmonic),

'Harmonic Variance': np**.**var(y\_harmonic),

'Percussive Mean': np**.**mean(y\_percussive),

'Percussive Variance': np**.**var(y\_percussive),

}

*# Add MFCCs*

**for** i, mfcc **in** enumerate(mfccs, 1):

features[f'MFCC {i} Mean'] **=** np**.**mean(mfcc)

features[f'MFCC {i} Variance'] **=** np**.**var(mfcc)

*# Create a DataFrame*

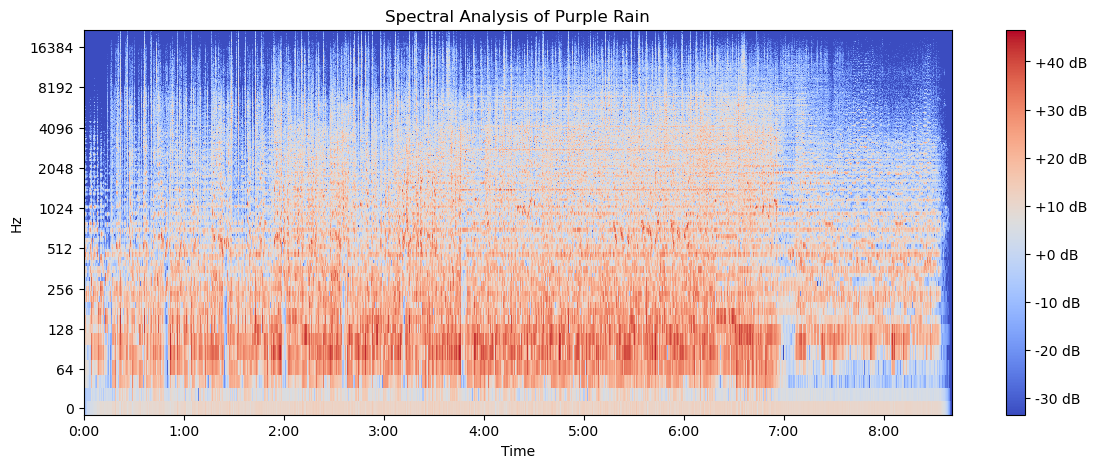
df\_features **=** pd**.**DataFrame([features])

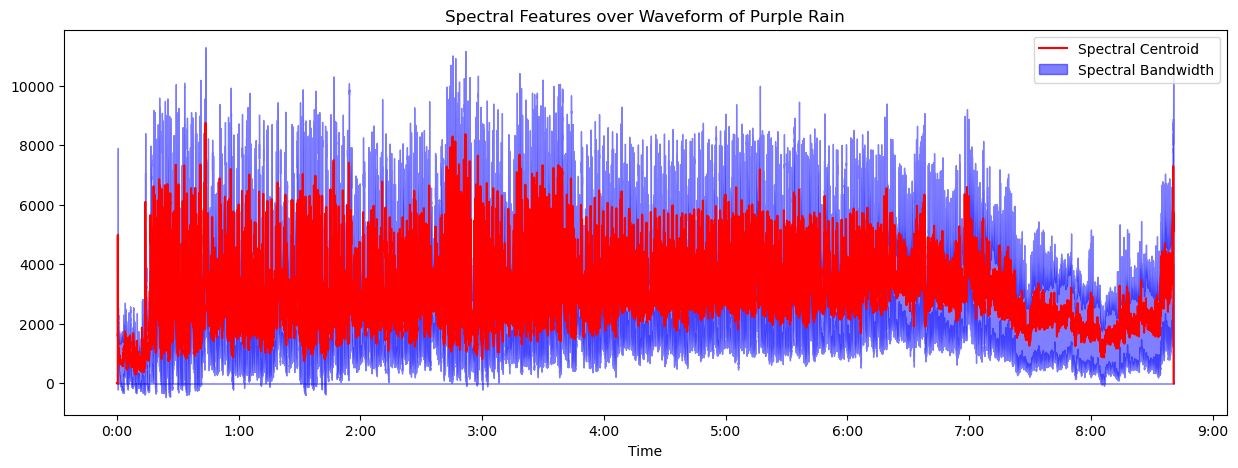
*# Save to CSV*

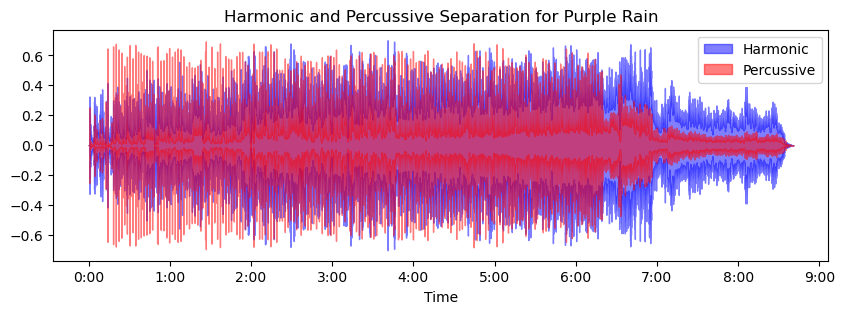
df\_features**.**to\_csv('audio\_features\_purple\_rain.csv', index**=False**)

