Task 4 - Audio AI - Guitar Clustering and Melody Transcription

Introduction

If you love the sound of the guitar, this task will be a lot of fun! This task consists of two subtasks:

1. You will need to write code to transcribe guitar audio into notes.
2. You should cluster audio by the guitar that was used for recording

1. Guitar Audio Transcription

Description:

You will be provided with a folder containing 4 audio files with a guitar playing. Your task is to write Python code that transcribes this music into notes. The output should be printed as a list of tuples where each tuple consists of (note_name, start_time_in_seconds, end_time_in_seconds).

Requirements:

- The main runnable file should be named audio_task_1.py.
- The script should take the path of the input audio file as an argument.
- The output should be a printed list of tuples.

Example:

```
Command to run the script: python audio_task_1.py path_to_audio_file.wav

Expected output [('A4', 0.001, 0.521), ('C#5', 0.5, 1.0), ('E5', 1.0, 1.5),
...]
```

2. Guitar Audio Clustering

Description:

In this task, you should use the same folder containing 4 audio files. Two different guitars were used to record two identical melodies. Your task is to write Python code that clusters these 4 audio files by the guitar that was used for the recording. Each cluster should contain two audio files recorded with the same guitar.

Requirements:

- The main runnable file should be named audio_task_2.py.
- The script should take the path to the folder containing the audio files as an argument.
- The output should be a printed list of tuples where each tuple contains the names of the audio files in each cluster.

Example:

```
Command to run the script: python audio_task_2.py path_to_audio_folder

Expected output: [('guitar1_file1.wav', 'guitar1_file2.wav'),
('guitar2_file1.wav', 'guitar2_file2.wav')]
```

Deliverables:

- python code with requirements.txt
- a pdf report describing all experiments, and the approach used for solving the task

Solution

Business Understanding

The goal of this project is to transcribe guitar recordings into musical notes and cluster audio files based on the guitar used for recording.

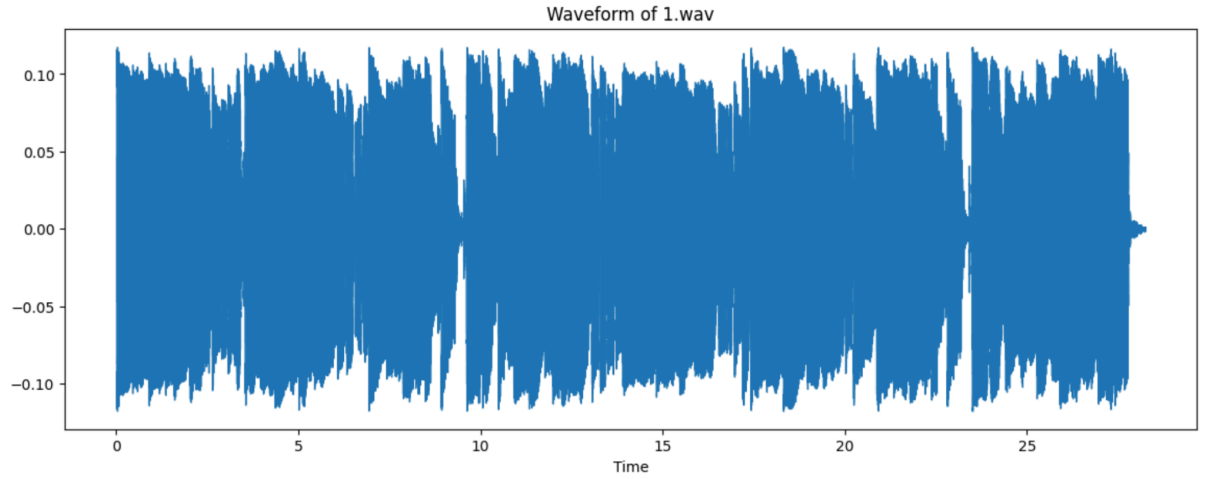
Requirements:

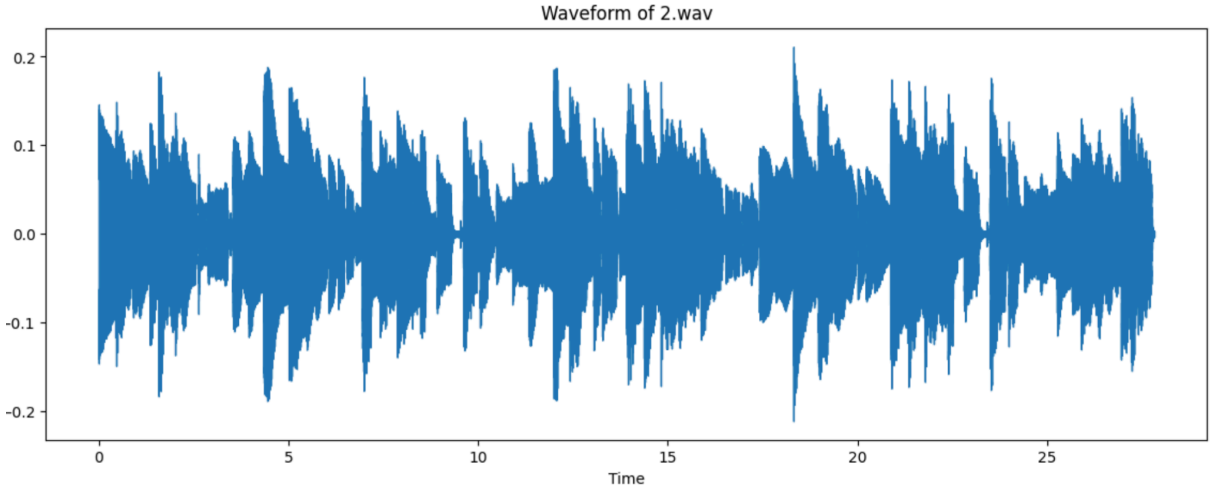
- The main runnable files should be named audio_task_1.py and audio_task_2.py
- The scripts should take the path of the input audio file as an argument.
- The output should be a printed list of tuples (for the second task - each tuple contains the names of the audio files in each cluster)

