main

February 3, 2025

1 Imports

[56]:
```
import os
import glob
import math
import numpy as np
import pandas as pd
from scipy.io import wavfile
from tqdm import tqdm
import librosa
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import logging
import warnings
from scipy.io.wavfile import WavFileWarning
```

2 Logger setup

[57]:
```
logger = logging.getLogger("audio_process")
logger.setLevel(logging.DEBUG)

# Clear existing handlers to avoid duplicates
if logger.hasHandlers():
    logger.handlers.clear()

# File Handler: Write all messages (DEBUG and above) to a log file.
fh = logging.FileHandler("process.log", mode='w')
fh.setLevel(logging.DEBUG)
fh_formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)

# Console Handler: Only show INFO and above on the console.
ch = logging.StreamHandler()
```

```
ch.setLevel(logging.WARNING)
ch_formatter = logging.Formatter('%(message)s')
ch.setFormatter(ch_formatter)
logger.addHandler(ch)

# Ignore metadata from scipy.wavfile
warnings.filterwarnings("ignore", category=WavFileWarning)
```

3 Creating and ranking z-vectors

[58]:
```
# 1. Create overlapping windows (z-vectors)
def create_z_vectors(file_path, window_size, hop_size):
    """
    Reads an audio file (WAV or MP3), converts it to mono if needed, and creates
    overlapping windows (each of length window_size) using the given hop_size.
    For MP3 files, librosa is used; for WAV files, scipy.io.wavfile is used.
    """
    ext = os.path.splitext(file_path)[1].lower()
    if ext == '.mp3':
        # Load mp3 using librosa.
        # sr=None preserves the native sampling rate.
        data, sr = librosa.load(file_path, sr=None, mono=True)
        logger.info(f"Processing MP3 '{file_path}' with sample rate: {sr}")
    elif ext == '.wav':
        sr, data = wavfile.read(file_path)
        logger.info(f"Processing WAV '{file_path}' with sample rate: {sr}")
        # If stereo, convert to mono by averaging channels.
        if data.ndim == 2:
            data = data.mean(axis=1)
    else:
        logger.error(f"Unsupported file extension: {ext}")
        raise ValueError(f"Unsupported file extension: {ext}")

    num_samples = len(data)
    vectors = []
    if hop_size > 0:
        num_windows = (num_samples - window_size) // hop_size
        for i in range(num_windows):
            start = i * hop_size
            end = start + window_size
            vectors.append(data[start:end])
    else:
        num_windows = num_samples // window_size
        for i in range(num_windows):
            start = i * window_size
            end = start + window_size
```

```
            vectors.append(data[start:end])
    if num_windows < 1:
        logger.error("Audio too short for the given window/hop parameters.")
        raise ValueError("Audio too short for the given window/hop parameters.")
    return np.array(vectors)


# 2. Compute ordinal patterns (ranking)
def rank_vector(z):
    """
    Returns the ordinal ranking of the values in vector z.
    (Smallest value gets rank 1, next smallest 2, etc.)
    """
    return np.argsort(z).argsort() + 1
```

