Lab 1: Data Integration and Preparation

Duration: 3-5 minutes **Presenter Guide:** This lab introduces audio data loading and basic concepts

---

Slide 1: Lab Overview (30 seconds)

Title: "Lab 1: Data Integration and Preparation"

Subtitle: "Getting Audio into Python"

What you'll learn: - How to load audio files in Python - Understanding audio as numerical data - Basic audio visualization

Team Member Connections: - **Cameron's POC:** Starts with loading "Intergalactic" audio file - **Yovannoa's Classifier:** Demonstrated dataset loading and transformation - **cervanj2's Architecture:** This is the foundation of the Preprocessing Layer

---

Slide 2: Audio as Data (45 seconds)

Title: "What is Audio Data?"

Key Concepts: - Audio is stored as numerical samples - Sample rate = samples per second (e.g., 22,050 Hz) - Each sample represents air pressure at a moment in time - We can manipulate audio like any other data

Visual:

```
Time (seconds):   0.0    0.1    0.2    0.3    0.4
Audio values:   [0.23, -0.45, 0.12, -0.89, 0.56, ...]
                 └──────────────────────────────┘
                    22,050 samples per second
```

Talking Points: - "Audio files are just arrays of numbers" - "Each number represents a sound wave position" - "This lets us process audio with code"

Slide 3: Loading Audio with Python (1 minute 30 seconds)

Title: "Hello World: Load and Display Audio"

Code Example:

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

# Load audio file
audio, sr = librosa.load('sample_audio.wav', sr=22050)

# Display basic information
print(f"Audio shape: {audio.shape}")
print(f"Sample rate: {sr}")
print(f"Duration: {len(audio)/sr:.2f} seconds")

# Visualize first 1000 samples
plt.figure(figsize=(10, 4))
plt.plot(audio[:1000])
plt.title("Audio Waveform (first 1000 samples)")
plt.xlabel("Sample Number")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```

Demo Flow: 1. Show the code 2. Run it live (or show pre-recorded output) 3. Point out the printed information 4. Show the waveform plot

Expected Output:

```
Audio shape: (661500,)
Sample rate: 22050
Duration: 30.00 seconds
[Waveform plot appears]
```

Slide 4: Introduction to Spectrograms (1 minute)

Title: "Preview: Spectrograms"

Why spectrograms matter: - Waveforms show amplitude over time - Spectrograms show frequency content over time - Neural networks work better with spectrograms - This is how we'll feed data into U-Net

Simple Code Preview:

```python
# Quick spectrogram preview
spectrogram = librosa.stft(audio)
magnitude = np.abs(spectrogram)

print(f"Waveform shape: {audio.shape}")
print(f"Spectrogram shape: {magnitude.shape}")
```

Expected Output:

```
Waveform shape: (661500,)
Spectrogram shape: (1025, 646)
```

Talking Points: - "Waveform: 661,500 time samples" - "Spectrogram: 1,025 frequencies × 646 time frames" - "This 2D representation is what U-Net will process" - "More on this in Lab 3!"

Slide 5: Connection to the Full Pipeline (45 seconds)

Title: "Where This Fits in the Astro Demo"

The Complete Pipeline:

```
[Lab 1] Load Audio → NumPy Processing → STFT Transform →
PyTorch Tensor → U-Net → ISTFT → Separated Stems
```

cervanj2's Architecture Mapping: 1. **User Interface Layer** ← Audio file selection 2. **Preprocessing Layer** ← We start here (load + prepare) 3. Inference Layer (coming in Lab 5) 4. Post-processing Layer (reverse of Lab 3) 5. Output Layer (save separated audio)

Cameron's POC: - His first step: `audio, sr = librosa.load('Intergalactic.mp3')` - We're replicating his starting point - He then processes 18 slices (more on this in Lab 2)

Demo Script (Detailed Timing) - Provided by Claude

0:00 - 0:30: Introduction

"Welcome to Lab 1: Data Integration and Preparation. Today we're learning how to get audio files into Python so we can process them. This is the foundation of Cameron's Astro POC and cervanj2's preprocessing layer."

0:30 - 1:15: Audio as Data

"Before we code, let's understand what audio really is. [Show Slide 2]. Audio files store numbers - lots of them. At 22,050 Hz, we have 22,050 samples every second. Each sample is just a number representing the air pressure at that moment. This means we can load audio into Python and work with it like any other data."

1:15 - 2:45: Live Coding Demo

"Let's see this in action. [Show Slide 3]. We'll use librosa, a Python library for audio processing that's used in the POC.

First, we import our libraries… [type or show code] Then we load the audio file… [run the code] Look at the output - our audio is now a NumPy array with 661,500 samples!

Now let's visualize it… [show the plot] This waveform shows the audio amplitude over time. You can see the oscillations - that's the sound wave!"