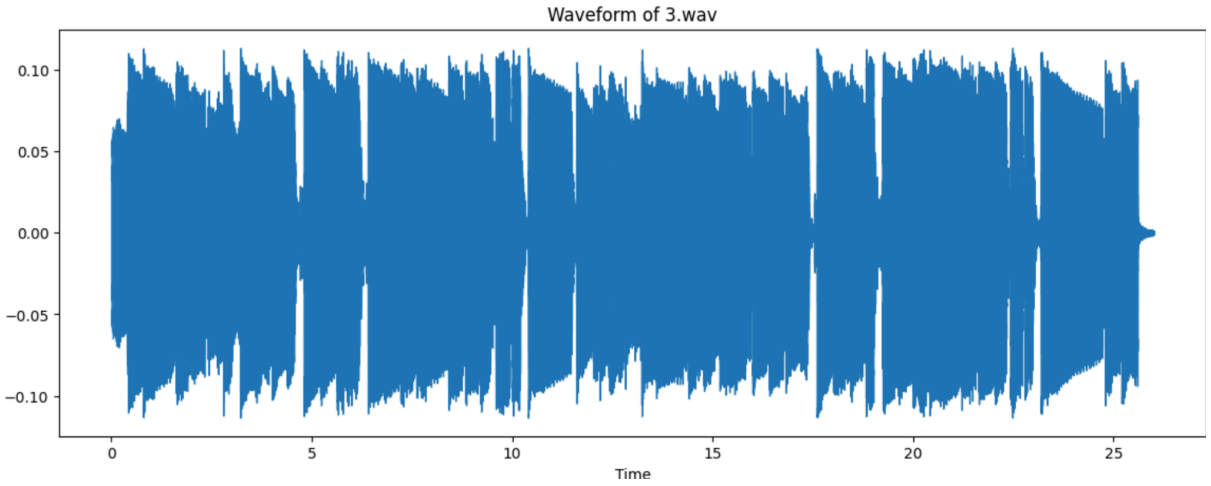
Data Understanding

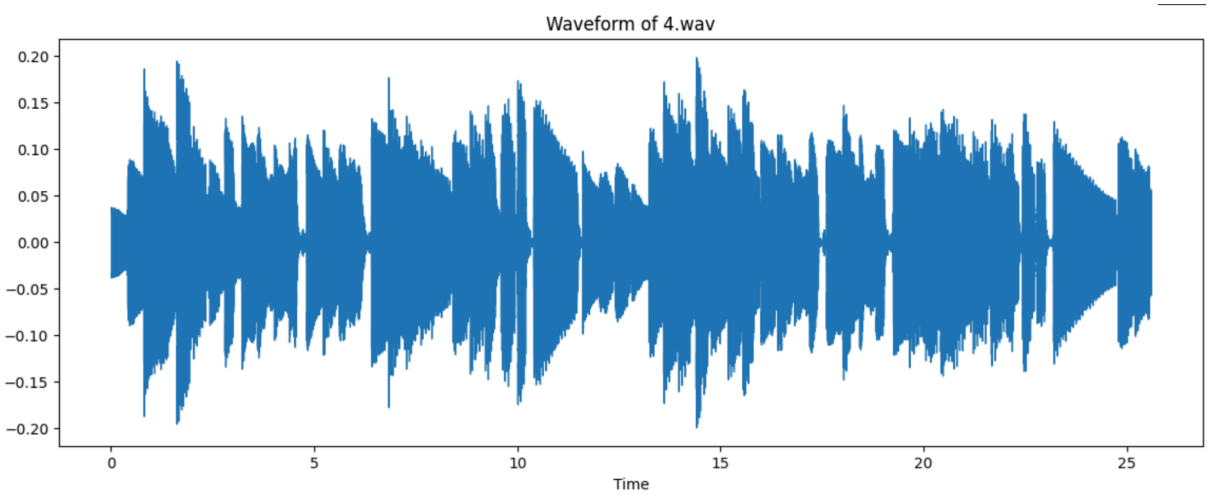
We started with a folder containing four audio files of guitar playing. Each file represents a recording of a melody. First and second audio files contain the same song recorded using different guitars as well as third and fourth.