4 Shannon entropy

[59]:
```
# 3. Shannon entropy (raw)
def compute_shannon_entropy(probabilities):
    """
    Computes Shannon entropy (in bits) given a probability vector.
    Ignores zero probabilities.
    """
    probs = probabilities[probabilities > 0]
    return -np.sum(probs * np.log2(probs))


# 4. Get the observed probability distribution over unique ordinal patterns
def row_probabilities_unique(ranked_z_vectors):
    """
    Given an array of ranked z-vectors (each row is one window's ordinal␣
 ↪pattern),
    returns:
      - probabilities: 1D array of the probability (frequency) of each unique␣
 ↪pattern.
      - unique_patterns: 2D array where each row is a unique ordinal pattern.
    """
    unique_patterns, counts = np.unique(ranked_z_vectors, axis=0,␣
 ↪return_counts=True)
    probabilities = counts / np.sum(counts)
    return probabilities, unique_patterns


# 5. Map an ordinal pattern to an index (using Lehmer code)
def permutation_to_index(perm):
    """
```

```
    Given a permutation (tuple) of length n (with values 1..n),
    returns its index in lexicographic order (0-indexed).
    (This is a simple Lehmer-code mapping.)
    """
    n = len(perm)
    # Convert from 1-indexed to 0-indexed:
    perm = [p - 1 for p in perm]
    index = 0
    for i in range(n):
        # Count how many of the remaining entries are smaller than perm[i]
        smaller = sum(1 for j in range(i + 1, n) if perm[j] < perm[i])
        index += smaller * math.factorial(n - i - 1)
    return index


# 6. Normalized permutation entropy
def compute_normalized_permutation_entropy(probabilities, embedding_dim):
    """
    Normalizes the permutation entropy by the theoretical maximum,
    which is log2(embedding_dim!).
    Returns a tuple: (H_norm, H_raw, H_max)
    """

    M_possible = math.factorial(embedding_dim)
    H_raw = compute_shannon_entropy(probabilities)
    H_max = math.log2(M_possible)
    return H_raw / H_max, H_raw, H_max


# 7. Build the full probability vector (of length m!).
#    For unobserved patterns, the probability is 0.
def compute_extended_probability_vector(probabilities, unique_patterns,␣
 ↪embedding_dim):
    M_possible = math.factorial(embedding_dim)
    p_extended = np.zeros(M_possible)
    # Fill in the observed probabilities by mapping each observed pattern to an␣
 ↪index.
    for prob, pattern in zip(probabilities, unique_patterns):
        idx = permutation_to_index(tuple(pattern))
        p_extended[idx] = prob
    return p_extended
```

5 JSD and complexity

[60]:
```
# 8. Jensen-Shannon divergence between two distributions
def compute_js_divergence(p, p_u):
    """
    Computes the Jensen-Shannon divergence between distributions p and p_u.
    p and p_u must be vectors of the same length.
    """
    m = 0.5 * (p + p_u)
    return compute_shannon_entropy(m) - 0.5 * compute_shannon_entropy(p) - 0.5␣
 ↪* compute_shannon_entropy(p_u)


# 9. Normalized JS divergence (disequilibrium)
def compute_normalized_js_divergence(p_extended, embedding_dim):
    """
    Computes the JS divergence between the observed (extended) distribution
    and the uniform distribution over all m! states, then normalizes it by
    a theoretical maximum.

    The theoretical maximum (Q_max) is given by:
      Q_max = -0.5 * [ ((M+1)/M)*log2(M+1) - 2*log2(2*M) + log2(M) ]
    where M = factorial(embedding_dim)
    """
    M_possible = math.factorial(embedding_dim)
    p_u = np.ones(M_possible) / M_possible
    JS = compute_js_divergence(p_extended, p_u)
    Q_max = -0.5 * (((M_possible + 1) / M_possible) * np.log2(M_possible + 1)
                    - 2 * np.log2(2 * M_possible)
                    + np.log2(M_possible))
    return JS / Q_max if Q_max > 0 else 0.0, JS, Q_max


# 10. Complexity measure: here we simply multiply the two normalized measures.
def complexity_measure(H_norm, JS_norm):
    return H_norm * JS_norm
```