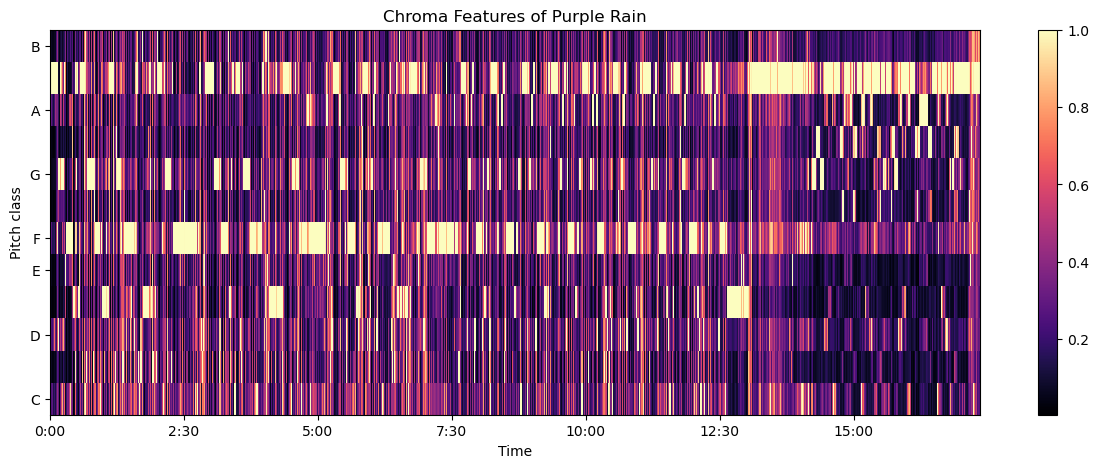
print("CSV file has been created with the following features:\n")

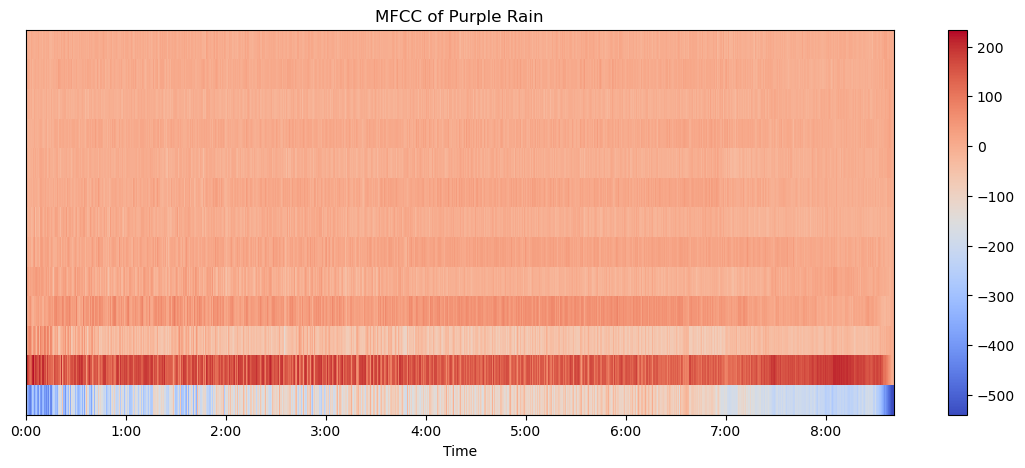
print(df\_features**.**head())