Audio files waveforms:

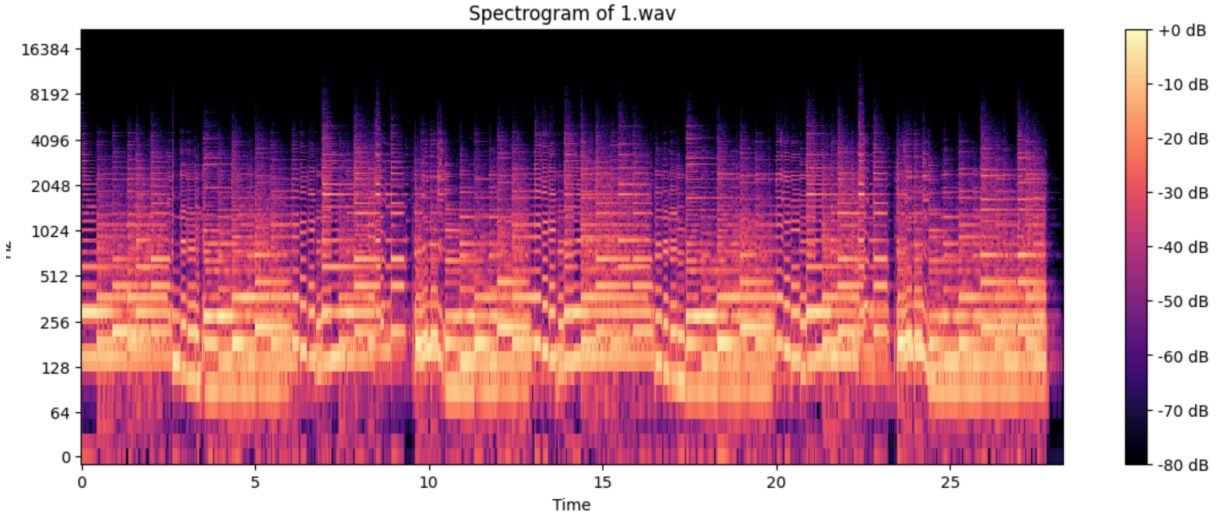

Waveform of 1.wav
0.10
0.05
0.00
−0.05
−0.10
0
5
10
15
20
25
Time

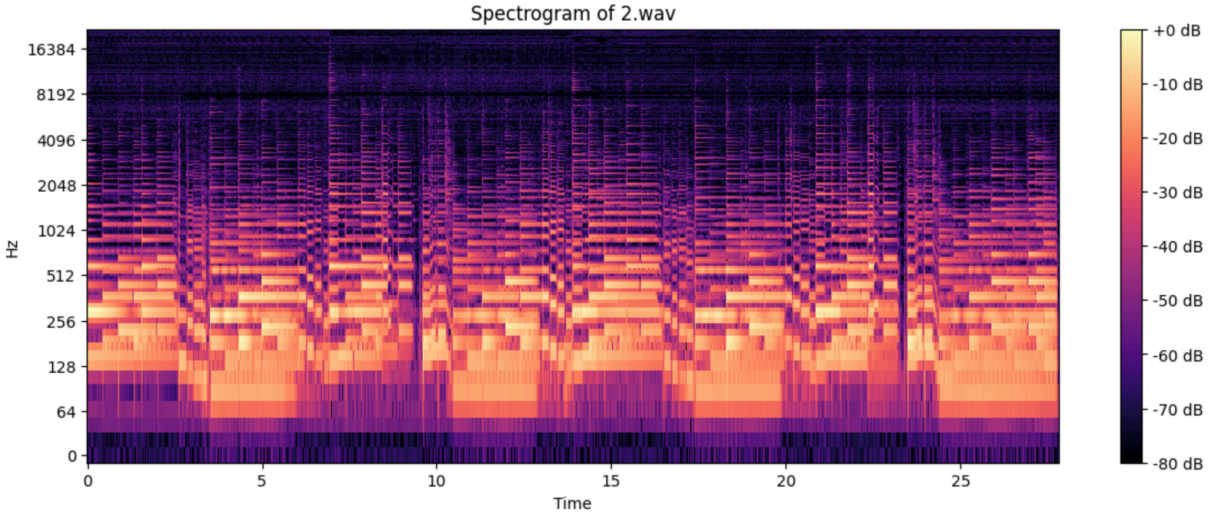

Waveform of 2.wav
0.2
0.1
0.0
-0.1
-0.2
0
5
10
15
20
25
Time

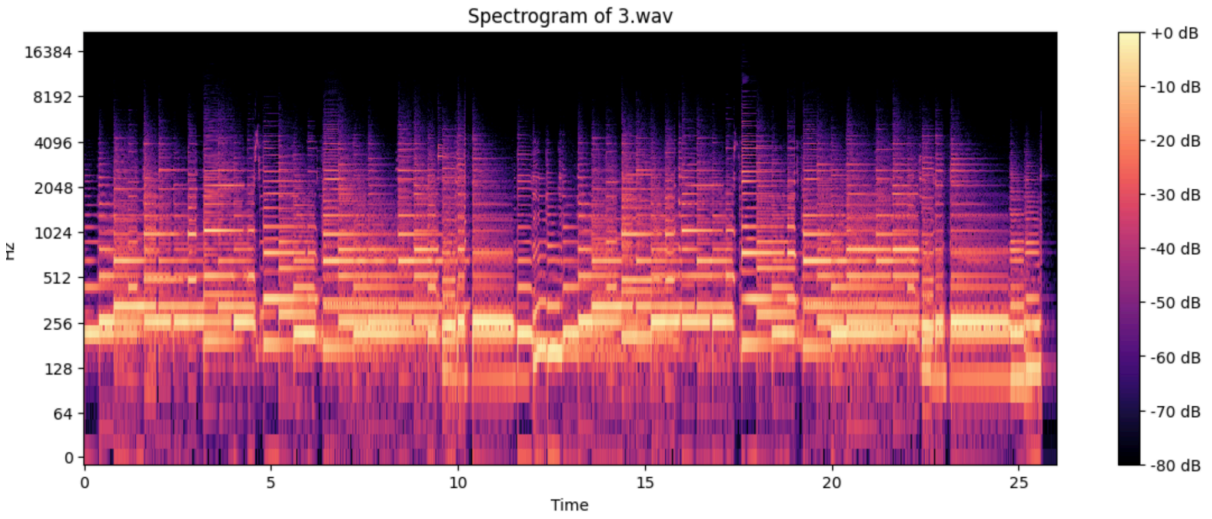

Waveform of 3.wav
0.10
0.05
0.00
−0.05
−0.10
0
5
10
15
20
25
Time

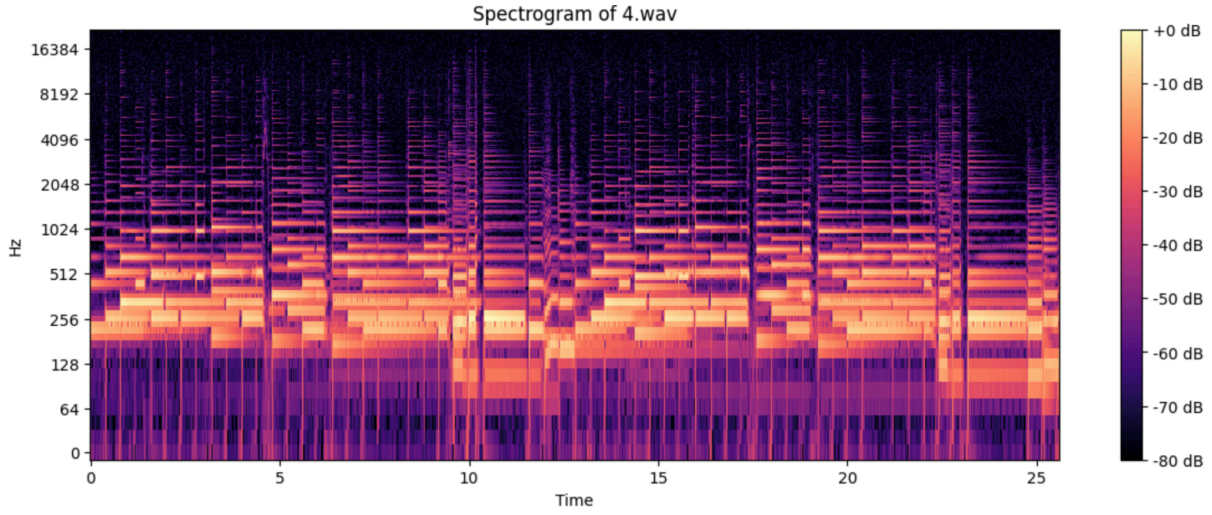

Waveform of 4.wav
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
5
10
15
20
25
Time

Audio files spectrograms:









We already can say that waveforms and spectrograms of second and fourth as well as first and third are similar.

Also, while exploring these audio files by listening to them I notice that second and fourth files contain some high-frequency noise.