6 Process

[61]:
```
# 11. Process a single audio file to compute H_norm and Complexity
def process_audio(file_path, window_size=10, hop_size=5):
    # 1. Get overlapping windows (z-vectors)
    z_vectors = create_z_vectors(file_path, window_size, hop_size)

    # 2. Compute ordinal (ranking) pattern for each window
    ranked_z_vectors = np.array([rank_vector(z) for z in z_vectors])
```

```
    # 3. Compute the probability distribution over the observed unique patterns
    probabilities, unique_patterns = row_probabilities_unique(ranked_z_vectors)
    # print("Number of unique ordinal patterns (observed):", len(probabilities))

    # 4. Compute the normalized permutation entropy (using the theoretical␣
↪maximum)
    H_norm, H_raw, H_max =␣
↪compute_normalized_permutation_entropy(probabilities, window_size)

    # 5. Extend the observed distribution to the full space of m! outcomes
    p_extended = compute_extended_probability_vector(probabilities,␣
↪unique_patterns, window_size)

    # 6. Compute the normalized JS divergence (disequilibrium)
    JS_norm, JS, Q_max = compute_normalized_js_divergence(p_extended,␣
↪window_size)

    # 7. Compute complexity as the product of the two normalized measures
    comp = complexity_measure(H_norm, JS_norm)

    logger.debug(f"File: {file_path}")
    logger.debug(f"Embedding dimension (window size): {window_size}")
    logger.debug(f"Total possible patterns (m!): {math.factorial(window_size)}")
    logger.debug(f"H_raw: {H_raw}, H_max: {H_max}, Normalized Entropy:␣
↪{H_norm}")
    logger.debug(f"JS raw: {JS}, Q_max: {Q_max}, Normalized JS Divergence:␣
↪{JS_norm}")
    logger.debug(f"Complexity: {comp}")
    logger.debug("-" * 50)

    return H_norm, comp


# 12. Process multiple files and scatter plot all points
def process_folder(folder_path, window_size=10, hop_size=5):
    """
    Processes all WAV and MP3 files in the given folder and returns lists of
    normalized entropy and complexity for each file.
    """
    # Include both .wav and .mp3 files.
    wav_files = glob.glob(os.path.join(folder_path, "*.wav"))
    mp3_files = glob.glob(os.path.join(folder_path, "*.mp3"))
    all_files = wav_files + mp3_files

    if not all_files:
```

```
        logger.info(f"No audio files found in {folder_path}")
        return [], [], []

    all_H = []
    all_comp = []
    file_labels = []

    # Use tqdm to add a progress bar over the file list.
    for file_path in tqdm(all_files, desc=f"Processing audio files in␣
↪{folder_path} with dim {window_size} and hop {hop_size}"):
        try:
            H_norm, comp = process_audio(file_path, window_size, hop_size)
            all_H.append(H_norm)
            all_comp.append(comp)
            file_labels.append(os.path.basename(file_path))
        except Exception as e:
            logger.exception(f"Error processing file {file_path}: {e}")

    return all_H, all_comp, file_labels
```

7 Graphs

[62]:
```
def plot_graph(folder_path, start, end, grid, folder = "plots"):
    logger.info(f"Processing folder {folder_path}")
    # (window sizes 5 to 10)
    iterations = list(range(start, end))
    n_iter = len(iterations)

    # For 2x3 grid
    n_cols = grid
    n_rows = math.ceil(n_iter / n_cols)

    # Create a combined Plotly figure with subplots.
    combined_fig = make_subplots(
        rows=n_rows,
        cols=n_cols,
        subplot_titles=[f"Dim = '{i}', Hop = '{i-1}'" for i in iterations]
    )

    iteration_index = 0
    for i in iterations:
        window_size = i
        hop_size = i - 1

        # Process the folder to obtain entropy & complexity values.
```

```
        all_H, all_comp, file_labels = process_folder(folder_path, window_size,␣
↪hop_size)
        df = pd.DataFrame({
            "Normalized Permutation Entropy": all_H,
            "Normalized Complexity": all_comp,
            "File": file_labels
        })

        trace = go.Scatter(
            x=df["Normalized Permutation Entropy"],
            y=df["Normalized Complexity"],
            mode="markers",
            marker=dict(color=df["Normalized Complexity"],
                        colorscale="Plotly3",
                        size=10),
            text=df["File"],
            hovertemplate=(
                "Entropy: %{x}<br>" +
                "Complexity: %{y}<br>" +
                "File: %{text}<extra></extra>"
            ),
            name=f"Dim = {window_size}, Hop = {hop_size}",
            showlegend=True
        )
        # Determine subplot row and column indices.
        row = iteration_index // n_cols + 1
        col = iteration_index % n_cols + 1
        combined_fig.add_trace(trace, row=row, col=col)
        iteration_index += 1


        # Matplotlib plot.
        plt.figure(figsize=(8, 6))
        plt.scatter(df["Normalized Permutation Entropy"],
                    df["Normalized Complexity"],
                    s=70,
                    c='blue',
                    edgecolors='black')
        plt.xlabel("Normalized Permutation Entropy")
        plt.ylabel("Normalized Complexity")
        plt.title(f"Entropy-Complexity for {folder_path}\nDim =␣
↪'{window_size}', Hop = '{hop_size}'")
        plt.xlim(0, 1)
        plt.ylim(0, 1)
        plt.grid(True, alpha=0.3)
        plt.gca().set_aspect('equal', adjustable='box')
        plt.tight_layout()
```

```
        output_dir = f"{folder}/{os.path.basename(folder_path)}"
        os.makedirs(output_dir, exist_ok=True)
        output_file = os.path.join(output_dir, f"entropy_complexity_{os.path.
↪basename(folder_path)}_dim_{window_size}_hop_{hop_size}.png")
        plt.savefig(output_file)
        plt.close()
        logger.info(f"Saved Matplotlib plot to {output_file}")

    # Finalize the combined Plotly figure.
    combined_fig.update_layout(
        width=1200,
        height=1400,
        title_text=f"Combined Entropy-Complexity Plots for Folder␣
↪{folder_path}",
        template="plotly_dark"
    )

    for axis in combined_fig.layout:
        if axis.startswith("xaxis"):
            combined_fig.layout[axis].update(range=[0, 1])
        if axis.startswith("yaxis"):
            combined_fig.layout[axis].update(range=[0, 1])

    combined_fig.show()
```

