Introduction to Audio Signal Processing with Librosa

- The video focuses on implementing the **amplitude envelope** feature from scratch using **librosa**, an audio processing library in Python. It also covers plotting waveforms and the amplitude envelope itself.
- `librosa` does not have a built-in amplitude envelope extractor, so one will be built manually.

Loading and Playing Audio Files

- To begin, `librosa` and `librosa.display` (for plotting utilities) are imported.
- Three 30-second WAV audio files are used for demonstration: an orchestral piece by Debussy, a jazz piece by Duke Ellington, and a rock song by Red Hot Chili Peppers.
- `IPython.display.Audio` (`ipd.Audio`) is used to play audio directly within a Jupyter Notebook environment.
- Audio files are loaded using `librosa.load()`, which returns the audio **signal** (a NumPy array) and its **sample rate**.
- The default sample rate for `librosa.load()` is 22050 Hz, which is generally suitable for typical needs.
- By default, `librosa.load()` converts audio to **mono** (single channel), which is often sufficient as significant information is usually not lost compared to stereo audio for many problems.

Calculating the Amplitude Envelope

- The **amplitude envelope** for a specific frame is determined by taking the **maximum amplitude value** across all samples within that frame.
- Two Python functions are demonstrated for calculating the amplitude envelope:

Simple Amplitude Envelope Function

- This function takes a `signal` and a `frame_size`.
- It iterates through the signal in steps equal to the `frame_size`, effectively creating **non-overlapping frames**.
- For each frame, it calculates the maximum value of the signal slice `signal[i : i + frame_size]`.
- The maximum values for each frame are collected into a list and then converted to a NumPy array. A common `frame_size` is 1024 samples.

Audio Signal Properties

- The total number of samples in a signal can be found using `signal.size`.
- The **duration of one sample** (in seconds) is calculated as the inverse of the sample rate: \text{sample_duration} = 1 / \text{sample_rate}.
- The **total duration of the audio signal** (in seconds) is calculated by multiplying the sample duration by the total number of samples: \text{signal_duration} = \text{sample_duration} \times \text{total_samples}. For the examples, this confirms the 30-second length of each audio file.

Visualizing Waveforms

- `matplotlib.pyplot` is imported as `plt` for plotting.
- Waveforms are visualized using `librosa.display.waveplot()` (or `waveshow()`), which takes the signal as input.
- Multiple waveforms can be stacked vertically using `plt.subplot(rows, columns, index)`.
- `plt.title()` sets the plot title, and `plt.ylim()` can set the y-axis range (e.g., -1 to 1 for normalized audio). Adding `alpha` can improve visual clarity.

Waveform Characteristics by Genre

- **Debussy (Classical Music):** The waveform shows a fluid envelope with a large rise and fall of intensity, indicating high variability at a macro level due to acoustic instruments.
- **Red Hot Chili Peppers (Popular/Rock Music):** The waveform has a relatively stable overall envelope with distinct, regular spikes, typically caused by drum elements like kick and snare drums. This stability is common in popular music genres using electric instruments.
- **Duke Ellington (Jazz Music):** Displays characteristics of both classical and rock music, with considerable intensity variability, though more at a micro level rather than large macro shifts.

Music Genre	Waveform Characteristics	Amplitude Envelope Observations
Classical	Fluid envelope; high variability; large, macro-level changes in tension and energy.	Fluid; generally lower mean amplitude.
Rock/ Popular	Stable overall envelope; distinct, regular spikes (e.g., drum kit). Less variability.	Generally higher mean amplitude than classical or jazz; stable with clear, periodic spikes.
Jazz	Mix of both; considerable micro-level intensity variability.	Variable, but without the extreme macro-level changes of classical or the consistent high mean of popular music.

Amplitude Envelope with Overlapping Frames

- For **overlapping frames**, an additional parameter, `hop_length`, is introduced alongside `frame_size` [s].
- The `hop_length` specifies how many samples to shift to the right to start the next frame [s].
- The iteration step in the function is changed from `frame_size` to `hop_length` (`for i in range(0, signal.size, hop_length)`) [s].
- A typical `hop_length` might be 512 samples when `frame_size` is 1024 [s] [s].

"Fancy" Amplitude Envelope Function

- A more concise "fancy" function achieves the same result using a single-line list comprehension within a NumPy array creation:

  python

  ```python
  np.array([max(signal[i:i+frame_size]) for i in range(0, signal.size, hop_length)])
  ```

  [s] [s]

- Both the simple and fancy functions produce identical results for the amplitude envelope calculation [s] [s].

Visualizing the Amplitude Envelope

- The calculated amplitude envelope can be plotted alongside the waveform using `plt.plot()` [s].
- To align the envelope with the waveform's time axis, the frame indices of the amplitude envelope must be converted to time values [s] [s].
- This conversion is done using `librosa.frames_to_time()`, which takes the frame indices (e.g., `np.arange(0, envelope.size)`) and the `hop_length` [s].
- The plotted amplitude envelope (often in red) visually follows the overall intensity contour of the waveform [s].
- **Key takeaways from visualization:** Rock/popular music often shows higher mean amplitude envelopes and distinct spikes from drums compared to classical or jazz music [s].