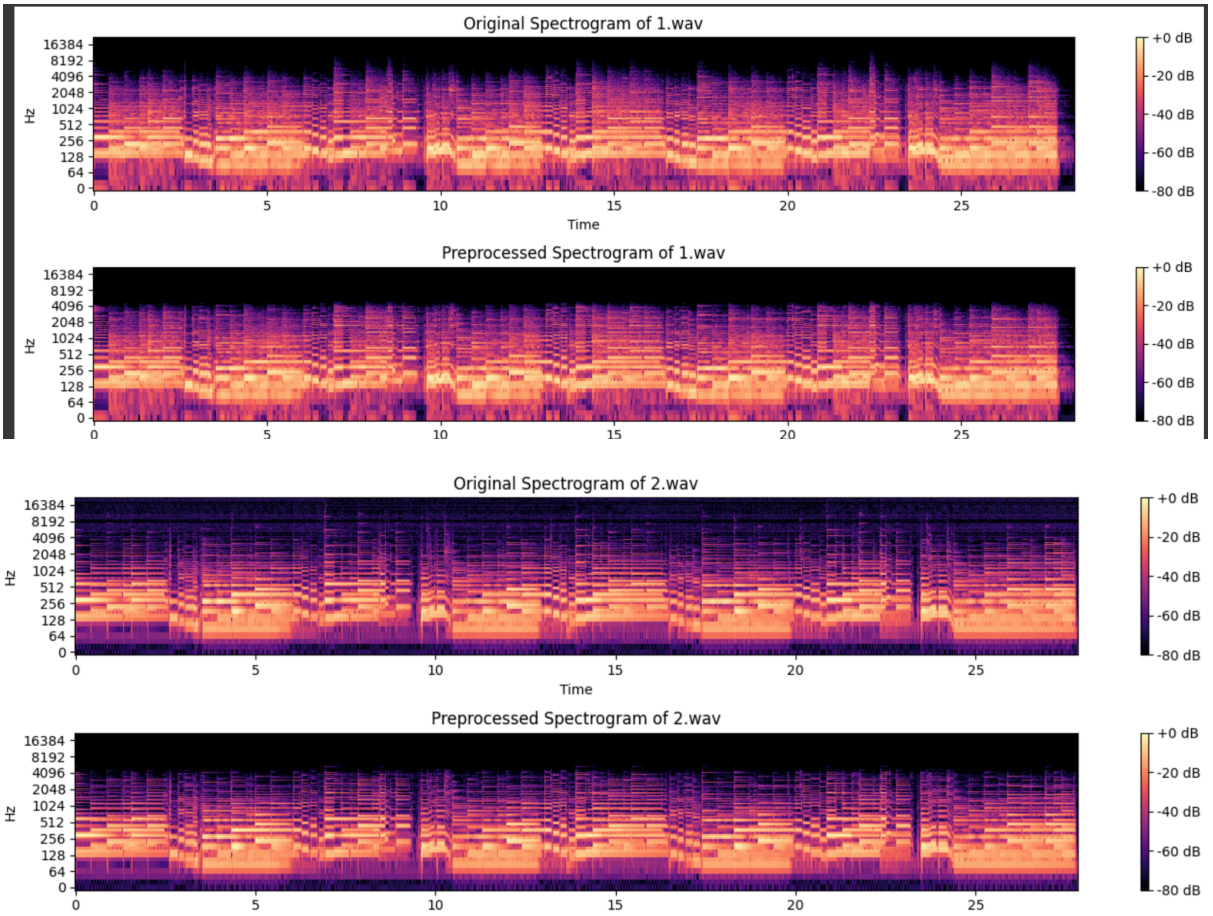
The next step was measuring the loudness of audio files.

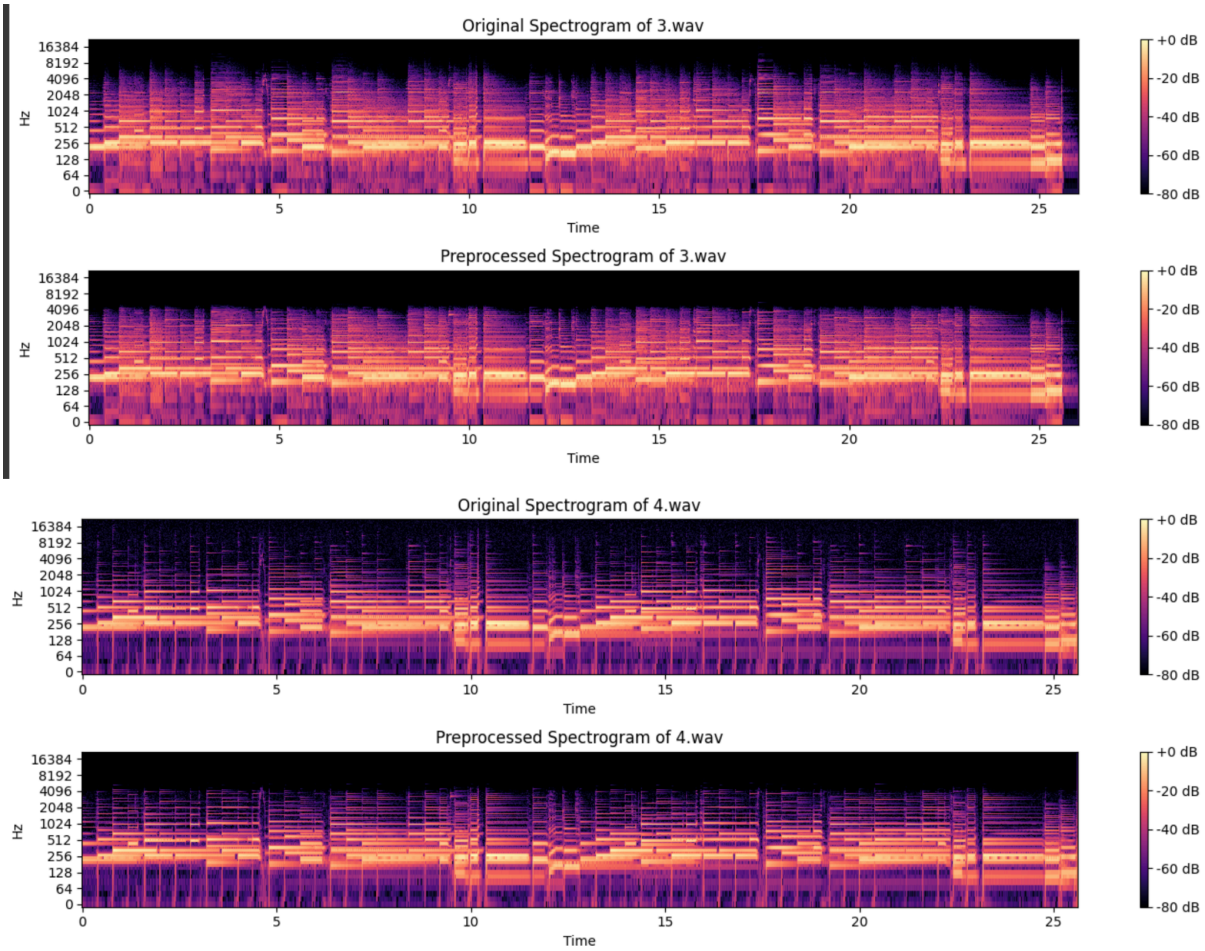
```
Record = 1.wav, Loudness = -27.03546668523165
Record = 2.wav, Loudness = -30.152273841003947
Record = 3.wav, Loudness = -27.039677423504482
Record = 4.wav, Loudness = -29.07794160070588
```