Noises and sin waves graph:

[63]:
```
folder_path = "data/SinAndNoise"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/SinAndNoise with dim 4 and hop 3:
100%|      | 18/18 [00:03<00:00,  5.28it/s]
Processing audio files in data/SinAndNoise with dim 5 and hop 4:
100%|      | 18/18 [00:02<00:00,  6.85it/s]
Processing audio files in data/SinAndNoise with dim 6 and hop 5:
100%|      | 18/18 [00:02<00:00,  8.30it/s]
Processing audio files in data/SinAndNoise with dim 7 and hop 6:
100%|      | 18/18 [00:01<00:00,  9.54it/s]
Processing audio files in data/SinAndNoise with dim 8 and hop 7:
100%|      | 18/18 [00:02<00:00,  8.60it/s]
Processing audio files in data/SinAndNoise with dim 9 and hop 8:
100%|      | 18/18 [00:02<00:00,  7.14it/s]
```

Blues

[64]:
```
folder_path = "data/genres_30sec/blues"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/blues with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.76it/s]
Processing audio files in data/genres_30sec/blues with dim 5 and hop 4:
100%|      | 100/100 [00:44<00:00,  2.24it/s]
Processing audio files in data/genres_30sec/blues with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.76it/s]
Processing audio files in data/genres_30sec/blues with dim 7 and hop 6:
100%|      | 100/100 [00:31<00:00,  3.13it/s]
Processing audio files in data/genres_30sec/blues with dim 8 and hop 7:
100%|      | 100/100 [00:31<00:00,  3.21it/s]
Processing audio files in data/genres_30sec/blues with dim 9 and hop 8:
100%|      | 100/100 [00:34<00:00,  2.92it/s]
```

Classical

[65]:
```
folder_path = "data/genres_30sec/classical"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/classical with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.76it/s]
Processing audio files in data/genres_30sec/classical with dim 5 and hop 4:
100%|      | 100/100 [00:44<00:00,  2.26it/s]
Processing audio files in data/genres_30sec/classical with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.76it/s]
Processing audio files in data/genres_30sec/classical with dim 7 and hop 6:
100%|      | 100/100 [00:31<00:00,  3.13it/s]
Processing audio files in data/genres_30sec/classical with dim 8 and hop 7:
100%|      | 100/100 [00:29<00:00,  3.42it/s]
Processing audio files in data/genres_30sec/classical with dim 9 and hop 8:
100%|      | 100/100 [00:29<00:00,  3.44it/s]
```

Country

[66]:
```
folder_path = "data/genres_30sec/country"
start = 4
end = 10
grid = 2
folder = "plots"
```

```
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/country with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.78it/s]
Processing audio files in data/genres_30sec/country with dim 5 and hop 4:
100%|      | 100/100 [00:44<00:00,  2.27it/s]
Processing audio files in data/genres_30sec/country with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.71it/s]
Processing audio files in data/genres_30sec/country with dim 7 and hop 6:
100%|      | 100/100 [00:32<00:00,  3.05it/s]
Processing audio files in data/genres_30sec/country with dim 8 and hop 7:
100%|      | 100/100 [00:32<00:00,  3.11it/s]
Processing audio files in data/genres_30sec/country with dim 9 and hop 8:
100%|      | 100/100 [00:34<00:00,  2.87it/s]
```