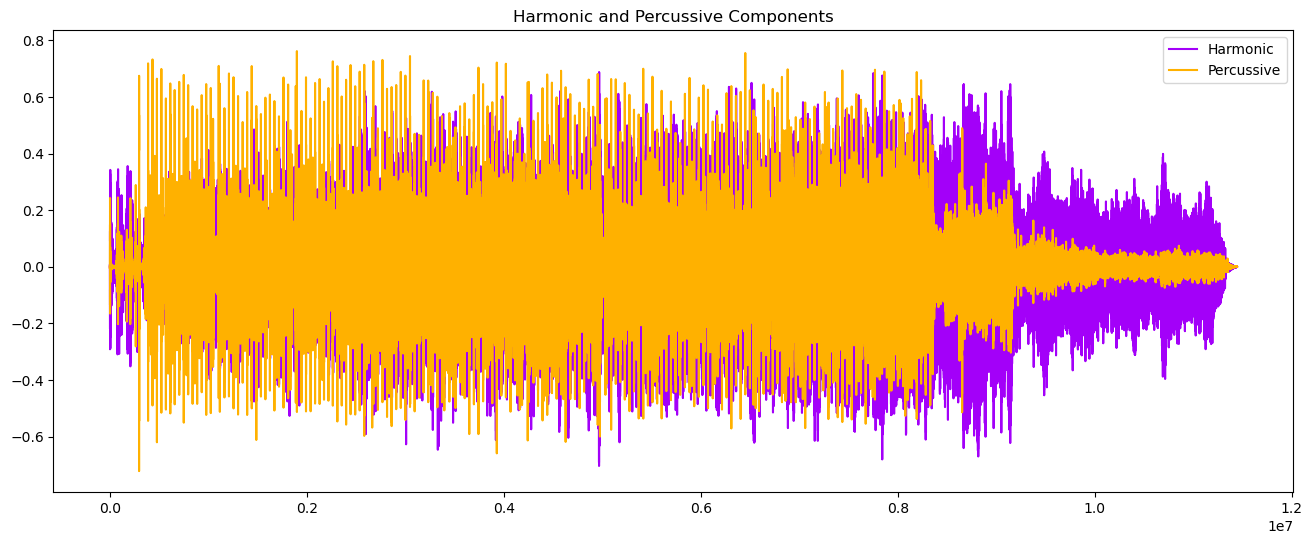


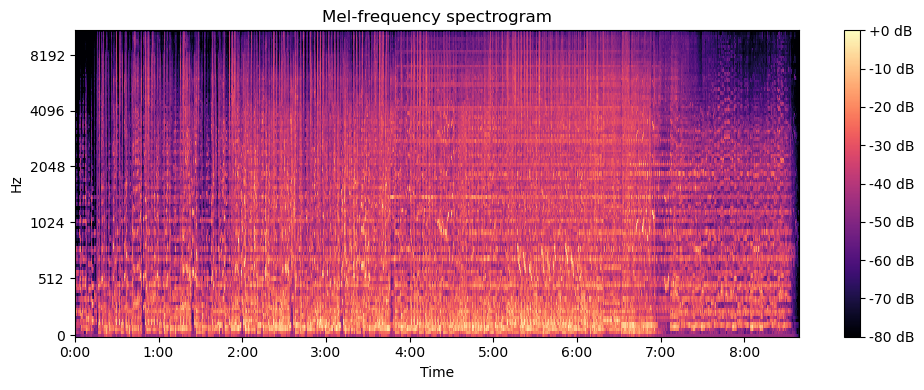


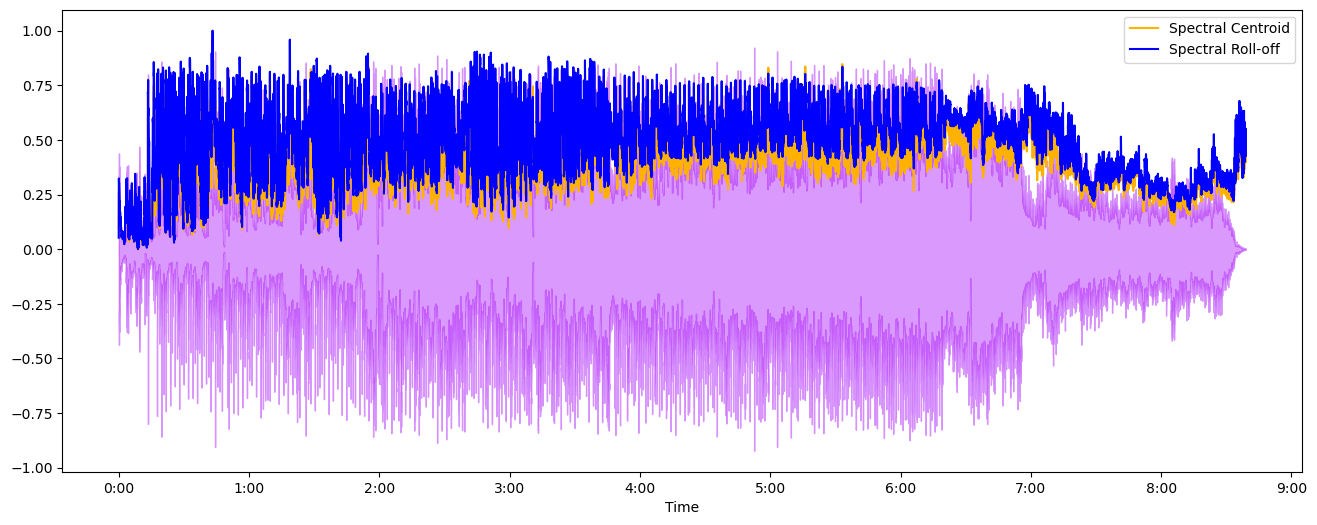










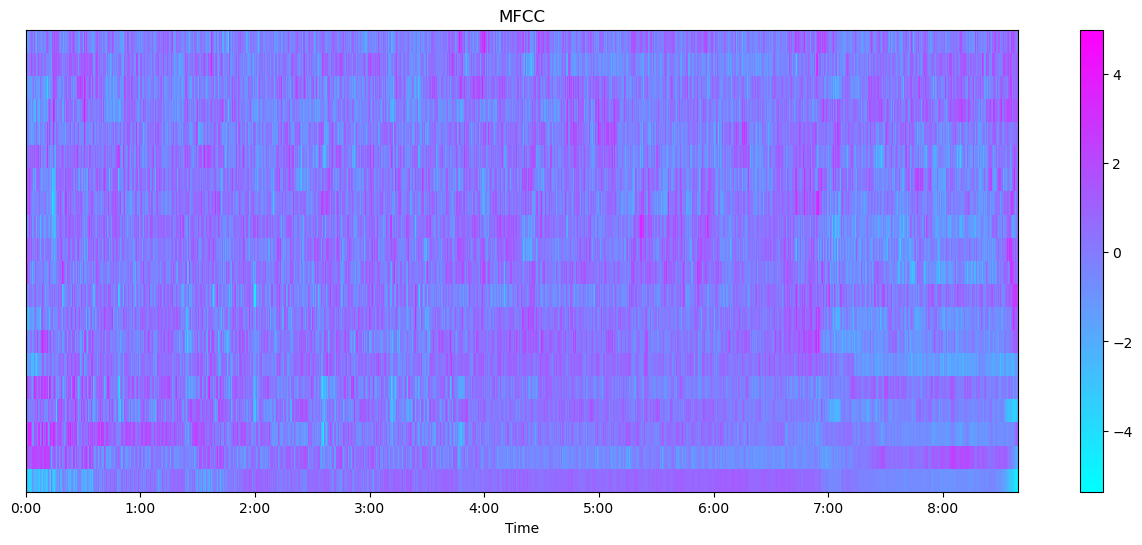


c:\ProgramData\Anaconda3\envs\ml\Lib\site-packages\sklearn\preprocessing\\_data.py:261: UserWarning: Numerical issues were encountered when centering the data and might not be solved. Dataset may contain too large values. You may need to prescale your features.

warnings.warn(

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.

warnings.warn(



c:\ProgramData\Anaconda3\envs\ml\Lib\site-packages\keras\src\layers\convolutional\base\_conv.py:99: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(