I thought that loudness normalization would not enhance my notes detecting algorithm whereas it can reduce cluster predicting ability, because the loudness can help us separate one guitar from another, so I skipped this step and moved on to data preprocessing.

Data Preparation

Considering I noticed the high-frequency noise I decided to use low-pass filtering to remove the high-frequency components of a signal, allowing only the low-frequency components to remain.





As a result my recording became pure, so I could move on to creating algorithms for first and second tasks.

Modeling

Task 1

The main idea was to identify the start times (onsets) of musical notes using the harmonic component and then extract the pitch and magnitude of the signal around each onset frame.

Steps

1. The filtered audio was normalized and processed to separate harmonic and percussive components.
2. Onset detection identified the starting points of notes.
3. Pitches and magnitudes were extracted, and spectral smoothing was applied to stabilize these estimates.
4. Notes were identified by analyzing the pitch and magnitude around each onset frame.

5. The detected notes were converted to note names (C4, G#3, …) and structured into a transcription format.
6. Spurious detections, such as notes of very short duration, were filtered out to ensure the accuracy and reliability of the transcription. (finished experiments with min_duration=0.2)
7. The final transcription was printed as a list of tuples, with each tuple containing the note name, start time, and end time.

Task 2

Here the main idea was to extract such features, which would help us distinguish the guitars, not songs.

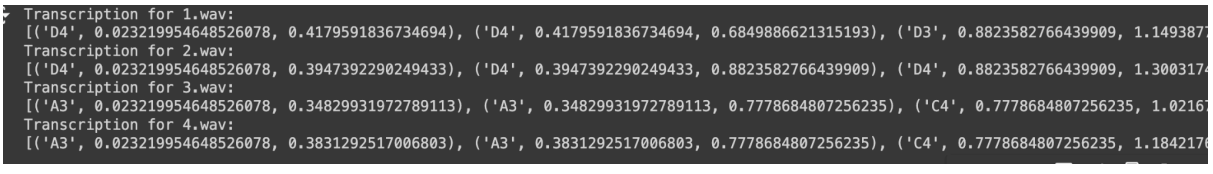
Feature Engineering

1. Spectral Contrast - measures the difference in amplitude between peaks and valleys in a sound spectrum, what may be useful for distinguishing tonal qualities and resonances of different guitars.
2. Zero-Crossing Rate - measures the rate at which the signal changes sign, capturing the noisiness and texture of the sound. It may be useful for identifying differences in playing style and guitar string characteristics.
3. Mel-Frequency Cepstral Coefficients - are one of the most commonly used features in speech and audio processing. They capture the power spectrum of a sound.

After I defined core features, modeling part was about using KMeans algorithm.
I combined my features taking mean of calculated characteristics and made clustering using this vector. Alternatively, dimensionality reduction techniques may be used.

Evaluation

Task 1



Evaluation can be done to compare outputs with true results.
I tried to compare outputs with tabs from songsterr

Stairway To Heaven - https://www.songsterr.com/a/wsa/led-zeppelin-stairway-to-heaven-tab-s27
Under The Bridge - https://www.songsterr.com/a/wsa/red-hot-chili-peppers-under-the-bridge-tab-s99,

but I am not sure that it is a good way of evaluating our results, because of the different ways of playing these songs.

So I was choosing the best model based on how similar were onsets and notes through all songs.
For example,