2:45 - 3:45: Spectrograms Preview

"Now here's something cool - and a preview of Lab 3. [Show Slide 4]. While waveforms are great, neural networks actually work better with spectrograms. Watch what happens when we transform our audio…

[Show the code and output]

See that? Our 1D waveform with 661,500 samples became a 2D spectrogram with 1,025 frequency bins and 646 time frames. This 2D image-like representation is what the POC analyzes, and it's what our U-Net will process. We'll dive deep into this in Lab 3."

3:45 - 4:30: Pipeline Context

"So where does this fit in the big picture? [Show Slide 5]. Loading audio is step one of our complete pipeline. Following cervanj2's architecture, we've just completed the first part of the Preprocessing Layer.

The POC starts exactly here - loading the audio file. In the following labs, we'll: - Lab 2: Use NumPy to manipulate this data - Lab 3: Create those spectrograms with librosa - Lab 4: Convert to PyTorch tensors - Lab 5: Process with U-Net for vocal separation

Every step builds on this foundation."

4:30 - 5:00: Wrap-up

"That's Lab 1! You now know how to load audio files, understand them as numerical data, and visualize them. This is the essential first step that everything else builds on.

Questions?"

Code Files to Provide

`lab1_basic.py`

```python
"""
Lab 1: Data Integration and Preparation
Basic audio loading and visualization
"""
import librosa
import numpy as np
import matplotlib.pyplot as plt

def load_audio_basic(filepath):
    """Load audio file and display basic information"""
    # Load audio file
    audio, sr = librosa.load(filepath, sr=22050)

    # Display information
    print("=== Audio Information ===")
    print(f"Audio shape: {audio.shape}")
    print(f"Sample rate: {sr} Hz")
    print(f"Duration: {len(audio)/sr:.2f} seconds")
    print(f"Data type: {audio.dtype}")
    print(f"Value range: [{audio.min():.3f}, {audio.max():.3f}]")

    return audio, sr

def visualize_waveform(audio, sr, num_samples=1000):
    """Visualize audio waveform"""
    plt.figure(figsize=(12, 4))
    plt.plot(audio[:num_samples])
    plt.title(f"Audio Waveform (first {num_samples} samples)")
    plt.xlabel("Sample Number")
    plt.ylabel("Amplitude")
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

def spectrogram_preview(audio):
    """Preview spectrogram transformation"""
    # Compute STFT
    spectrogram = librosa.stft(audio)
    magnitude = np.abs(spectrogram)

    print("\n=== Spectrogram Preview ===")
    print(f"Waveform shape: {audio.shape}")
    print(f"Spectrogram shape: {magnitude.shape}")
    print(f"Frequency bins: {magnitude.shape[0]}")
    print(f"Time frames: {magnitude.shape[1]}")

    return magnitude

if __name__ == "__main__":
    # Example usage
    filepath = "sample_audio.wav"  # Replace with your audio file

    # Load audio
    audio, sr = load_audio_basic(filepath)

    # Visualize
    visualize_waveform(audio, sr)

    # Preview spectrogram
    magnitude = spectrogram_preview(audio)
```

Visual Aids Needed

1. **Audio as Numbers Diagram:** Visual showing waveform → samples
2. **Sample Rate Illustration:** 22,050 samples fitting into 1 second timeline
3. **Pipeline Flowchart:** Highlighting “Load Audio” as step 1
4. **cervanj2’s Architecture Diagram:** With Preprocessing Layer highlighted

---

Backup Slides (If Time Allows)

Extra: Different Audio Formats

- WAV: uncompressed, large files
- MP3: compressed, smaller files
- librosa handles both automatically

Extra: Sample Rate Choices

- 22,050 Hz: Good for most audio processing
- 44,100 Hz: CD quality
- 48,000 Hz: Professional audio
- Lower sample rate = faster processing, lower quality

Q&A Prep - Provided by Claude

Expected Questions:

Q: "Why 22,050 Hz specifically?" A: "It's half of CD quality (44,100 Hz). For vocal separation, we don't need super high frequencies. This provides a good balance between quality and processing speed."

Q: "What file formats work?" A: "librosa handles WAV, MP3, FLAC, OGG and more. It uses audioread under the hood to support many formats."

Q: "Can we load stereo (2-channel) audio?" A: "Yes! By default, librosa converts stereo to mono by averaging the channels. You can keep both channels with `mono=False` if needed."

Q: "What if my audio file is really long?" A: "You can load just a portion using the `duration` and `offset` parameters in librosa.load(). Useful for testing with long files."

Success Criteria

Students should be able to: - [ ] Load an audio file using librosa - [ ] Explain what sample rate means - [ ] Understand audio as numerical data - [ ] Visualize a waveform - [ ] Explain why spectrograms are useful (preview)

Connection Summary

Cameron's POC: First step of his tutorial **Yovannoa's Tutorial:** Dataset loading principles **cervanj2's Architecture:** Preprocessing Layer foundation **Next Lab:** NumPy operations on this data