**Model: "sequential"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ conv2d (Conv2D) │ (None, 126, 126, 32) │ 320 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d (MaxPooling2D) │ (None, 63, 63, 32) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_1 (Conv2D) │ (None, 61, 61, 64) │ 18,496 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 30, 30, 64) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ flatten (Flatten) │ (None, 57600) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense (Dense) │ (None, 64) │ 3,686,464 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_1 (Dense) │ (None, 10) │ 650 │

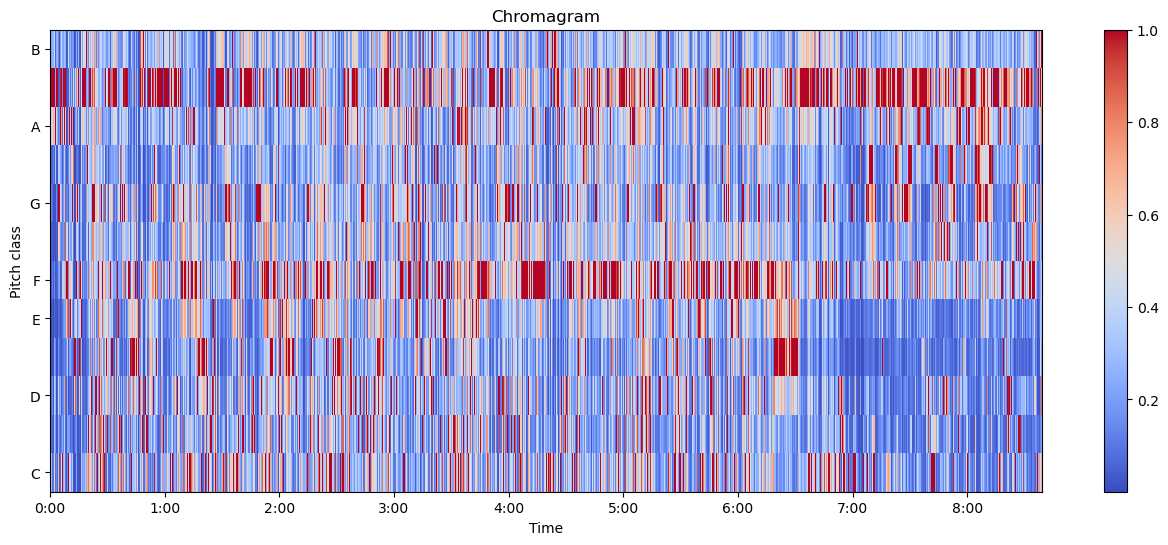
└─────────────────────────────────┴────────────────────────┴───────────────┘