Disco

[67]:
```
folder_path = "data/genres_30sec/disco"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/disco with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.78it/s]
Processing audio files in data/genres_30sec/disco with dim 5 and hop 4:
100%|      | 100/100 [00:44<00:00,  2.25it/s]
Processing audio files in data/genres_30sec/disco with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.76it/s]
Processing audio files in data/genres_30sec/disco with dim 7 and hop 6:
100%|      | 100/100 [00:32<00:00,  3.08it/s]
Processing audio files in data/genres_30sec/disco with dim 8 and hop 7:
100%|      | 100/100 [00:33<00:00,  2.94it/s]
Processing audio files in data/genres_30sec/disco with dim 9 and hop 8:
100%|      | 100/100 [00:40<00:00,  2.47it/s]
```

Hiphop

[68]:
```
folder_path = "data/genres_30sec/hiphop"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/hiphop with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.78it/s]
Processing audio files in data/genres_30sec/hiphop with dim 5 and hop 4:
100%|      | 100/100 [00:43<00:00,  2.28it/s]
```

```
Processing audio files in data/genres_30sec/hiphop with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.76it/s]
Processing audio files in data/genres_30sec/hiphop with dim 7 and hop 6:
100%|      | 100/100 [00:32<00:00,  3.08it/s]
Processing audio files in data/genres_30sec/hiphop with dim 8 and hop 7:
100%|      | 100/100 [00:35<00:00,  2.85it/s]
Processing audio files in data/genres_30sec/hiphop with dim 9 and hop 8:
100%|      | 100/100 [00:41<00:00,  2.43it/s]
```