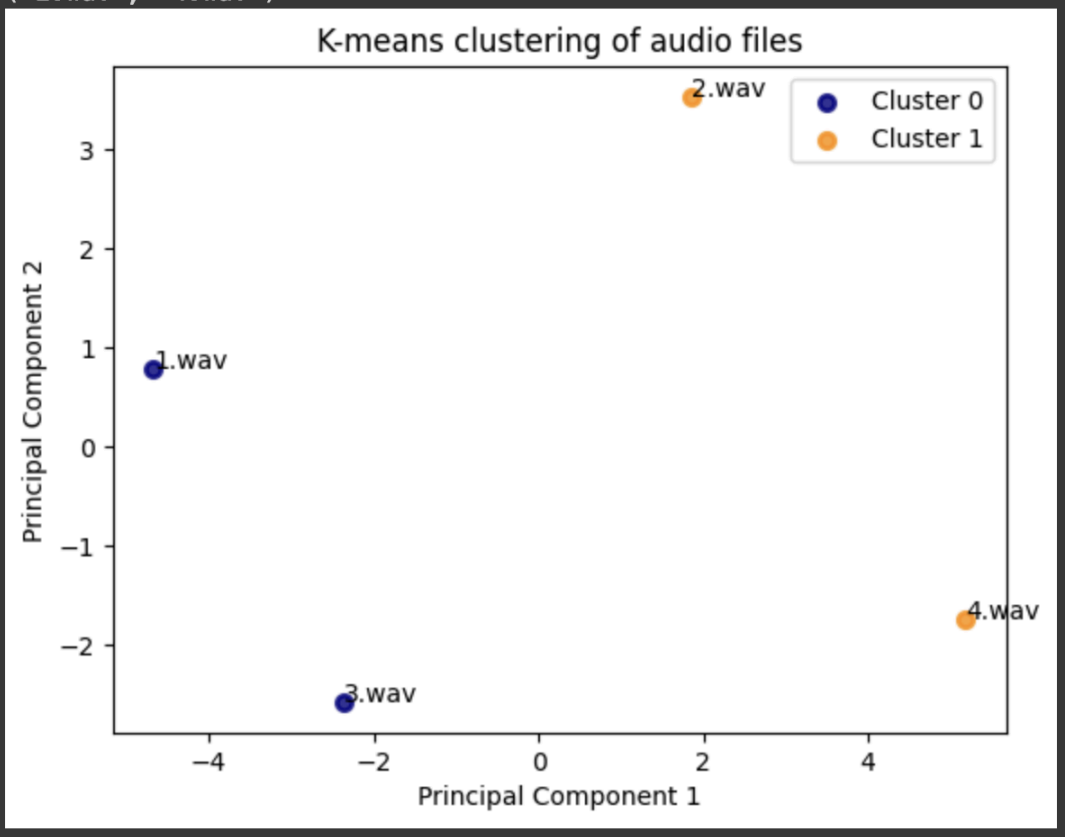
```
Transcription for 1.wav:
[('D4', 0.023219954648526078, 0.4179591836734694),
('D4',0.4179591836734694, 0.6849886621315193),
('D3', 0.8823582766439909, 1.1493877551020408),
('A3', 1.3351473922902495, 1.5673469387755101),

Transcription for 2.wav:
[('D4', 0.023219954648526078, 0.3947392290249433),
('D4',0.3947392290249433, 0.8823582766439909),
('D4', 0.8823582766439909, 1.3003174603174603),
('A3', 1.3003174603174603, 1.5209070294784581),
```

We see that over all onsets are pretty similar and detected notes are the same.
Of course, It is certainly not the best way of evaluating my transcription and it can be improved by, for example, restoring the songs from a given transcription and then evaluating it just by listening. Also can make up some formal metric, which will tell us whether our transcription is good compared to other open source transcriptions etc. But for now I leave it as it is.

Task 2

```
plot_clusters(features, labels, file_names)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Fu
  warnings.warn(
('1.wav', '3.wav')
('2.wav', '4.wav')
```



Here we can see that our clustering does not work clearly in terms of pure similarity between objects of one class. It works more in the way of finding one similar class and the rest are treated as outliers that do not come into this particular class.
Let's look at some clusterization metrics, which do not require knowing of true class labels.

```
Silhouette Score: 0.32
Davies-Bouldin Index: 0.76
Calinski-Harabasz Index: 3.40
```

Silhouette Score:
Measures the similarity of an object to its own cluster compared to other clusters.
Values close to 1 indicate well-separated clusters, values close to 0 indicate overlapping clusters, and negative values indicate incorrect clustering.

Davies-Bouldin Index:
Measures the average similarity ratio of each cluster with its most similar cluster.
Lower values indicate better clustering performance.

Calinski-Harabasz Index:

Measures the ratio of between-cluster dispersion to within-cluster dispersion.
Higher values indicate better-defined clusters.

Our metrics analysis:

Silhouette Score:
A score of 0.32 indicates that there might be some overlap between clusters or that the clusters are not very dense.

Davies-Bouldin Index:
A value of 0.76 is relatively low, suggesting that the clusters are compact and well-separated

Calinski-Harabasz Index:
A value of 3.40 is quite low. Higher values indicate that clusters are well-separated by dense points. This suggests that the clusters might not be very well-defined.

Overall, out clustering certainly can be improved firstly by more advanced feature engineering and some algorithm tuning(maybe using DBSCAN etc.) ,which might lead to better class separation. Nevertheless, I got the result, which I made up with after listening to my audio files and looking at waveforms and spectrograms. This time I can afford to do this, but, in general, much advanced feature engineering is needed.