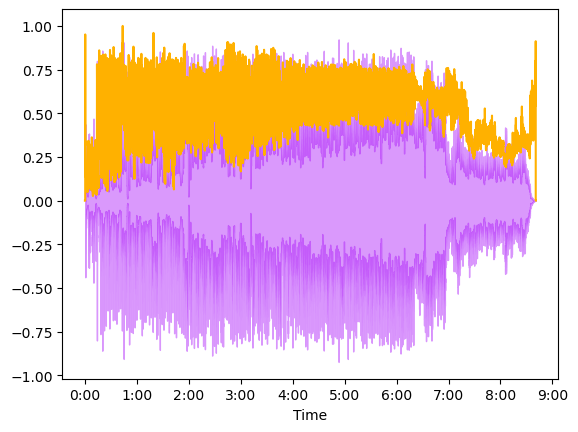
**Total params:** 3,705,930 (14.14 MB)

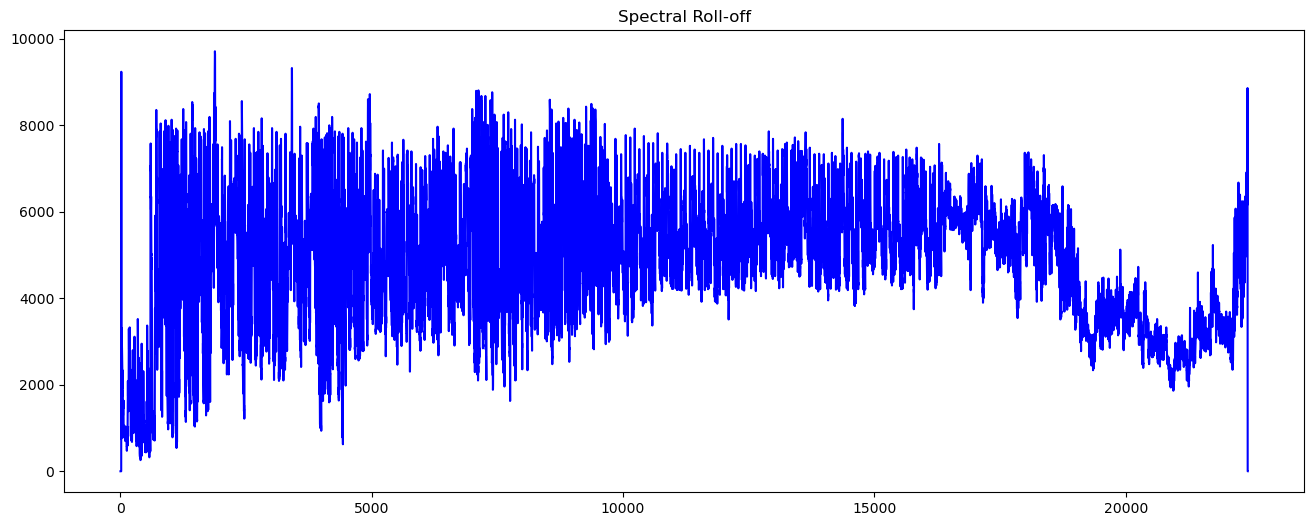
**Trainable params:** 3,705,930 (14.14 MB)

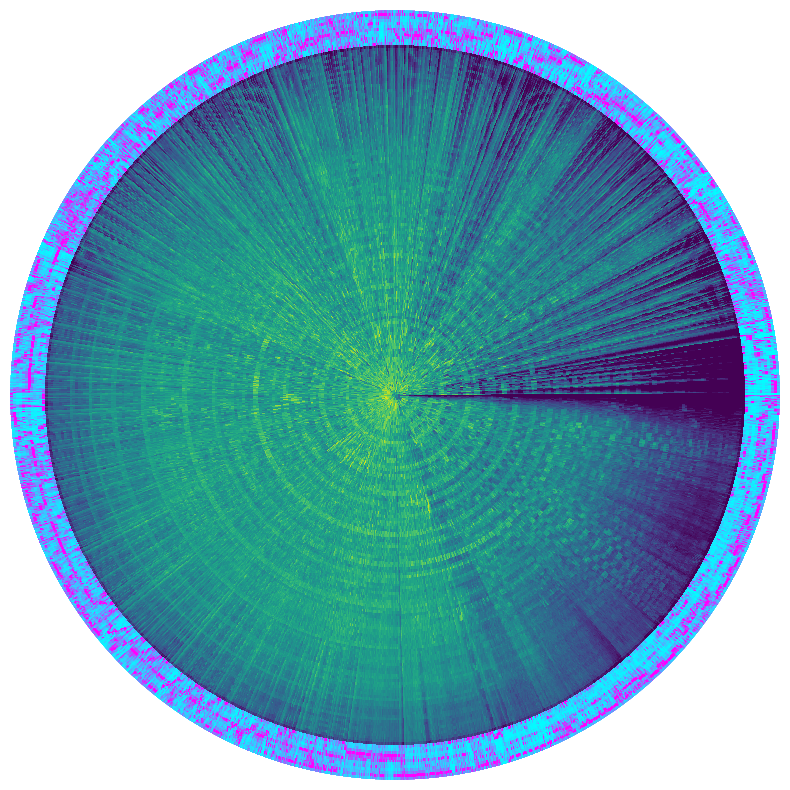
**Non-trainable params:** 0 (0.00 B)