Jazz

[70]:
```
folder_path = "data/genres_30sec/jazz"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/jazz with dim 4 and hop 3:
44%|       | 44/100 [00:25<00:32,  1.74it/s]Error processing file
data/genres_30sec/jazz/jazz.00054.wav: File format b'\xcb\x15\x1e\x16' not
understood. Only 'RIFF', 'RIFX', and 'RF64' supported.
Traceback (most recent call last):
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 58, in process_folder
    H_norm, comp = process_audio(file_path, window_size, hop_size)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 4, in process_audio
    z_vectors = create_z_vectors(file_path, window_size, hop_size)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/3619294251.py",
line 15, in create_z_vectors
    sr, data = wavfile.read(file_path)
               ^^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 677, in read
    file_size, is_big_endian, is_rf64 = _read_riff_chunk(fid)
                                        ^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 536, in _read_riff_chunk
    raise ValueError(f"File format {repr(str1)} not understood. Only "
ValueError: File format b'\xcb\x15\x1e\x16' not understood. Only 'RIFF', 'RIFX',
and 'RF64' supported.
```

```
Processing audio files in data/genres_30sec/jazz with dim 4 and hop 3:
100%|     | 100/100 [00:56<00:00,  1.78it/s]
Processing audio files in data/genres_30sec/jazz with dim 5 and hop 4:
44%|       | 44/100 [00:19<00:25,  2.23it/s]Error processing file
data/genres_30sec/jazz/jazz.00054.wav: File format b'\xcb\x15\x1e\x16' not
understood. Only 'RIFF', 'RIFX', and 'RF64' supported.
Traceback (most recent call last):
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 58, in process_folder
    H_norm, comp = process_audio(file_path, window_size, hop_size)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 4, in process_audio
    z_vectors = create_z_vectors(file_path, window_size, hop_size)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/3619294251.py",
line 15, in create_z_vectors
    sr, data = wavfile.read(file_path)
               ^^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 677, in read
    file_size, is_big_endian, is_rf64 = _read_riff_chunk(fid)
                                        ^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 536, in _read_riff_chunk
    raise ValueError(f"File format {repr(str1)} not understood. Only "
ValueError: File format b'\xcb\x15\x1e\x16' not understood. Only 'RIFF', 'RIFX',
and 'RF64' supported.
Processing audio files in data/genres_30sec/jazz with dim 5 and hop 4:
100%|     | 100/100 [00:43<00:00,  2.27it/s]
Processing audio files in data/genres_30sec/jazz with dim 6 and hop 5:
44%|       | 44/100 [00:16<00:20,  2.71it/s]Error processing file
data/genres_30sec/jazz/jazz.00054.wav: File format b'\xcb\x15\x1e\x16' not
understood. Only 'RIFF', 'RIFX', and 'RF64' supported.
Traceback (most recent call last):
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 58, in process_folder
    H_norm, comp = process_audio(file_path, window_size, hop_size)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 4, in process_audio
```

```
    z_vectors = create_z_vectors(file_path, window_size, hop_size)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/3619294251.py",
line 15, in create_z_vectors
    sr, data = wavfile.read(file_path)
               ^^^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 677, in read
    file_size, is_big_endian, is_rf64 = _read_riff_chunk(fid)
                                        ^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 536, in _read_riff_chunk
    raise ValueError(f"File format {repr(str1)} not understood. Only "
ValueError: File format b'\xcb\x15\x1e\x16' not understood. Only 'RIFF', 'RIFX',
and 'RF64' supported.
Processing audio files in data/genres_30sec/jazz with dim 6 and hop 5:
100%|     | 100/100 [00:36<00:00,  2.76it/s]
Processing audio files in data/genres_30sec/jazz with dim 7 and hop 6:
44%|       | 44/100 [00:14<00:18,  3.08it/s]Error processing file
data/genres_30sec/jazz/jazz.00054.wav: File format b'\xcb\x15\x1e\x16' not
understood. Only 'RIFF', 'RIFX', and 'RF64' supported.
Traceback (most recent call last):
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 58, in process_folder
    H_norm, comp = process_audio(file_path, window_size, hop_size)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 4, in process_audio
    z_vectors = create_z_vectors(file_path, window_size, hop_size)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/3619294251.py",
line 15, in create_z_vectors
    sr, data = wavfile.read(file_path)
               ^^^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 677, in read
    file_size, is_big_endian, is_rf64 = _read_riff_chunk(fid)
                                        ^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 536, in _read_riff_chunk
```

```
    raise ValueError(f"File format {repr(str1)} not understood. Only "
ValueError: File format b'\xcb\x15\x1e\x16' not understood. Only 'RIFF', 'RIFX',
and 'RF64' supported.
Processing audio files in data/genres_30sec/jazz with dim 7 and hop 6:
100%|     | 100/100 [00:32<00:00,  3.10it/s]
Processing audio files in data/genres_30sec/jazz with dim 8 and hop 7:
44%|       | 44/100 [00:13<00:17,  3.19it/s]Error processing file
data/genres_30sec/jazz/jazz.00054.wav: File format b'\xcb\x15\x1e\x16' not
understood. Only 'RIFF', 'RIFX', and 'RF64' supported.
Traceback (most recent call last):
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 58, in process_folder
    H_norm, comp = process_audio(file_path, window_size, hop_size)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 4, in process_audio
    z_vectors = create_z_vectors(file_path, window_size, hop_size)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/3619294251.py",
line 15, in create_z_vectors
    sr, data = wavfile.read(file_path)
               ^^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 677, in read
    file_size, is_big_endian, is_rf64 = _read_riff_chunk(fid)
                                        ^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 536, in _read_riff_chunk
    raise ValueError(f"File format {repr(str1)} not understood. Only "
ValueError: File format b'\xcb\x15\x1e\x16' not understood. Only 'RIFF', 'RIFX',
and 'RF64' supported.
Processing audio files in data/genres_30sec/jazz with dim 8 and hop 7:
100%|     | 100/100 [00:31<00:00,  3.17it/s]
Processing audio files in data/genres_30sec/jazz with dim 9 and hop 8:
44%|       | 44/100 [00:14<00:17,  3.18it/s]Error processing file
data/genres_30sec/jazz/jazz.00054.wav: File format b'\xcb\x15\x1e\x16' not
understood. Only 'RIFF', 'RIFX', and 'RF64' supported.
Traceback (most recent call last):
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 58, in process_folder
    H_norm, comp = process_audio(file_path, window_size, hop_size)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/2814569364.py",
line 4, in process_audio
    z_vectors = create_z_vectors(file_path, window_size, hop_size)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/var/folders/ww/ltx99cqx36v87nl20lt5x_zc0000gn/T/ipykernel_9172/3619294251.py",
line 15, in create_z_vectors
    sr, data = wavfile.read(file_path)
               ^^^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 677, in read
    file_size, is_big_endian, is_rf64 = _read_riff_chunk(fid)
                                        ^^^^^^^^^^^^^^^^^^^^^

  File
"/Users/mverzhbitskiy/Documents/GitHub/musicAnalysis/venv/lib/python3.11/site-
packages/scipy/io/wavfile.py", line 536, in _read_riff_chunk
    raise ValueError(f"File format {repr(str1)} not understood. Only "
ValueError: File format b'\xcb\x15\x1e\x16' not understood. Only 'RIFF', 'RIFX',
and 'RF64' supported.
Processing audio files in data/genres_30sec/jazz with dim 9 and hop 8:
100%|      | 100/100 [00:34<00:00,  2.93it/s]
```