Deployment

Solutions are implemented as scripts called audio_task_1.py and audio_task_2.py accordingly. They can be launched from terminal using

```
python audio_task_1.py <path_to_audio_file>
```

```
/Users/ivanbashtovyi/miniforge3/envs/audio_env/bin/python /Users/ivanbashtovyi/Documents/It-Jim/clusteting_transcription/audio_task_1.py
(audio_env) (base) ivanbashtovyi@MacBook-Pro-Ivan clusteting_transcription % /Users/ivanbashtovyi/miniforge3/envs/audio_env/bin/python /Users/ivanba
shtovyi/Documents/It-Jim/clusteting_transcription/audio_task_1.py
Usage: python audio_task_1.py <path_to_audio_file>
(audio_env) (base) ivanbashtovyi@MacBook-Pro-Ivan clusteting_transcription % python audio_task_1.py /Users/ivanbashtovyi/Documents/It-Jim/clusteting
_transcription/recordings/2.wav
[('D4', 0.023, 0.395), ('D4', 0.395, 0.882), ('D4', 0.882, 1.3), ('A3', 1.3, 1.521), ('F♯4', 1.521, 1.95), ('D4', 1.95, 2.485), ('A4', 2.485, 2.612)
, ('C♯4', 2.682, 2.868), ('B2', 2.868, 3.065), ('A2', 3.065, 3.437), ('G♯3', 3.518, 3.936), ('C♯4', 3.936, 4.319), ('C♯4', 4.319, 4.783), ('F♯4', 4.
783, 4.992), ('F♯4', 4.992, 5.399), ('A♯3', 5.399, 6.06), ('F♯4', 6.06, 6.304), ('E3', 6.304, 6.536), ('D4', 6.536, 6.734), ('D3', 6.734, 6.92), ('D
4', 6.92, 7.407), ('D5', 7.407, 7.628), ('D4', 7.628, 7.825), ('D4', 7.825, 8.243), ('A4', 8.243, 8.464), ('A4', 8.464, 8.603), ('E3', 8.754, 8.893)
, ('D5', 8.893, 9.323), ('A3', 9.323, 9.451), ('D4', 9.59, 9.764), ('F♯3', 9.764, 9.868), ('A2', 9.868, 10.008), ('E3', 10.008, 10.252), ('F♯1', 10.
472, 10.879), ('C♯4', 10.879, 11.297), ('C♯4', 11.297, 11.703), ('F♯4', 11.703, 11.97), ('C♯4', 11.97, 12.399), ('A♯4', 12.399, 13.003), ('F♯4', 13.
003, 13.259), ('E3', 13.259, 13.479), ('D4', 13.479, 13.677), ('A2', 13.677, 13.851), ('B3', 13.851, 14.153), ('D4', 14.153, 14.315), ('D4', 14.315,
 14.605), ('F♯4', 14.605, 14.803), ('F♯4', 14.803, 15.232), ('A4', 15.232, 15.453), ('A4', 15.453, 15.848), ('D5', 15.848, 16.463), ('C♯3', 16.509,
16.742), ('F♯3', 16.742, 16.951), ('A3', 16.951, 17.171), ('G♯3', 17.171, 17.403), ('G♯3', 17.403, 17.821), ('C♯4', 17.821, 18.286), ('C♯4', 18.286,
 18.704), ('F♯4', 18.704, 18.936), ('F♯4', 18.936, 19.365), ('A♯3', 19.365, 19.923), ('F♯4', 19.923, 20.213), ('E3', 20.213, 20.434), ('D3', 20.434,
 20.666), ('D4', 20.666, 20.84), ('F3', 20.84, 21.119), ('D4', 21.119, 21.304), ('D4', 21.304, 21.56), ('D4', 21.56, 21.769), ('D3', 21.769, 22.14),
 ('A4', 22.14, 22.361), ('A4', 22.361, 22.558), ('A3', 22.663, 22.767), ('D4', 22.767, 23.22), ('D4', 23.22, 23.324), ('F♯4', 23.441, 23.766), ('G3'
, 23.766, 23.951), ('E3', 23.951, 24.23), ('D3', 24.265, 24.381), ('C♯4', 24.381, 24.811), ('C♯4', 24.811, 25.228), ('C♯4', 25.228, 25.646), ('F♯4',
 25.646, 25.879), ('C♯4', 25.879, 26.285), ('A♯4', 26.285, 26.494), ('A♯4', 26.494, 26.935), ('A♯4', 26.935, 27.156), ('A♯4', 27.156, 27.388), ('A♯4
', 27.388, 27.826)]
(audio_env) (base) ivanbashtovyi@MacBook-Pro-Ivan clusteting_transcription %
```

or

```
python audio_task_2.py <path_to_audio_folder>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(audio_env) (base) ivanbashtovyi@MacBook-Pro-Ivan clusteting_transcription % python audio_task_2.py /Users/ivanbashtovyi/Documents/It-Jim/clusteting
_transcription/recordings
[('1.wav', '3.wav'), ('4.wav', '2.wav')]
(audio_env) (base) ivanbashtovyi@MacBook-Pro-Ivan clusteting_transcription %
```

Conclusion

Overall, I liked this task, it was really interesting to look at these songs from another side and dive into the world of signal processing, especially when you haven't been listening to Stairway to Heaven and Under The Bridge for a long time. Thank you for this opportunity!

Source code of exploring different approaches - https://colab.research.google.com/drive/1EYGnjOj3VhaTqy9dji6dgHYSG0Z5v8o1?usp=sharing