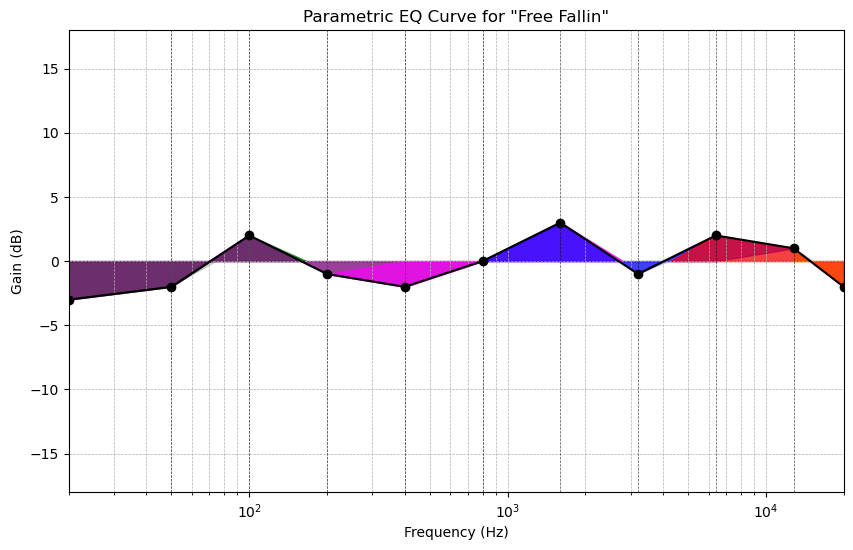
None

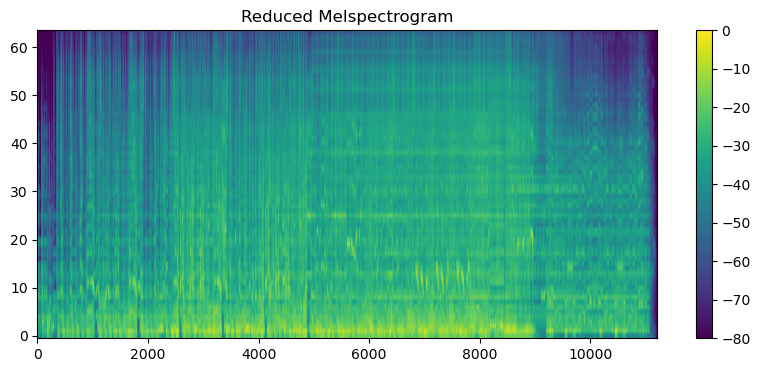


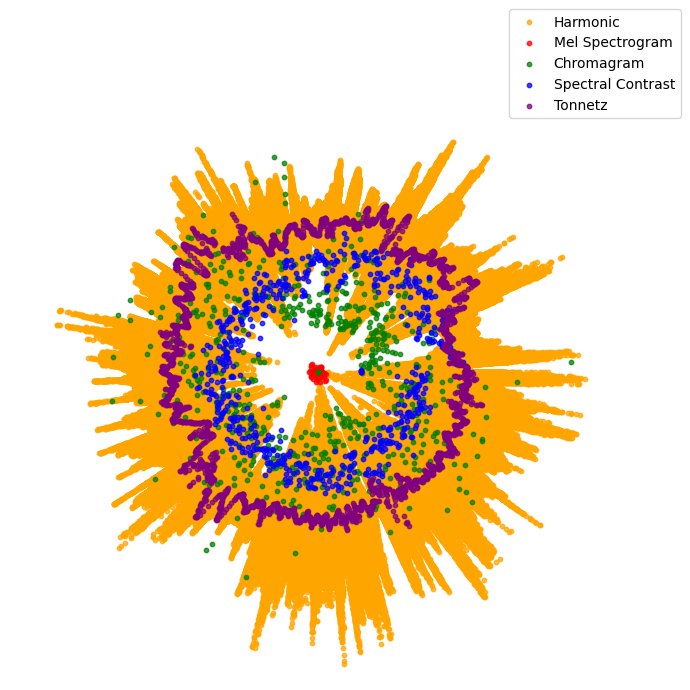