Metal

[71]:
```
folder_path = "data/genres_30sec/metal"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/metal with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.77it/s]
Processing audio files in data/genres_30sec/metal with dim 5 and hop 4:
100%|      | 100/100 [00:44<00:00,  2.25it/s]
Processing audio files in data/genres_30sec/metal with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.76it/s]
Processing audio files in data/genres_30sec/metal with dim 7 and hop 6:
100%|      | 100/100 [00:31<00:00,  3.14it/s]
Processing audio files in data/genres_30sec/metal with dim 8 and hop 7:
100%|      | 100/100 [00:32<00:00,  3.12it/s]
Processing audio files in data/genres_30sec/metal with dim 9 and hop 8:
100%|      | 100/100 [00:38<00:00,  2.59it/s]
```

Pop

[72]:
```
folder_path = "data/genres_30sec/pop"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/pop with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.78it/s]
Processing audio files in data/genres_30sec/pop with dim 5 and hop 4:
100%|      | 100/100 [00:43<00:00,  2.27it/s]
Processing audio files in data/genres_30sec/pop with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.77it/s]
Processing audio files in data/genres_30sec/pop with dim 7 and hop 6:
100%|      | 100/100 [00:32<00:00,  3.04it/s]
Processing audio files in data/genres_30sec/pop with dim 8 and hop 7:
100%|      | 100/100 [00:36<00:00,  2.74it/s]
Processing audio files in data/genres_30sec/pop with dim 9 and hop 8:
100%|      | 100/100 [00:44<00:00,  2.25it/s]
```

Reggae

[73]:
```
folder_path = "data/genres_30sec/reggae"
start = 4
end = 10
grid = 2
folder = "plots"
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/reggae with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.77it/s]
Processing audio files in data/genres_30sec/reggae with dim 5 and hop 4:
100%|      | 100/100 [00:43<00:00,  2.28it/s]
Processing audio files in data/genres_30sec/reggae with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.77it/s]
Processing audio files in data/genres_30sec/reggae with dim 7 and hop 6:
100%|      | 100/100 [00:32<00:00,  3.08it/s]
Processing audio files in data/genres_30sec/reggae with dim 8 and hop 7:
100%|      | 100/100 [00:33<00:00,  2.99it/s]
Processing audio files in data/genres_30sec/reggae with dim 9 and hop 8:
100%|      | 100/100 [00:37<00:00,  2.65it/s]
```

Rock

[74]:
```
folder_path = "data/genres_30sec/rock"
start = 4
end = 10
grid = 2
folder = "plots"
```

```
plot_graph(folder_path, start, end, grid, folder)
```

```
Processing audio files in data/genres_30sec/rock with dim 4 and hop 3:
100%|      | 100/100 [00:56<00:00,  1.76it/s]
Processing audio files in data/genres_30sec/rock with dim 5 and hop 4:
100%|      | 100/100 [00:44<00:00,  2.26it/s]
Processing audio files in data/genres_30sec/rock with dim 6 and hop 5:
100%|      | 100/100 [00:36<00:00,  2.74it/s]
Processing audio files in data/genres_30sec/rock with dim 7 and hop 6:
100%|      | 100/100 [00:32<00:00,  3.07it/s]
Processing audio files in data/genres_30sec/rock with dim 8 and hop 7:
100%|      | 100/100 [00:32<00:00,  3.05it/s]
Processing audio files in data/genres_30sec/rock with dim 9 and hop 8:
100%|      | 100/100 [00:36<00:00,  2.72it/s]
```