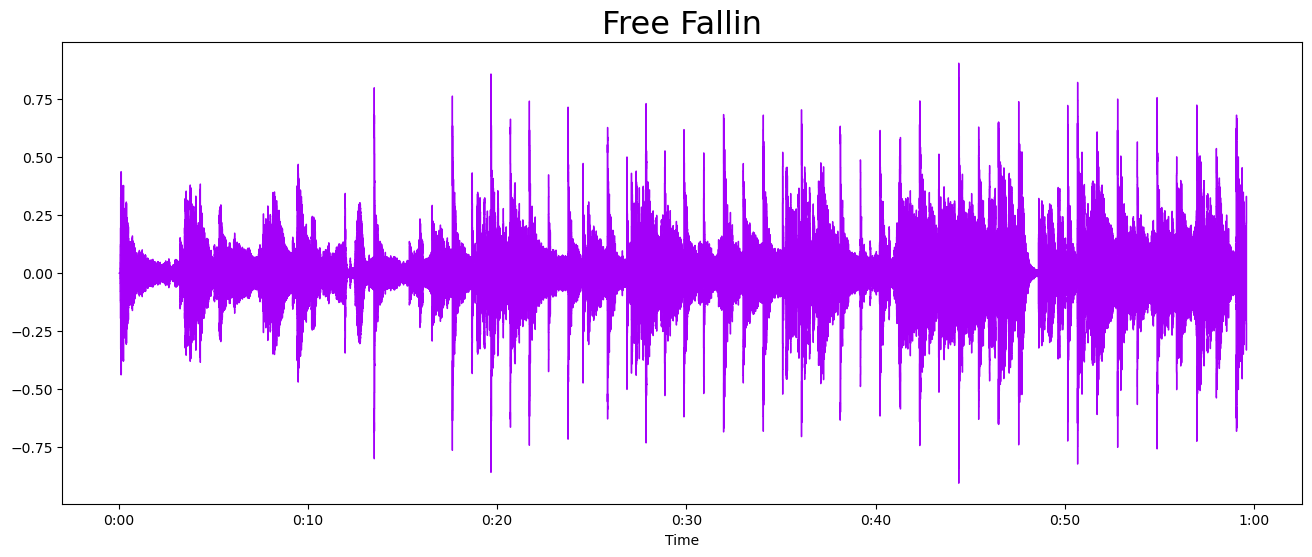




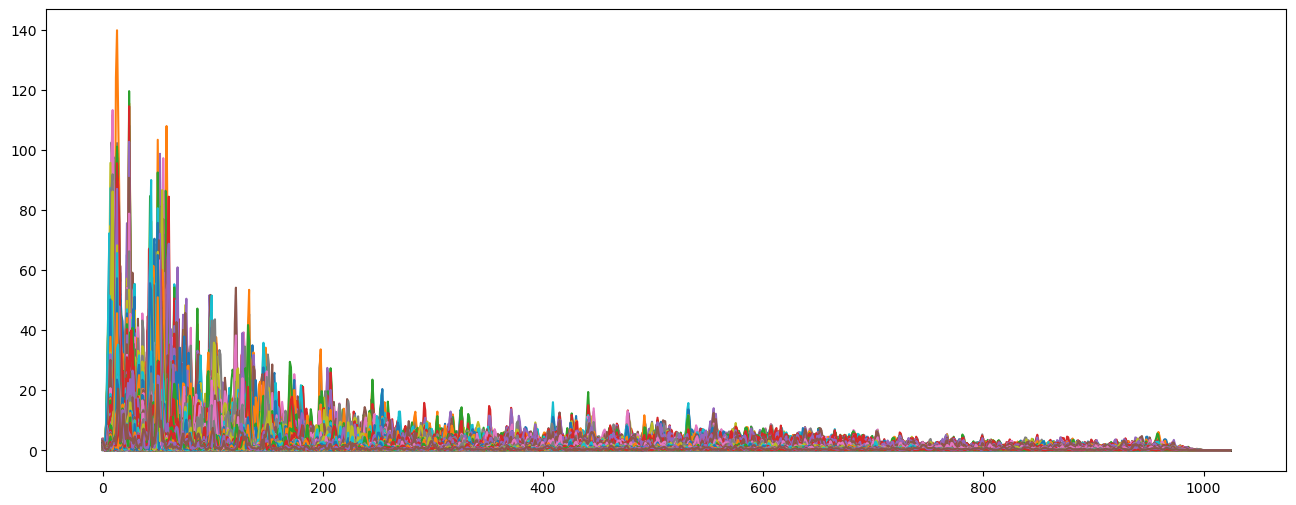
Audio File: [ 0. 0. 0. ... -0.00061406 -0.01219668

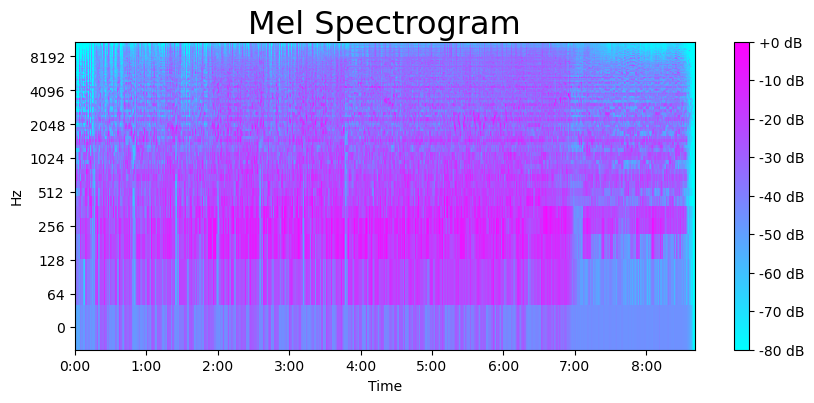
0.01087289]

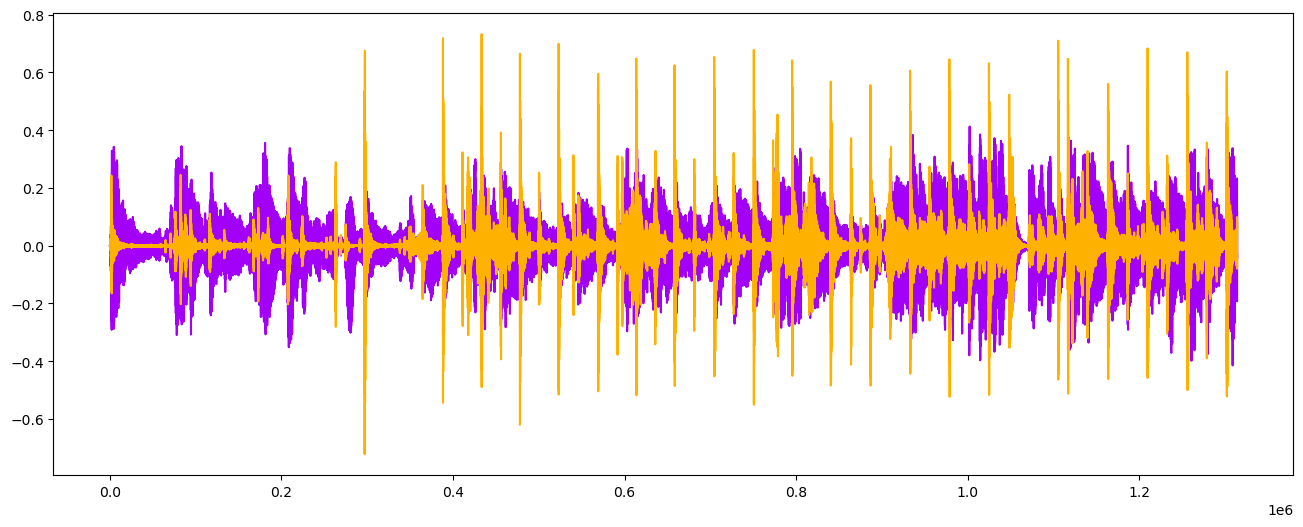
Audio File shape: (1313784,)



Shape of D object: (1025, 2566)







CSV file has been created with the following features:

Spectral Centroid Mean Spectral Centroid Variance \

0 2788.123933 1.290150e+06

Spectral Bandwidth Mean Spectral Bandwidth Variance Harmonic Mean \

0 3176.521325 727782.02066 -0.000647

Harmonic Variance Percussive Mean Percussive Variance MFCC 1 Mean \

0 0.011172 -0.002803 0.001815 -151.836197

MFCC 1 Variance ... MFCC 9 Mean MFCC 9 Variance MFCC 10 Mean \

0 7640.326172 ... -3.235682 82.697403 5.922324

MFCC 10 Variance MFCC 11 Mean MFCC 11 Variance MFCC 12 Mean \

0 80.26339 -4.196476 63.863556 4.455205

MFCC 12 Variance MFCC 13 Mean MFCC 13 Variance

0 69.409782 0.670831 55.529629

[1 rows x 34 columns]

In [ ]:

*# # You might need to install nbformat and nbconvert if not already installed*

*# # pip install nbformat nbconvert*

*# import nbformat*

*# from nbconvert import HTMLExporter*

*# # Load your notebook*

*# with open('purple\_rain,ipynb', 'r', encoding='utf-8') as f:*

*# nb = nbformat.read(f, as\_version=4)*

*# # Setup a HTML exporter and convert the notebook*

*# html\_exporter = HTMLExporter()A*

*# html\_exporter.template\_name = 'classic'*

*# (body, resources) = html\_exporter.from\_notebook\_node(nb)*

*# # Write the output HTML file*

*# with open('purple\_rain.html', 'w', encoding='utf-8') as f:*

*# f.write(body)*

In [ ]: