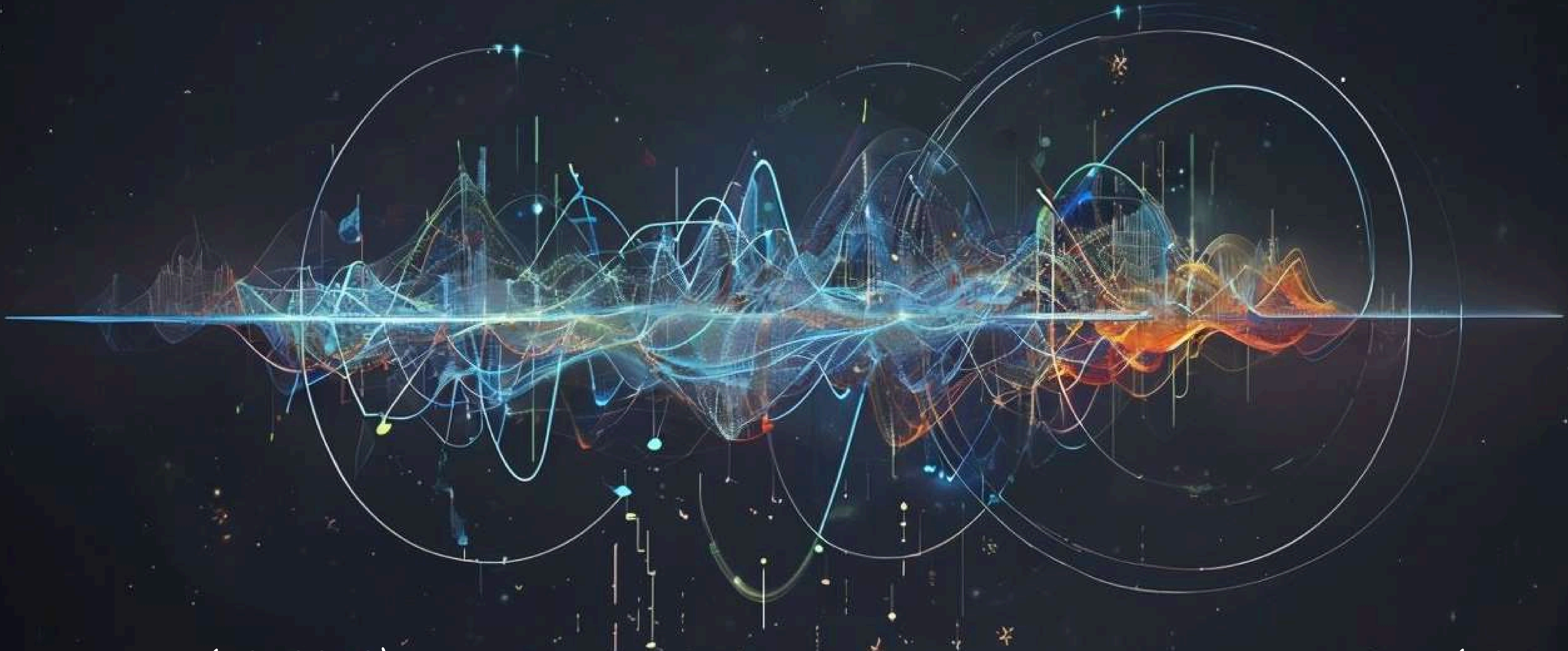
VOICEPRINTS UNVEILED



PANAV SHAH (23B3323)

AADI BHARATIA (23B1858)

SANIYA KHINVASARA (23B1268)

YASH PALWE (23B1823)

DS 203 Project

INTRODUCTION

- This project focuses on analyzing a dataset of 115 songs using **Mel-Frequency Cepstral Coefficients (MFCCs)** to group the songs by their genres and artists
- We aim to identify and classify specific songs, such as the **Indian National Anthem**, songs by iconic artists like **Asha Bhosle**, **Kishore Kumar**, and **Michael Jackson** and **Marathi Bhavgeet** and **Lavni**
- The analysis involves applying **machine learning techniques** to classify songs based on their vocal and instrumental characteristics

WHAT ARE **MFCC COEFFICIENTS?**

MFCCs are **Mel-Frequency Cepstral Coefficients.**

Mel-frequency cepstral coefficients (MFCCs) are a set of features that represent the spectral envelope of a sound signal. They are commonly used in speech recognition systems to analyze and model human voice characteristics

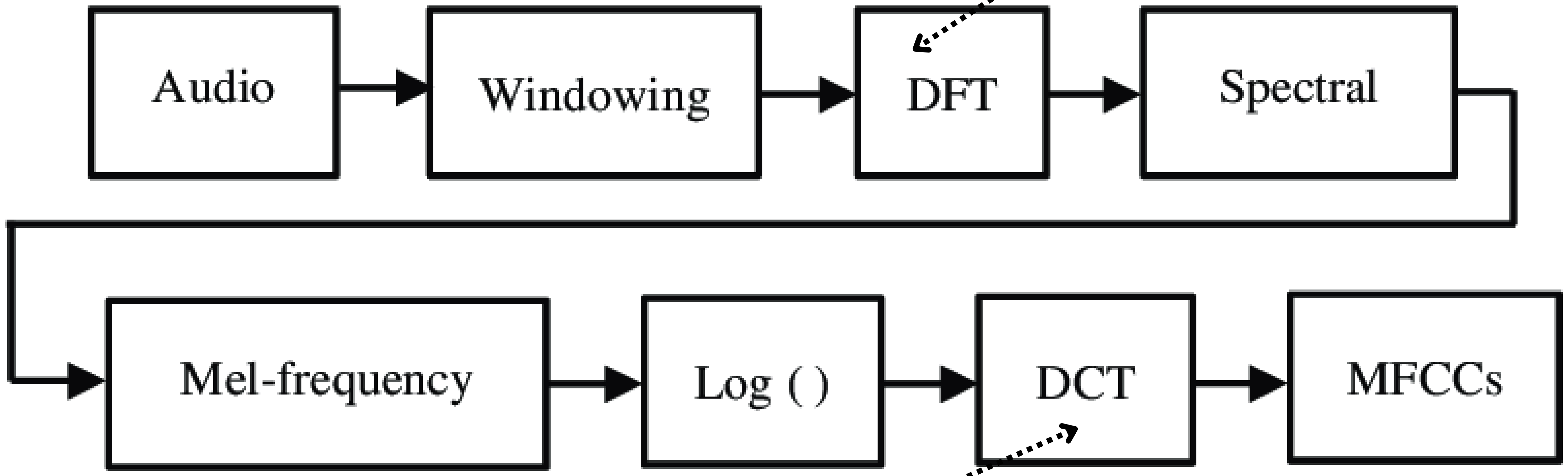
MFCCs tell us about the following:

1. **Frequency Content**
2. **Perceived Pitch and Timbre**
3. **Speech Patterns**

WHAT ARE **MFCC COEFFICIENTS?**

How are they **calculated**?

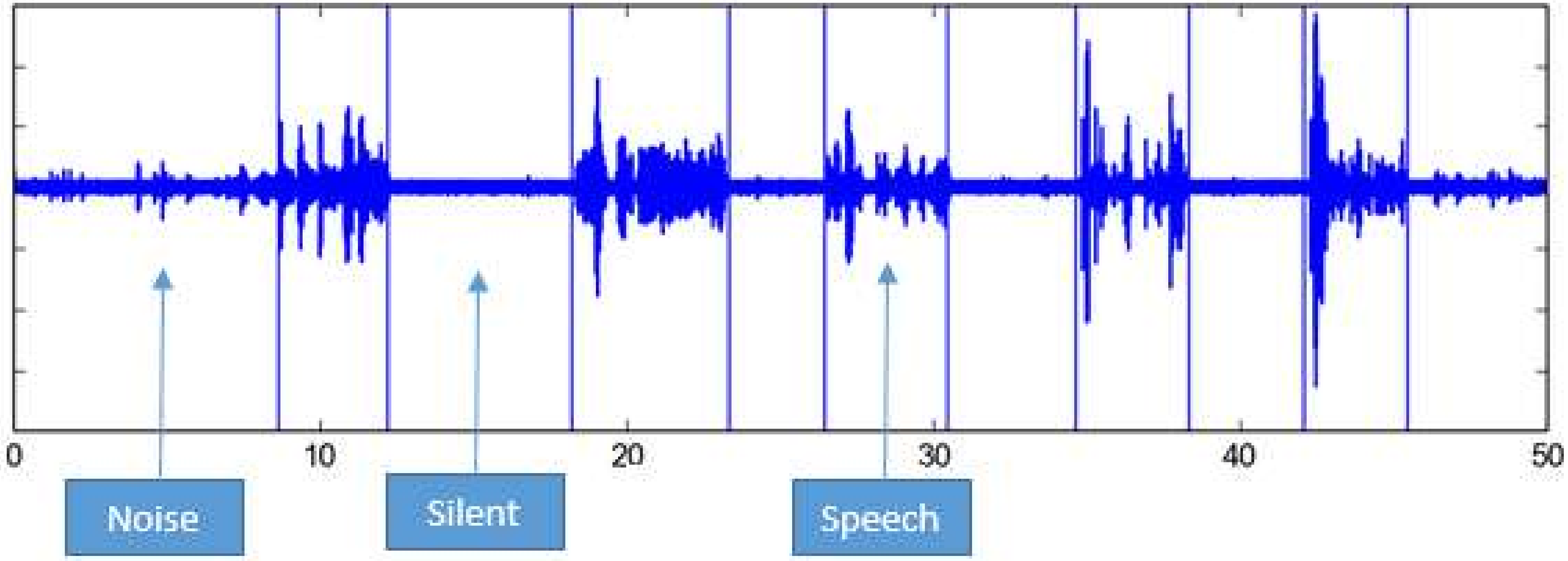
Discrete Fourier Transform



Discrete Cosine Transform

FILE SEGMENTATION

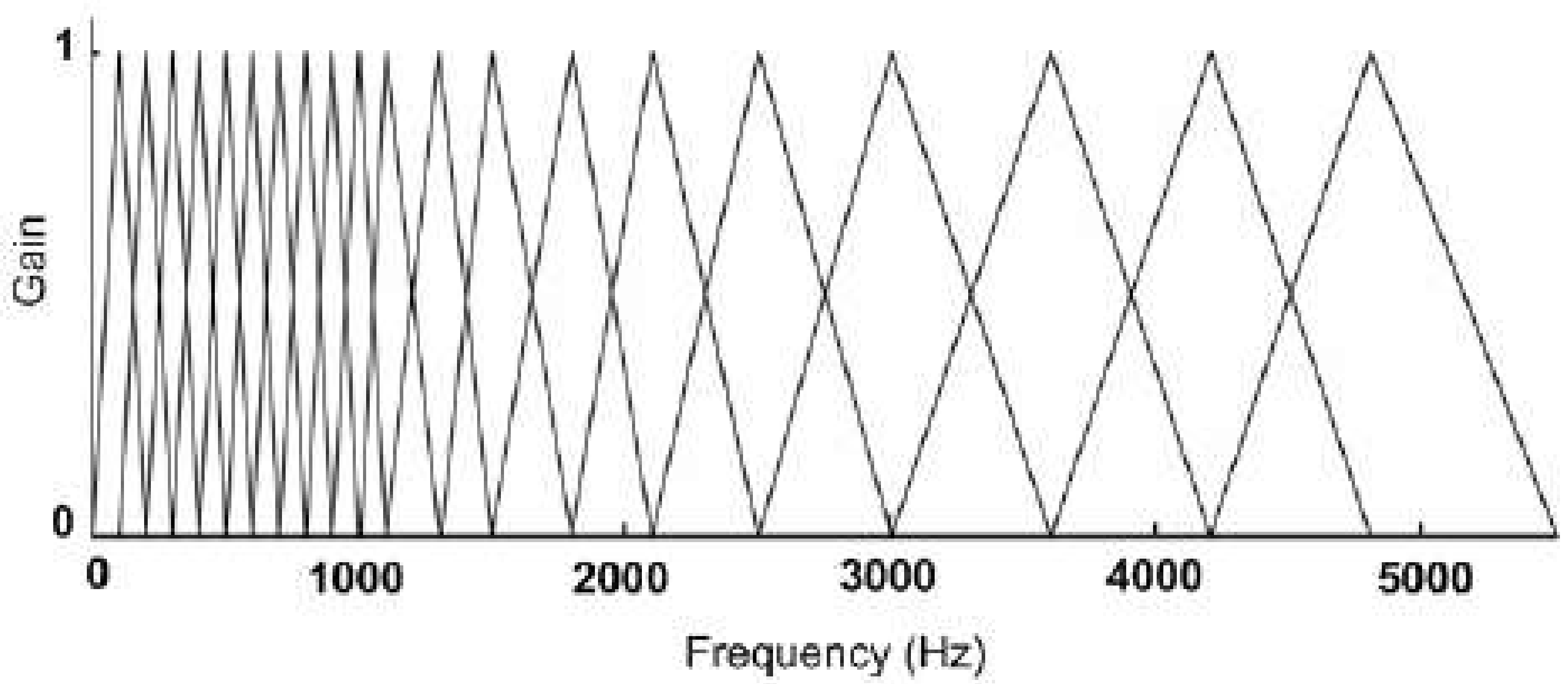
Step 1 : **Framing** and **Windowing** involves **segmenting** the signal into short frames and applying a window function to reduce signal **edge effects**.



FILE PROCESSING

Step 2 : Finding the **Fast Fourier Transform**, converting each frame to the **frequency** domain

Step 3 : Applies **triangular filters** spaced according to the **Mel scale** to approximate human hearing



Step 4 : The data is then **compressed** by taking the **log** of the power spectrum and applying **DCT**, resulting in MFCCs

LIMITATIONS OF MFCC

- Loss of **Temporal Information**
- Sensitivity to **Noise**
- Inflexibility with **Pitch Variations**
- Limited **Effectiveness** for Music and Non-Speech Sounds

The Overall Approach...

A brief overview of the thought process and steps involved behind solving the problems

Step I

- Downloaded the **MFCC zip files** and performed **EDA** on them; Decided on a **CNN model**

Step II

- **Downloaded** audio files from YouTube
- Converted the audio files into **CSV files** containing the respective MFCC coefficients
- **Preprocessed** the data, removed outliers

Step III

- Split the downloaded data into **training** and **test** data
- **Trained** the CNN model based on the training data

Step IV

- Calculated train and test data **performance metrics** for the model
- Fed the given data into the model and obtained the **grouped data**

EXPLORATORY DATA ANALYSIS

HEAT MAP

CORRELATION MATRIX

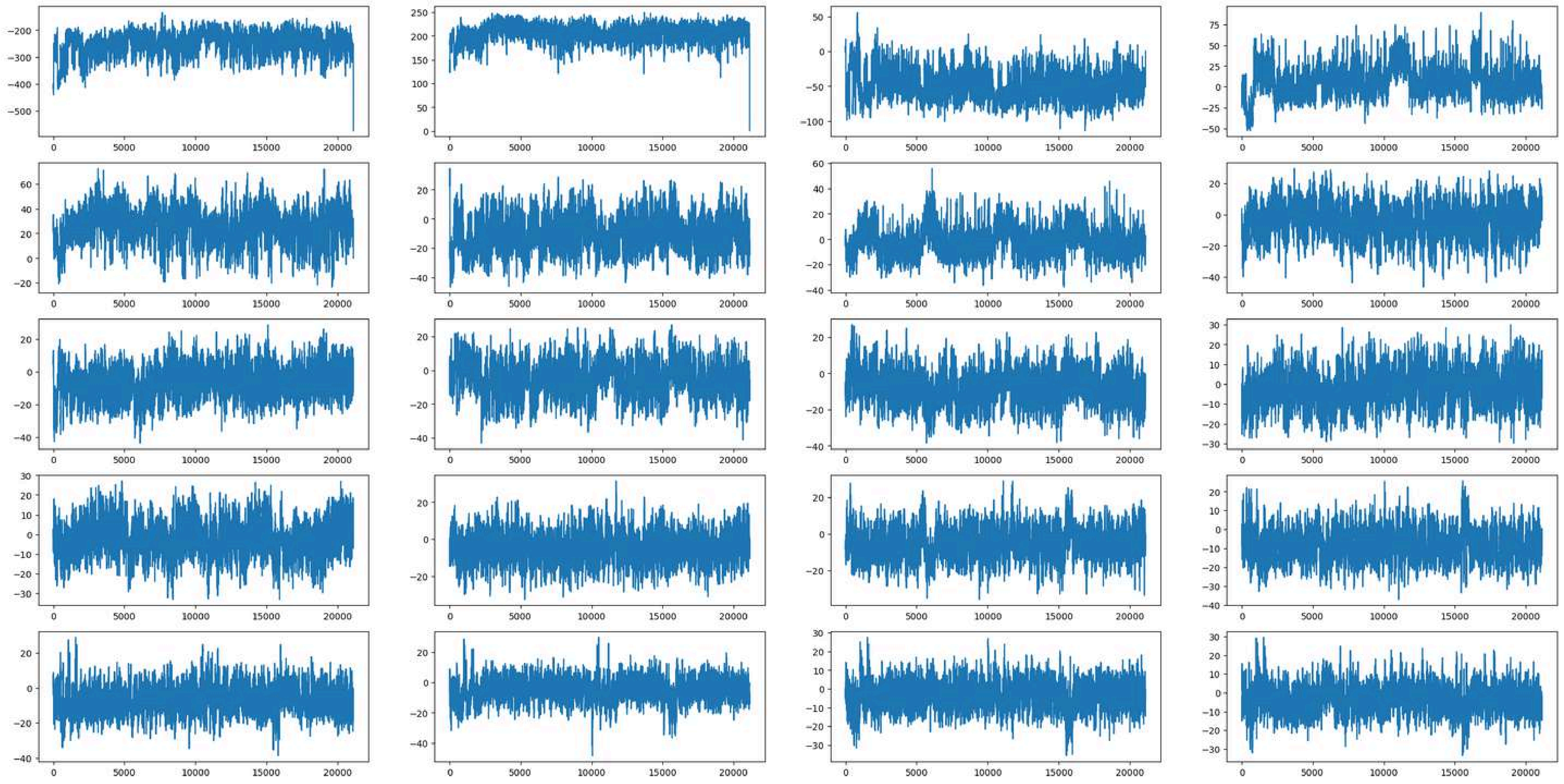
PRINCIPAL COMPONENT ANALYSIS

SCREE PLOT

TIME SERIES ANALYSIS

- Plots of each **component vs time**
- Uses a time scale where **1 second** approximately equals **86 units** on the x-axis
- The **first component** remains consistently **negative**, the **second** stays **positive**, and the remaining components fluctuate near **zero**

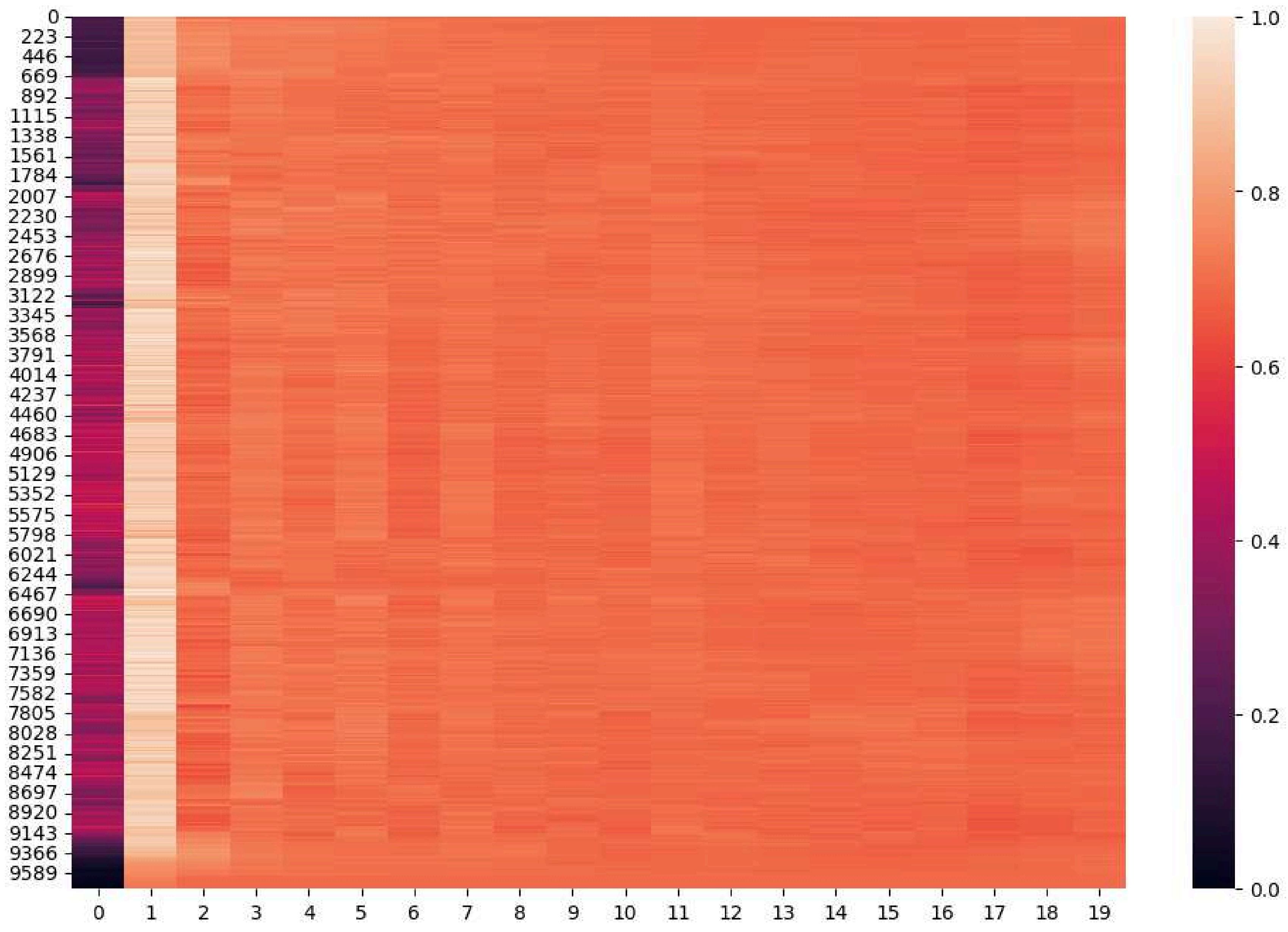
Each of the 20 components of the MFCC file



HEAT MAP

- The adjoining heatmap is of the **normalised data**
- It shows that the **lower coefficients** capture relatively **more variation**, as can be seen by the distinct stripes

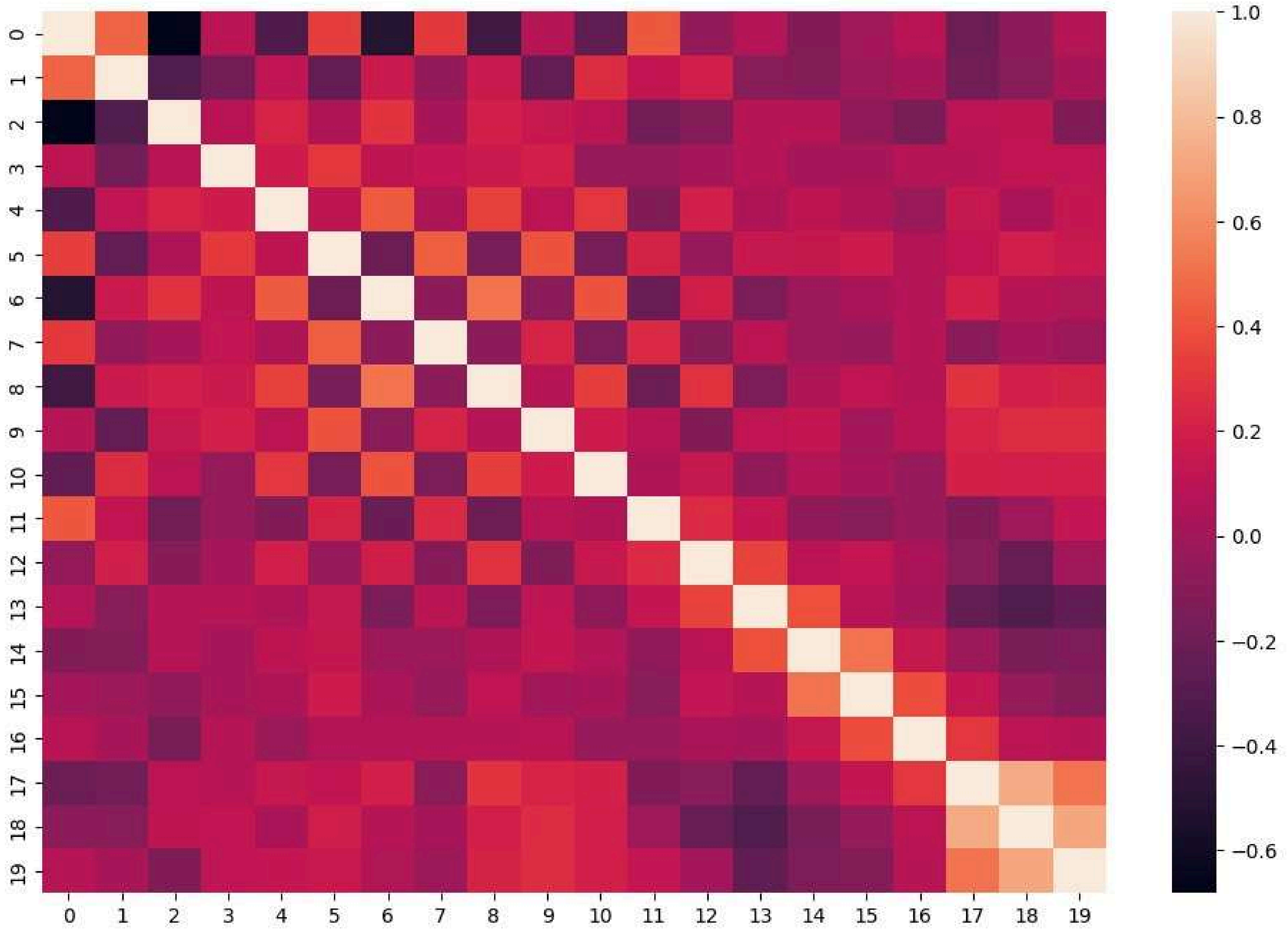
Heat map for one of the MFCC files



CORRELATION MATRIX

- Apart from the diagonal, the values in the correlation matrix is primarily between **-0.4** and **0.4**
- This tells us that there is very **less correlation** between the 20 features of the MFCC data

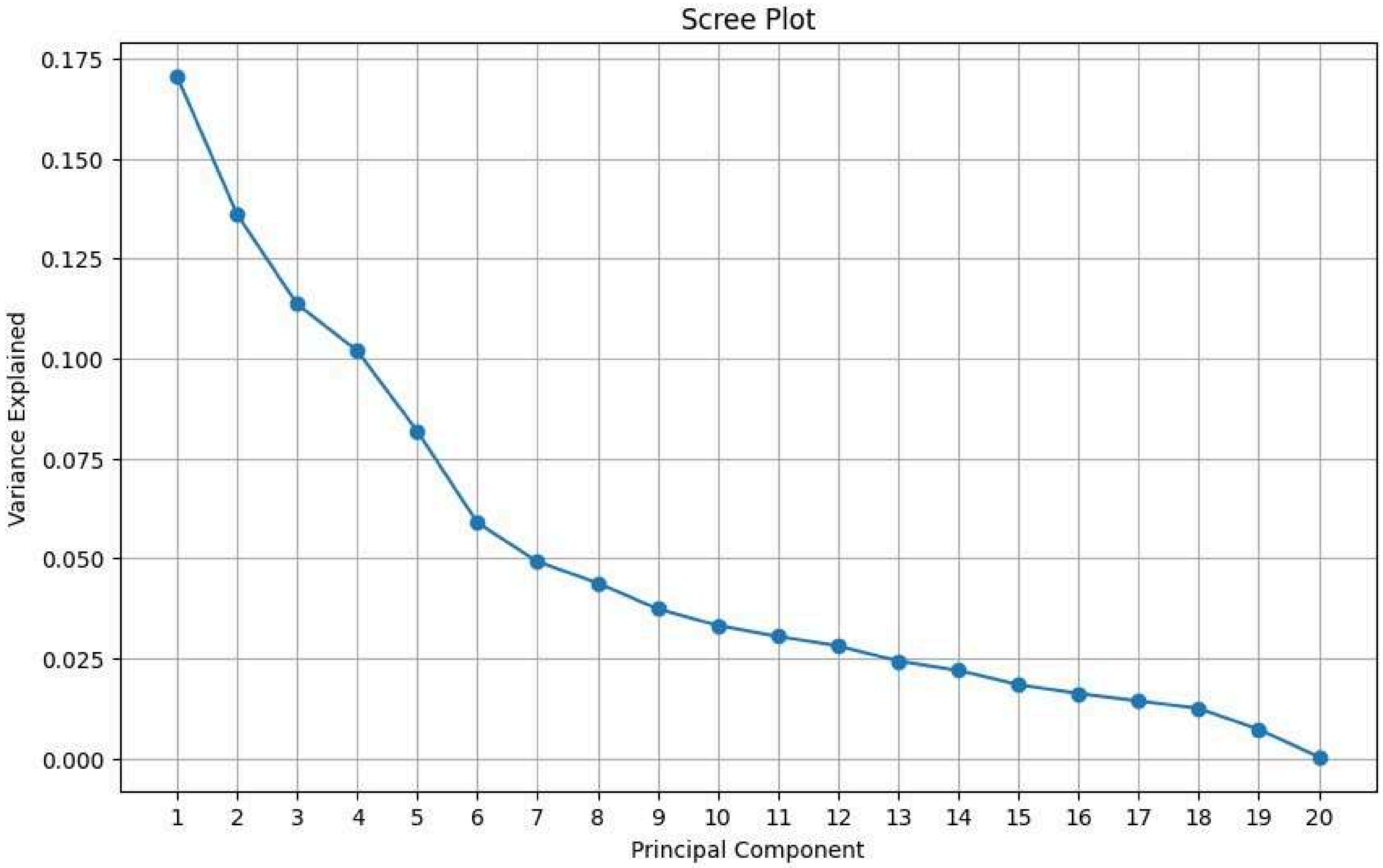
20 x 20 correlation matrix for the coefficients



SCREE PLOT

- The scree plot takes a **sudden turn** at principal component **6**
- This shows that considering around **5-6 principal components** is enough because after that the variance starts falling a lot

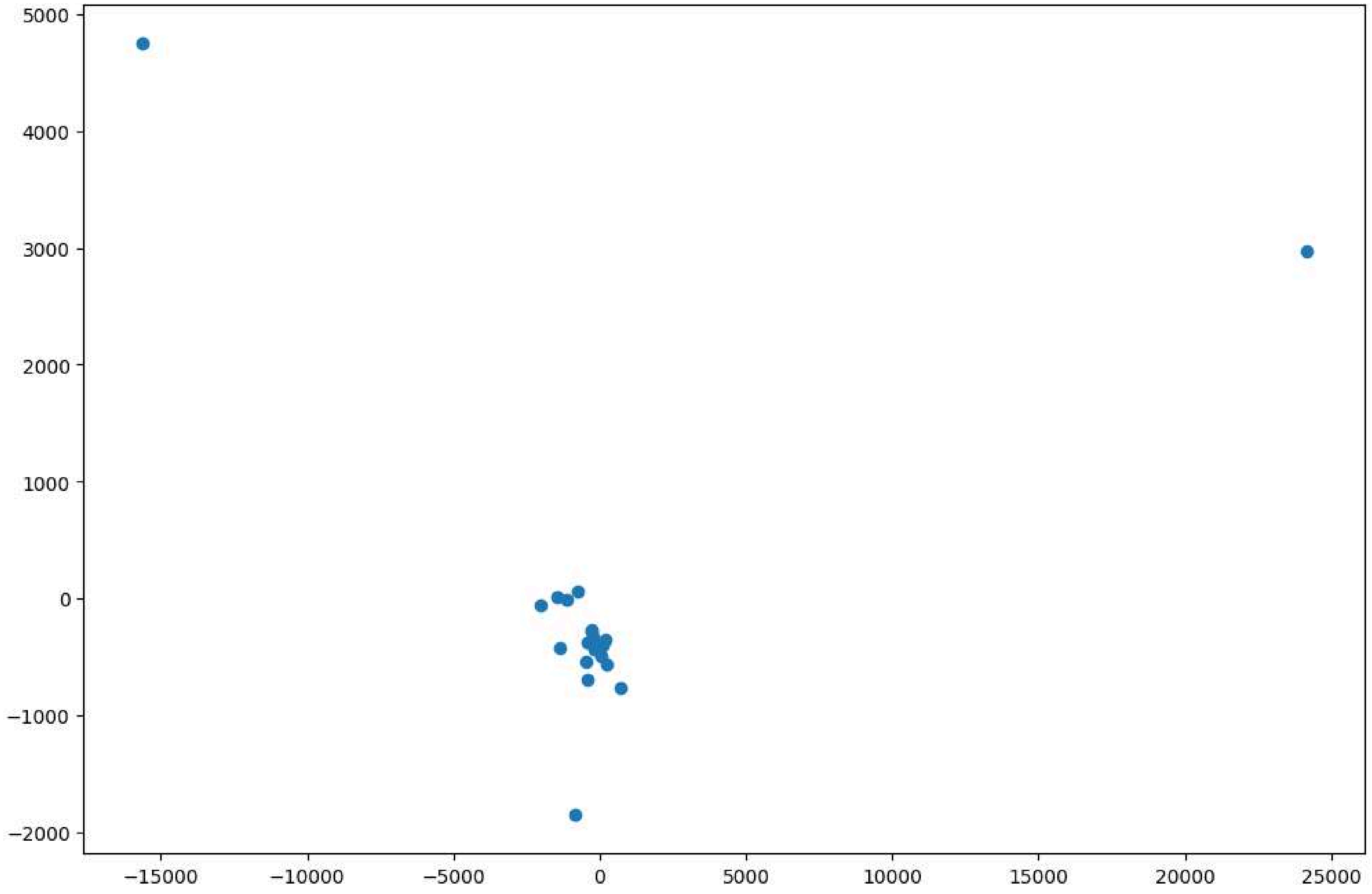
Scree Plot for cumulative variance vs number of principal components



PCA ANALYSIS

- The points around (25000,3000) and (–15000,5000) are particularly distant from the main cluster
- This might mean these features have a **significantly different variance or distribution** compared to the rest
- These outliers are the **first two components** of the given MFCC data

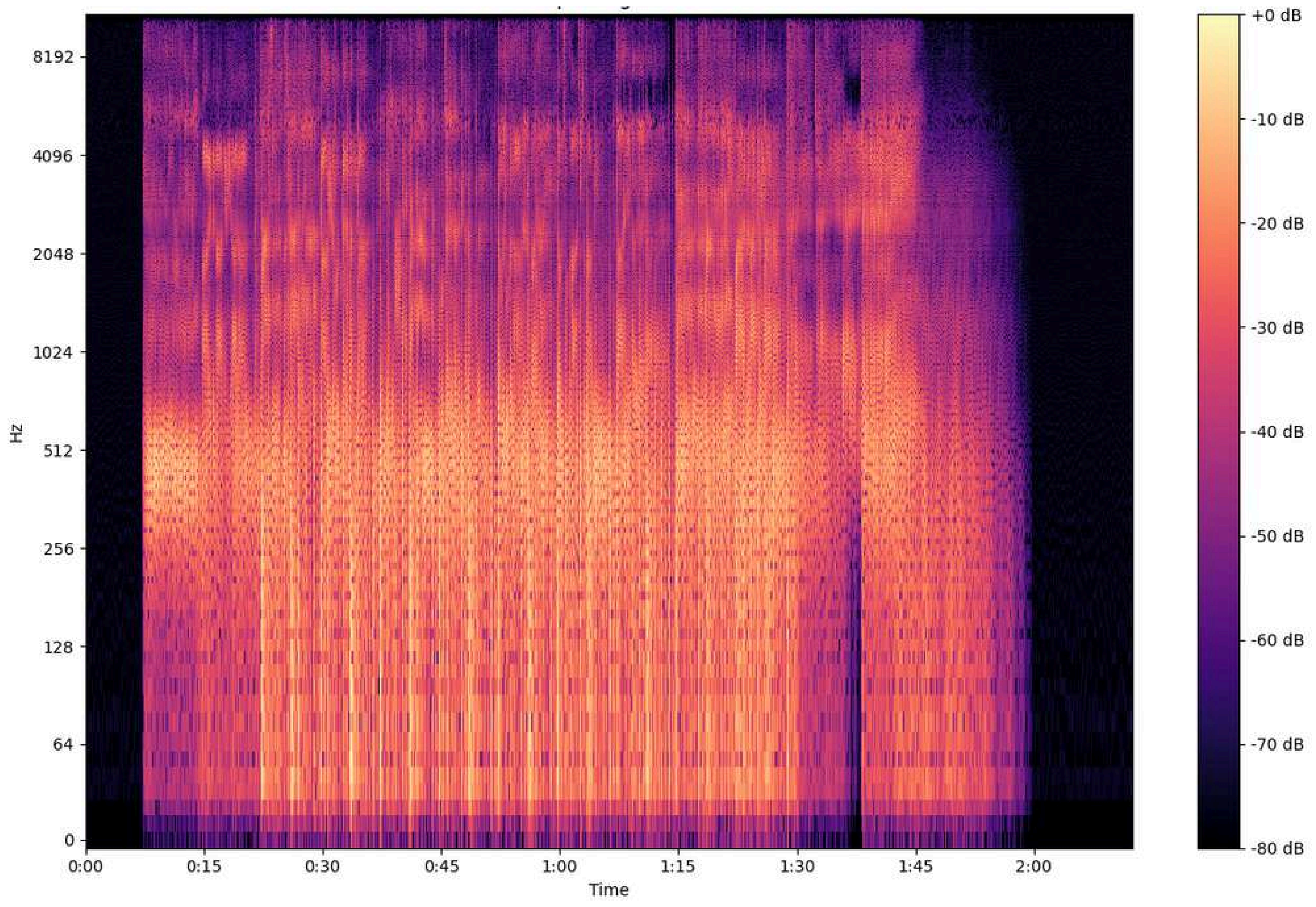
PC1 vs PC2 for each component



SPECTOGRAM

- We created a spectrogram to show the **frequency content** of audio signals over time, making it easier to see patterns and differences in the sounds
- This helps us identify **key features** for each song or genre

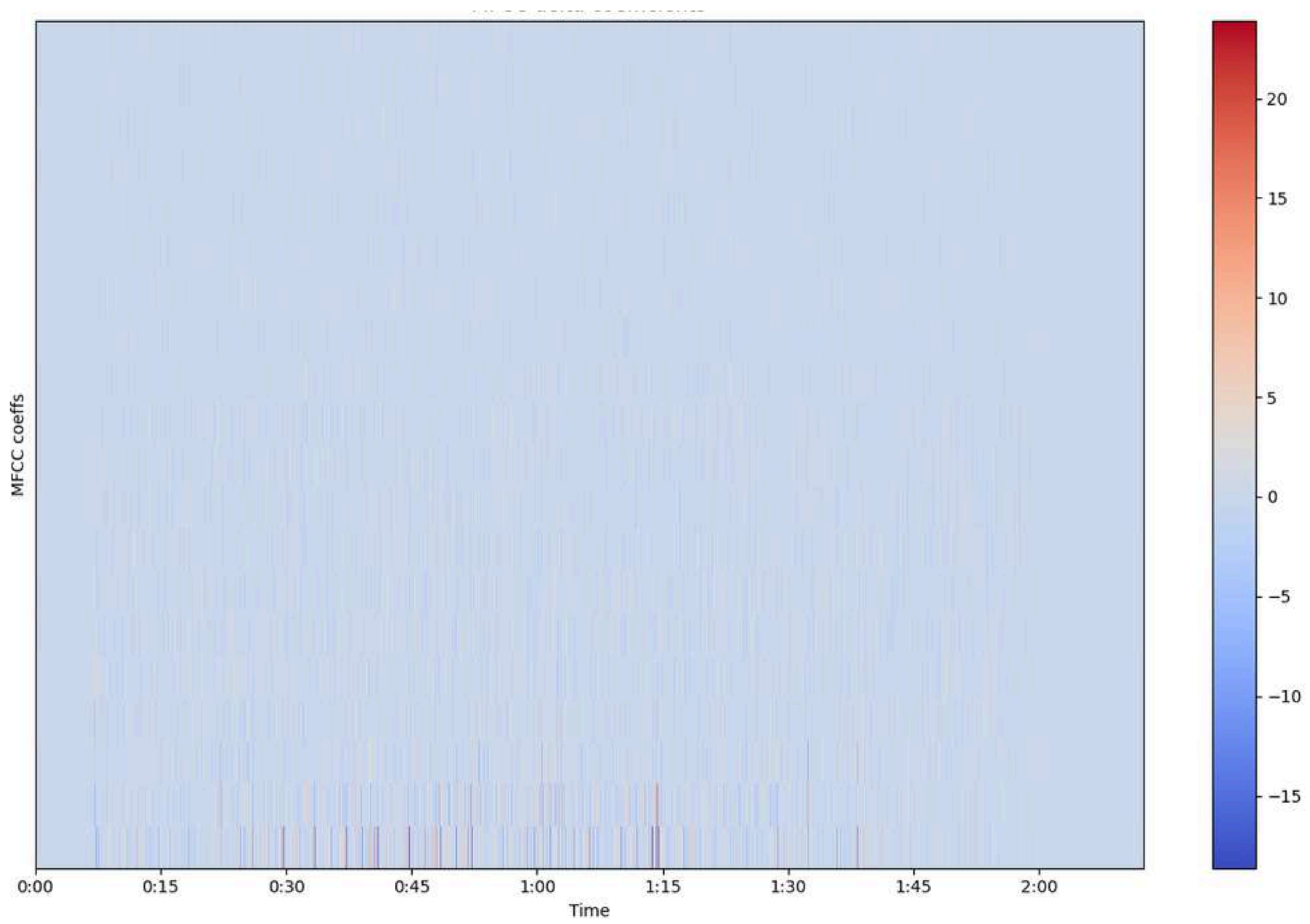
Spectogram of one of the audio files



DELTA COEFFICIENTS

- Since most of the heatmap is very light, it suggests that the delta coefficients are **close to zero** for much of the audio, meaning there are **few strong transitions** in MFCC values over time.

MFCC delta coefficients

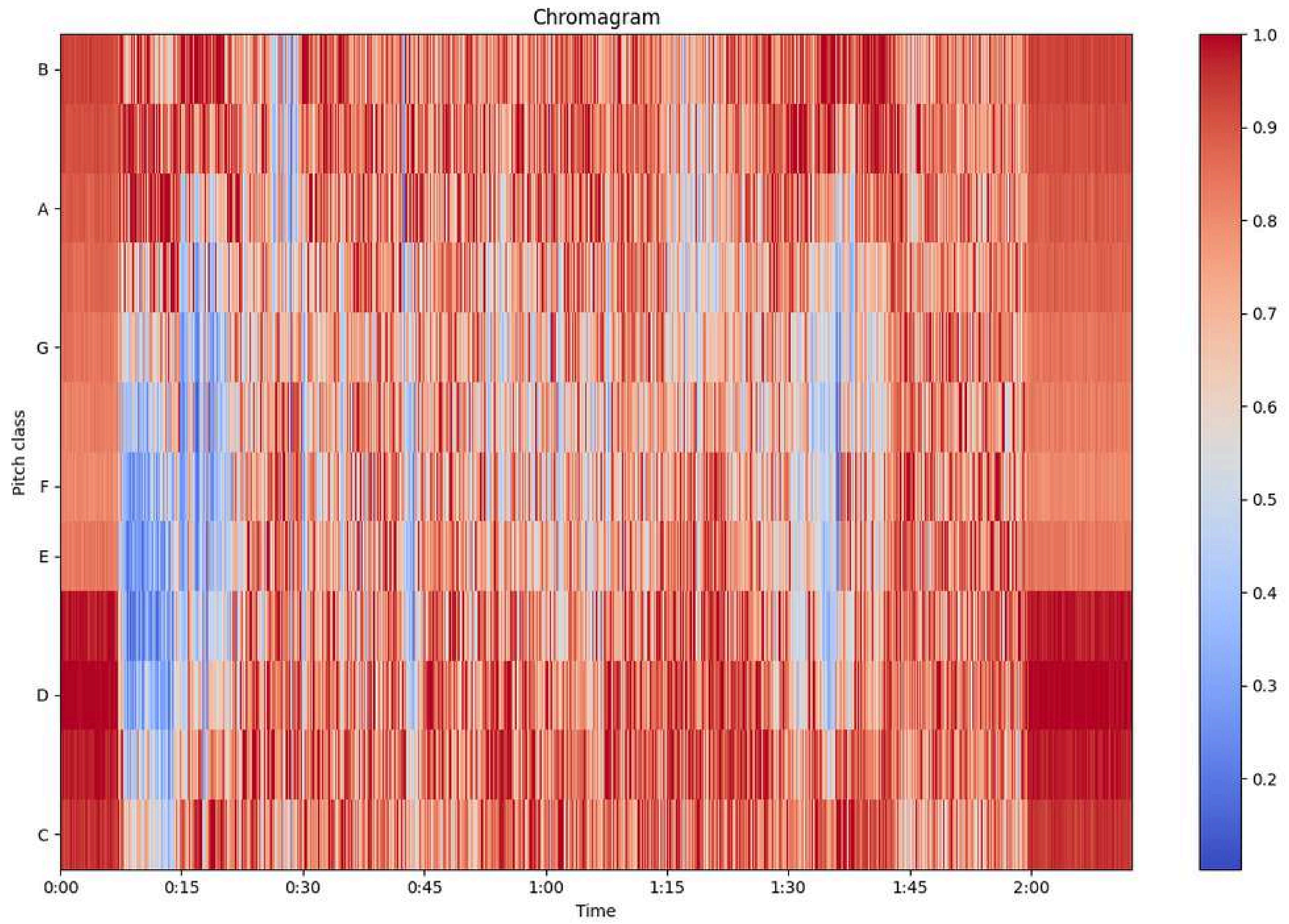


CHROMAGRAM

- Chroma features capture the **harmonic content** of a sound by mapping the audio spectrum into **12 distinct pitch classes** in this order :

  C, C#, D, D#, E, F, F#, G, G#, A, A#, and B.

Chromagram of one of the songs

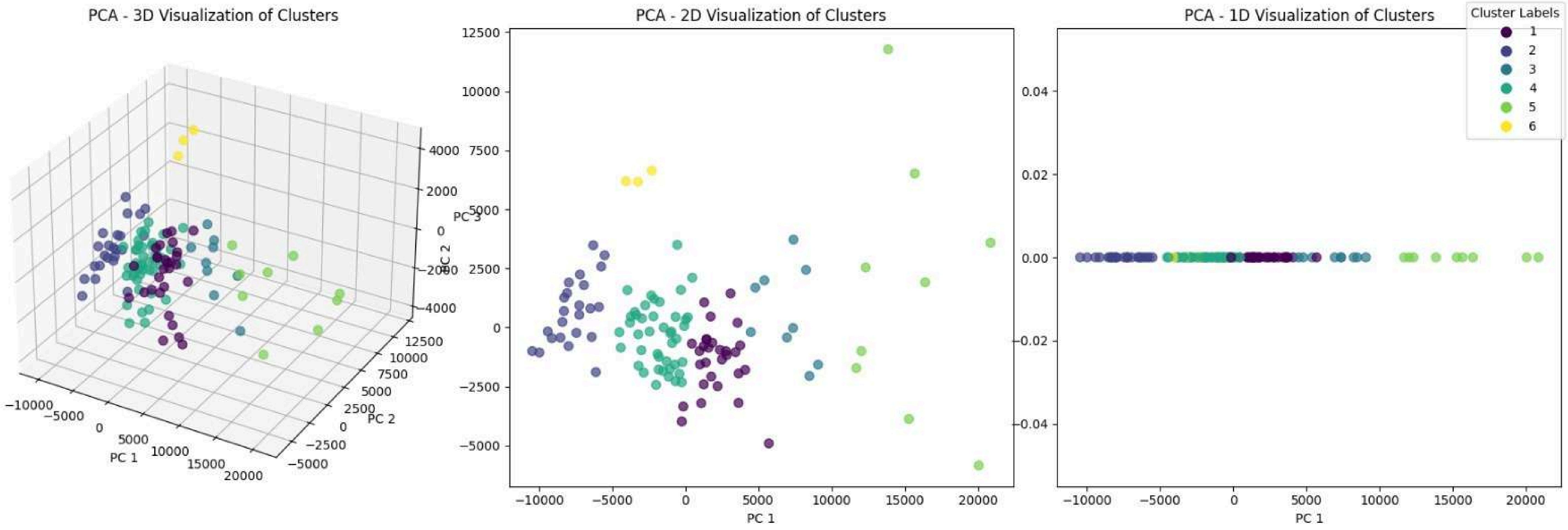


Our apprach for Unsupervised learning

- For each attempt, we selected our features, applied the **K-means clustering** algorithm, and evaluated the resulting clusters using the **Silhouette score**
- Then, we used **PCA** to reduce the data to the top 3 principal components for visualization in a **scatter plot**

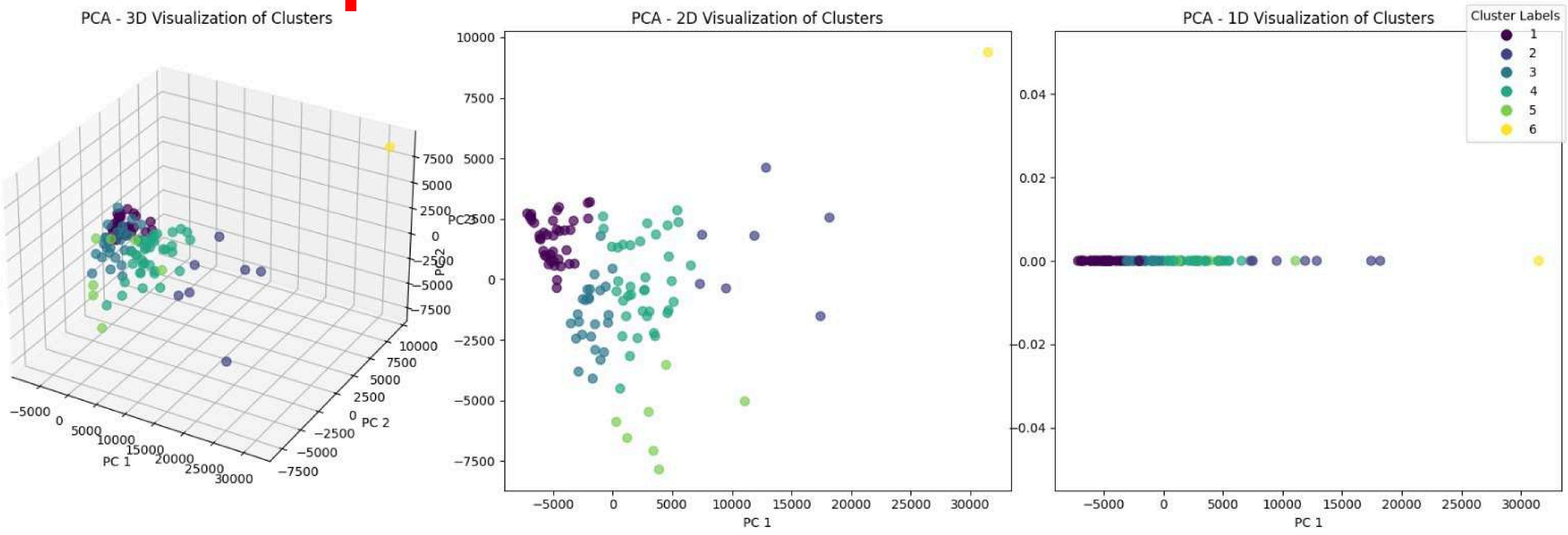
OUR FAILED ATTEMPTS USING UNSUPERVISED LEARNING ARE ON THE NEXT 4 SLIDES

Failed Attempt 1



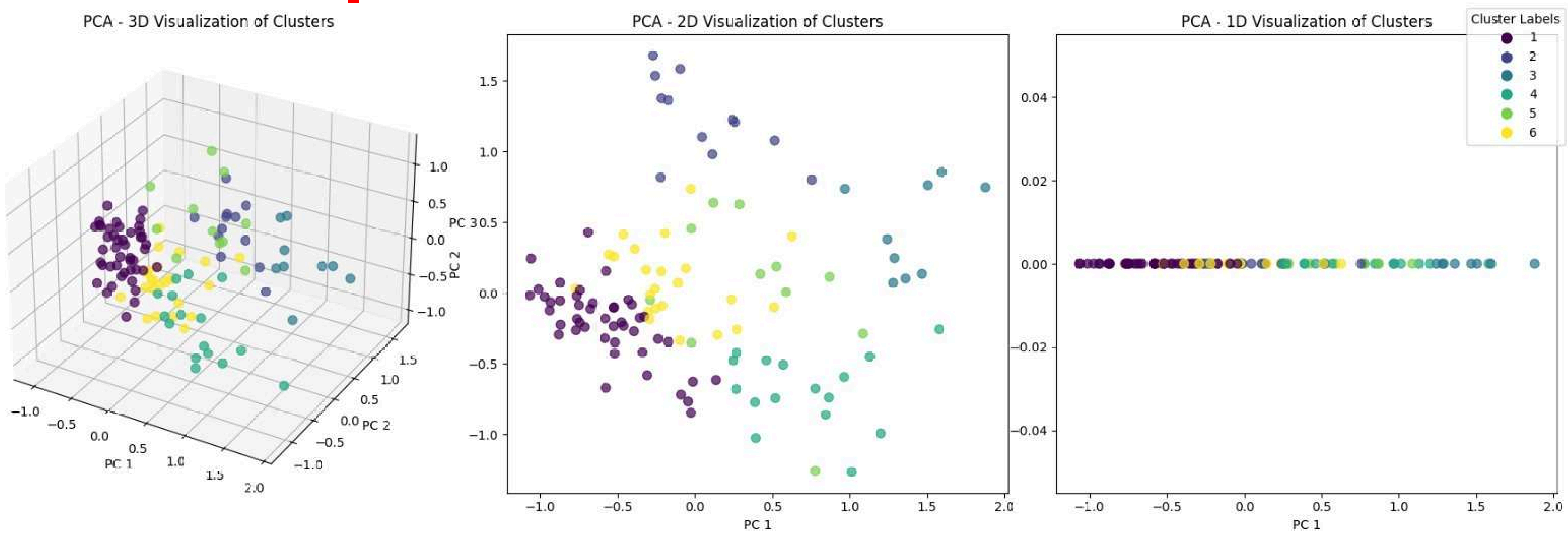
- We calculated the **mean**, **variance**, **skewness**, **kurtosis**, **delta mean**, **delta variance**, **delta square mean**, and **delta square variance** as input features for each column.
- Silhouette Score: **0.2987**
- Why it doesn't work : The extracted statistical features lack the ability to capture the underlying structure of the data, leading to poor clustering results with k-means.

Failed Attempt 2



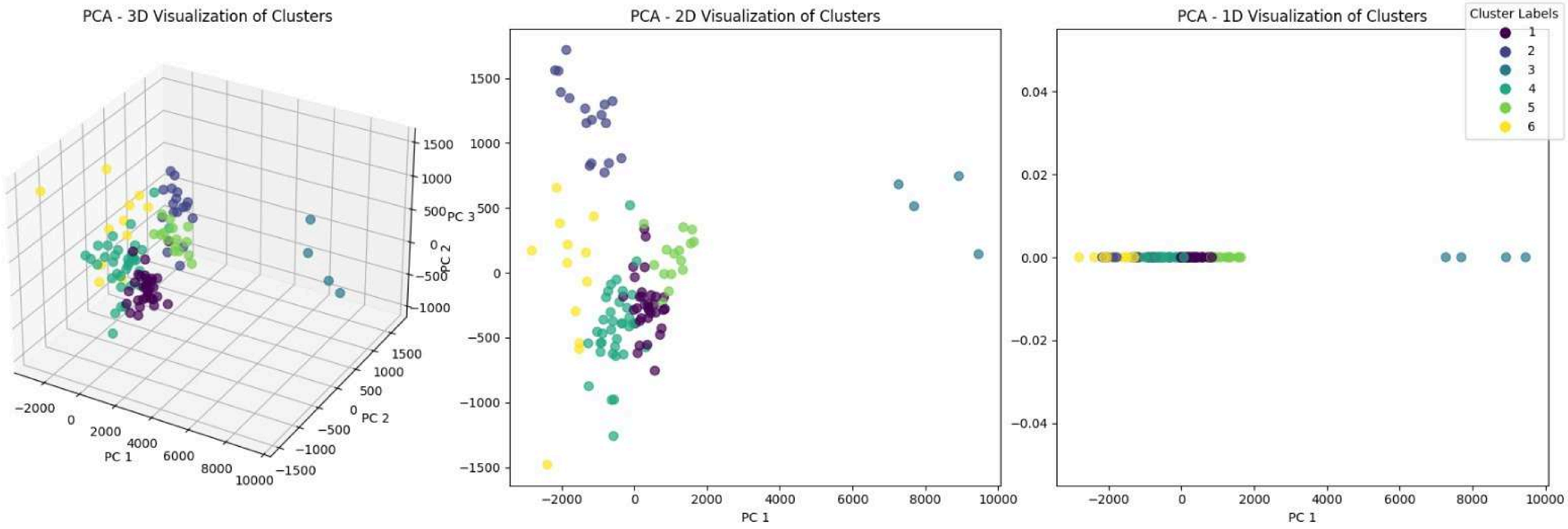
- Here we used data of only the **first 5 second** of each song, and again calculated the same 8 parameters to check if the starting beat could help classify the songs
- Silhouette Score: **0.3001**
- Why it doesn't work : Early sections of songs can often be similar or less varied, especially across genres with overlapping styles, making it harder for the model to capture unique patterns for accurate classification.

Failed Attempt 3



- We **shrunk** all the songs to **60 seconds** by removing datapoints at fixed intervals expecting clear clusters for National Anthem as the data given for the National anthem was of varied length but the National Anthem is supposed to be 52 sec.
- Silhouette Score : **0.1184**
- Why it doesn't work : Reducing all songs to 60 seconds may have removed meaningful temporal features, causing poor clustering

Failed Attempt 4



- Here we reduced the rows (time series data) which ranged from 5,000-20,000 of them to 20 by calculating the **PCA** and using the top 20 features
- Silhouette Score : **0.2008**
- Why this doens't work : Reducing the time series data to 20 PCA features likely discarded essential temporal patterns, leading to weak clustering performance

Why we chose CNNs...

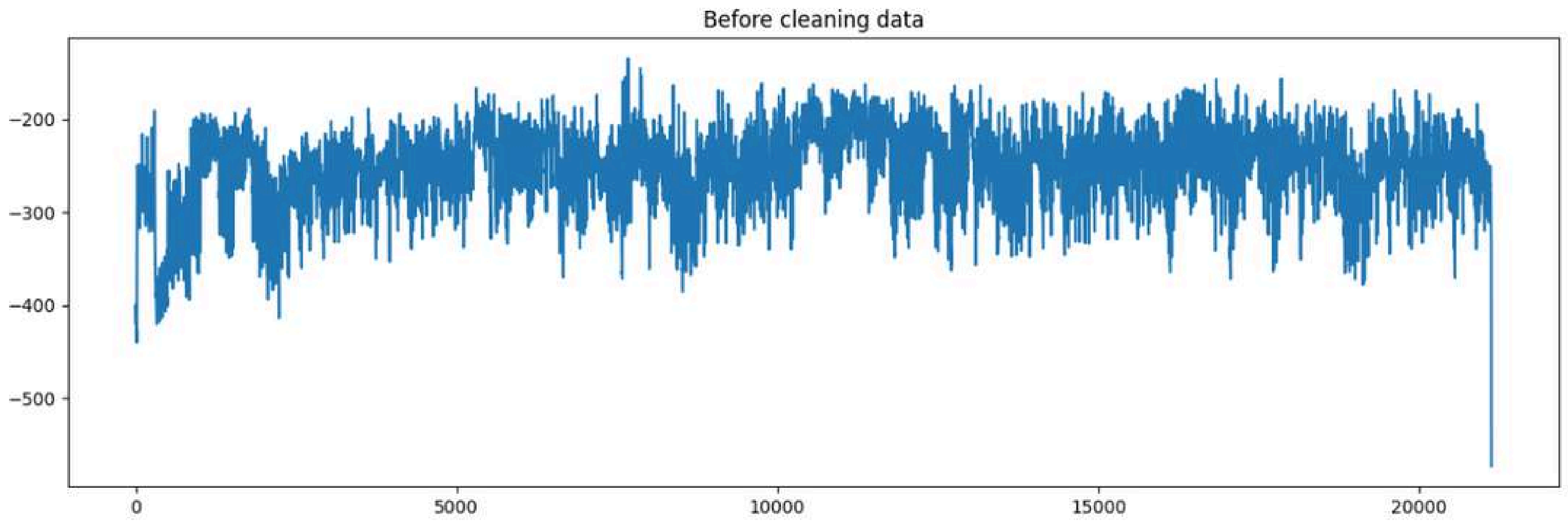
- Despite trying various unsupervised methods, the **silhouette score** stayed **below 0.3**
- As a result, we switched to using a **CNN** to automatically learn and extract meaningful features, reducing the data to **512 components**
- We then performed clustering on these learned representations

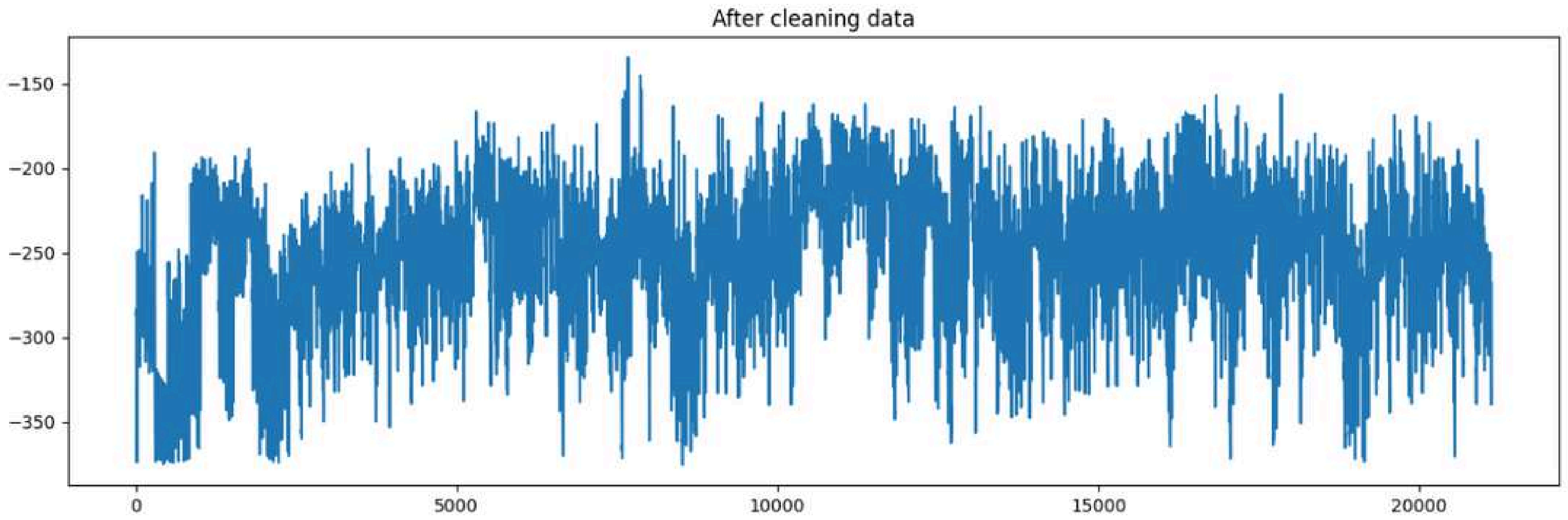
GENERATING THE DATASET

- Used an **API** to retrieve around **80** songs per artist from **Spotify** and downloaded them from YouTube; for categories like Bhavgeet, downloaded about 80 songs directly from their **YouTube** playlists
- Converted each downloaded song into **MFCC** coefficients using the given conversion code for further processing

PREPROCESSING THE DATA

- Used the Exponential Moving Average (**EMA**) method to **smoothen** the generated data, reducing **noise** and emphasising **trends**
- Removed outliers from the data to improve **accuracy** and prevent anomalies from affecting the results





CREATING THE DATASET

- Preprocessed the data and reduced each sample to a **20x5000 tensor** using **interpolation** to standardize input size
- Imported CSV files and applied an **80-20 train-test split** for model evaluation
- Implemented a **custom Dataset** and **DataLoader class** for efficient batch processing and seamless data feeding into the model

```
from torch.utils.data import Dataset

class MusicDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = MusicDataset(X_train, y_train)
test_dataset = MusicDataset(X_test, y_test)

len(train_dataset), len(test_dataset)
```

✓ 0.0s

```
(375, 94)
```

```
from torch.utils.data import DataLoader
BATCH_SIZE = 4
train_dataloader = DataLoader(dataset=train_dataset,
                              batch_size=BATCH_SIZE,
                              shuffle=True,
                              num_workers=0)  # Disable mu

test_dataloader = DataLoader(dataset=test_dataset,
                             batch_size=BATCH_SIZE,
                             shuffle=False,
                             num_workers=0)  # Disable mu


train_dataloader, test_dataloader, len(train_dataloader),

```

✓ 0.0s

CREATING THE MODEL

- We chose a **CNN model** due to the **high-dimensional** nature of each input (**5000x20**), allowing it to effectively capture spatial and temporal patterns through hierarchical feature extraction

```python
import torch
import torch.nn as nn

class SimpleCNN(nn.Module):
    def __init__(self, input_channels: int = 1, output_shape: int = num_classes):
        super(SimpleCNN, self).__init__()

        self.conv_block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_channels, out_channels=32, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),
            nn.Dropout(0.3)
        )

        self.conv_block_2 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),
            nn.Dropout(0.3)
        )

        self.conv_block_3 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),
            nn.Dropout(0.3)
        )

        self._dummy_input = torch.zeros(1, input_channels, 20, 5000)
        self._conv_out_size = self._get_conv_output(self._dummy_input)
        print("Calculated output size after conv layers:", self._conv_out_size)

        self.fc1 = nn.Linear(self._conv_out_size, 512)
        self.fc2 = nn.Linear(512, output_shape)

    def _get_conv_output(self, x):
        x = self.conv_block_1(x)
        x = self.conv_block_2(x)
        x = self.conv_block_3(x)
        return int(torch.prod(torch.tensor(x.size()[1:])))  # Flatten the dimensions

    def forward(self, x, extract_features=False):
        x = self.conv_block_1(x)
        x = self.conv_block_2(x)
        x = self.conv_block_3(x)

        x = x.view(x.size(0), -1)

        if extract_features:
            features = self.fc1(x)
            return features

        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

CNN EXTRACTING FEATURES

- The CNN extracts **relevant features** from the input data, progressively capturing more complex patterns through each **convolutional block**
- Before the **final layer**, the network reduces these features to a **512-dimensional vector**, which we use for cluster analysis and data visualization

```
from torchsummary import summary
summary(model, (1, 5000, 20))
```

✓ 1.1s

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 32, 5000, 20]             320
              ReLU-2        [-1, 32, 5000, 20]               0
         MaxPool2d-3        [-1, 32, 2500, 10]               0
           Dropout-4        [-1, 32, 2500, 10]               0
            Conv2d-5        [-1, 64, 2500, 10]          18,496
              ReLU-6        [-1, 64, 2500, 10]               0
         MaxPool2d-7         [-1, 64, 1250, 5]               0
           Dropout-8         [-1, 64, 1250, 5]               0
            Conv2d-9        [-1, 128, 1250, 5]          73,856
             ReLU-10        [-1, 128, 1250, 5]               0
        MaxPool2d-11         [-1, 128, 625, 2]               0
          Dropout-12         [-1, 128, 625, 2]               0
           Linear-13                  [-1, 512]      81,920,512
           Linear-14                    [-1, 6]           3,078
================================================================
Total params: 82,016,262
Trainable params: 82,016,262
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.38
Forward/backward pass size (MB): 106.21
Params size (MB): 312.87
Estimated Total Size (MB): 419.45
----------------------------------------------------------------
```

TRAINING THE MODEL

- Ran the model for **19 epochs**, monitoring the loss and accuracy to ensure convergence
- Used an **Adam optimizer** and a **Cross entropy loss rate** scheduler to fine-tune the training process.
- Logged metrics such as loss, accuracy, and validation performance for each epoch

```
from train_utils import train_step, test_step, train

model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[133] ✓ 10m 0.8s

```
Epoch: 0 | Train loss: 4.1993 | Train acc: 0.2323 | Test loss: 1.786311 | Test acc: 0.2083
Epoch: 1 | Train loss: 1.7437 | Train acc: 0.2553 | Test loss: 1.679188 | Test acc: 0.2604
Epoch: 2 | Train loss: 1.5822 | Train acc: 0.3209 | Test loss: 1.621248 | Test acc: 0.2604
Epoch: 3 | Train loss: 1.4392 | Train acc: 0.4211 | Test loss: 1.612250 | Test acc: 0.4167
Epoch: 4 | Train loss: 1.3762 | Train acc: 0.4344 | Test loss: 1.509551 | Test acc: 0.3646
```

```
model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[134] ✓ 6m 3.6s

```
Epoch: 0 | Train loss: 1.2407 | Train acc: 0.5443 | Test loss: 1.558015 | Test acc: 0.3854
Epoch: 1 | Train loss: 1.1991 | Train acc: 0.5372 | Test loss: 1.381171 | Test acc: 0.4375
Epoch: 2 | Train loss: 1.1499 | Train acc: 0.5408 | Test loss: 1.393902 | Test acc: 0.4479
```

```
model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[ ]

```
model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[135] ✓ 5m 46.2s

```
Epoch: 0 | Train loss: 1.0475 | Train acc: 0.5975 | Test loss: 1.314240 | Test acc: 0.4792
Epoch: 1 | Train loss: 0.9240 | Train acc: 0.6232 | Test loss: 1.336660 | Test acc: 0.5417
Epoch: 2 | Train loss: 0.8735 | Train acc: 0.6879 | Test loss: 1.430228 | Test acc: 0.4896
```

```
model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[136] ✓ 5m 36.1s

```
Epoch: 0 | Train loss: 0.7976 | Train acc: 0.6746 | Test loss: 1.746764 | Test acc: 0.2917
Epoch: 1 | Train loss: 0.8378 | Train acc: 0.6746 | Test loss: 1.417313 | Test acc: 0.5208
Epoch: 2 | Train loss: 0.7622 | Train acc: 0.6950 | Test loss: 1.354573 | Test acc: 0.5521
```

```
model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[137] ✓ 6m 21.0s

```
Epoch: 0 | Train loss: 0.6630 | Train acc: 0.7305 | Test loss: 1.433082 | Test acc: 0.5208
Epoch: 1 | Train loss: 0.7246 | Train acc: 0.7668 | Test loss: 1.741617 | Test acc: 0.3125
Epoch: 2 | Train loss: 0.5831 | Train acc: 0.7686 | Test loss: 1.822694 | Test acc: 0.4479
```

```
model_results = train(model=model, train_dataloader=train_dataloader, test_dataloader=test_da
```
[138] ✓ 6m 7.8s

```
Epoch: 0 | Train loss: 0.6313 | Train acc: 0.7660 | Test loss: 1.871951 | Test acc: 0.4583
Epoch: 1 | Train loss: 0.5420 | Train acc: 0.7863 | Test loss: 1.832060 | Test acc: 0.4062
Epoch: 2 | Train loss: 0.6498 | Train acc: 0.7402 | Test loss: 1.629282 | Test acc: 0.5208
```

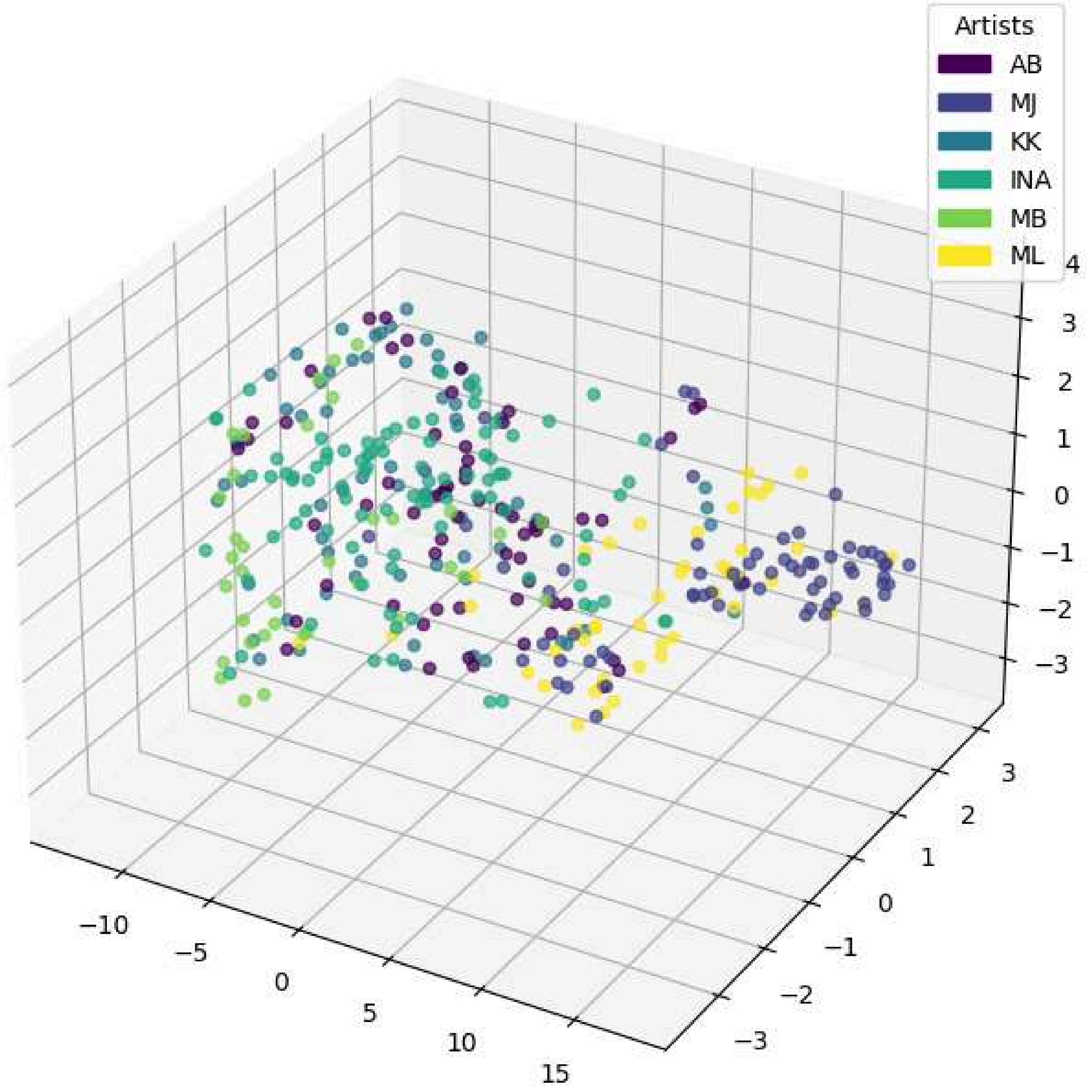
ACCURACY ON THE DATASETS

- This **does not reflect** the **accuracy** we would get on the **dataset** given but gives a good general idea because the audio files are very **similar**
- After running **19 epochs**, the following metrics were obtained

Metric	Epoch 0	Epoch 19
Train Loss	4.1993	0.6498
Train Accuracy	0.2323	0.7402
Test Loss	1.7863	1.6293
Test Accuracy	0.2083	0.5208

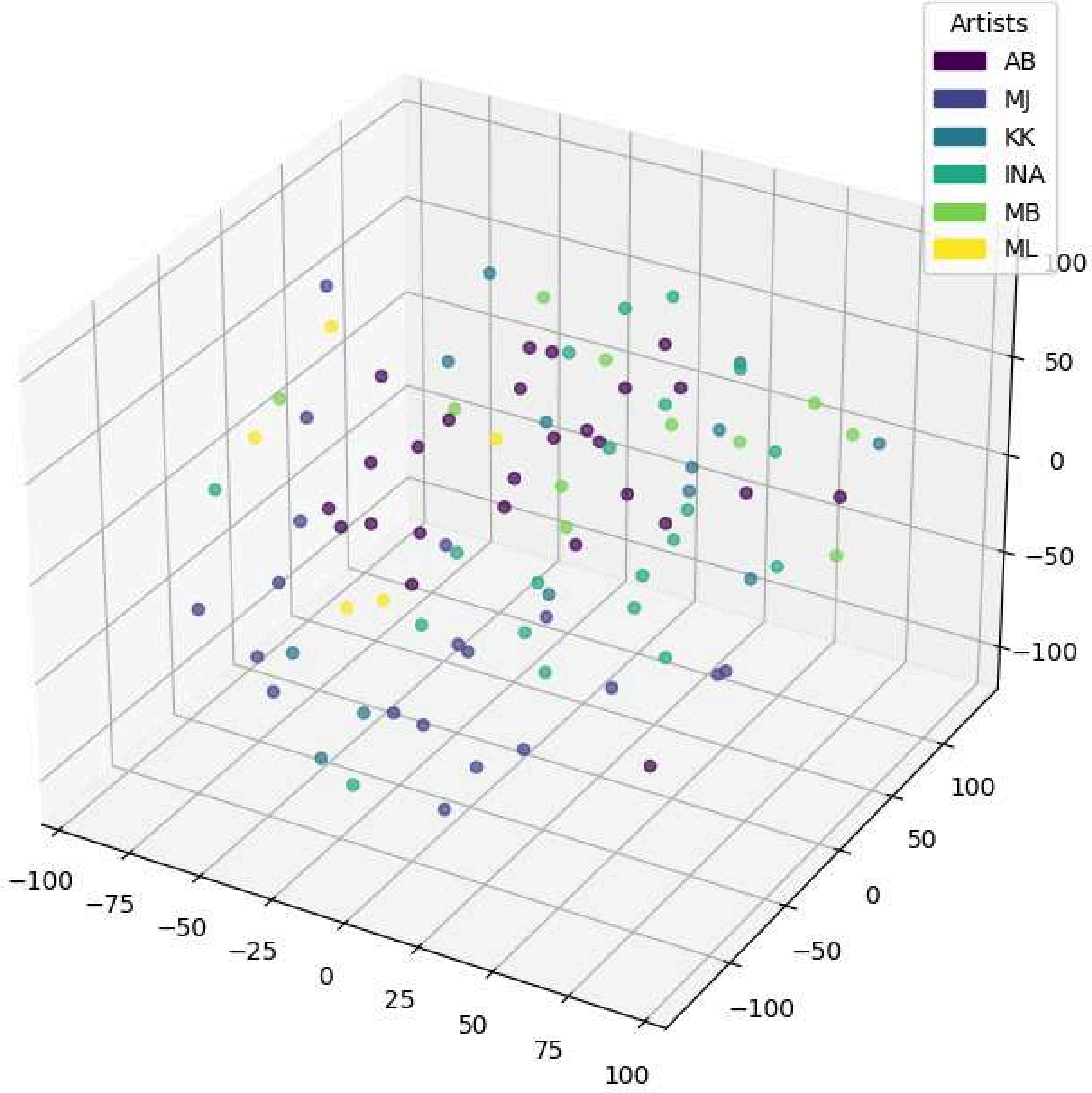
t-SNE OF THE TRAIN DATA

- Used **t-SNE** on training data to visualize high-dimensional patterns in 3D, limited to the **top 3 principal components**
- Clusters appear **unclear**; using more components could **improve separation** and reveal **feature distinctions**



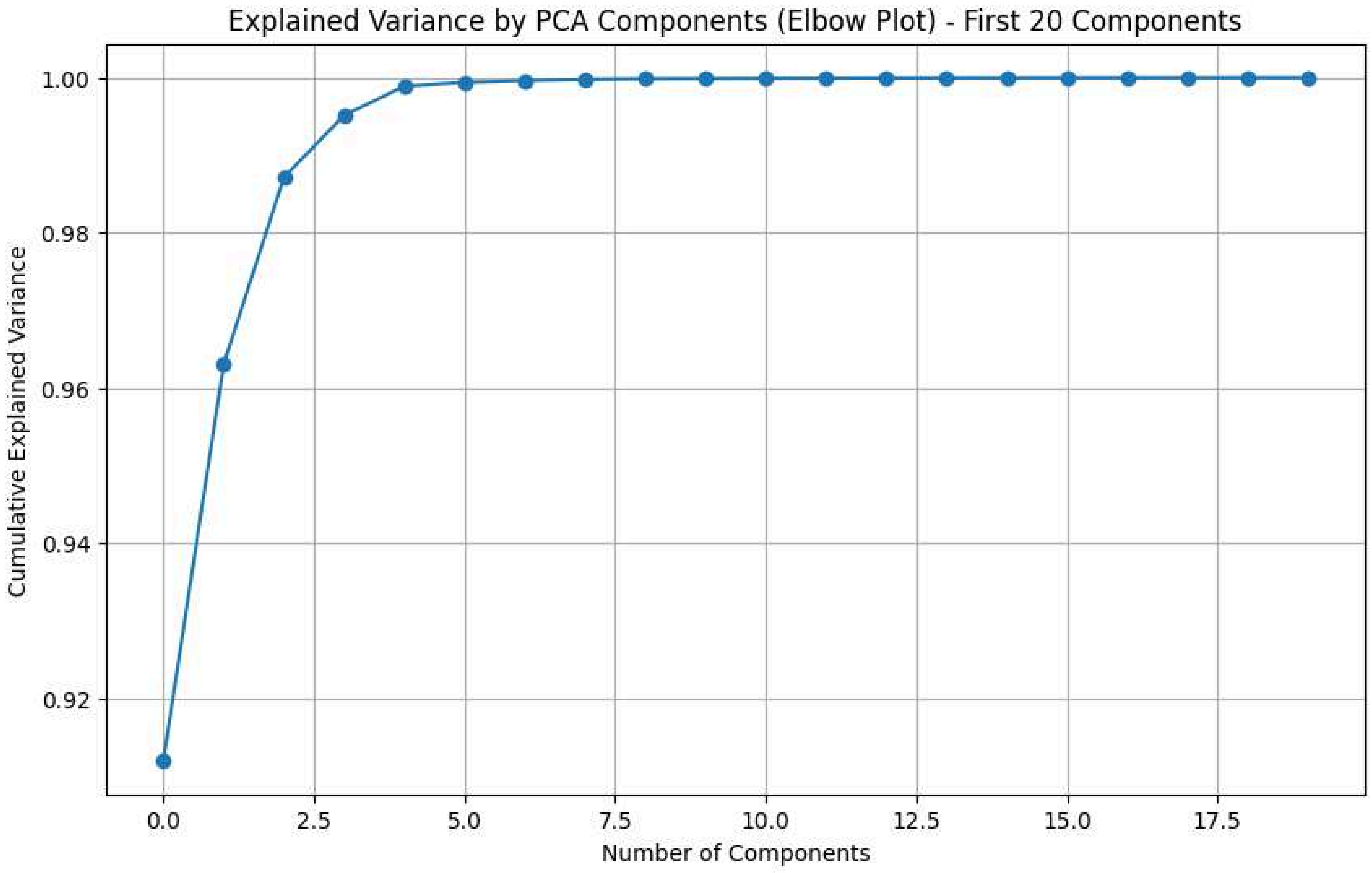
t-SNE OF THE TEST DATA

- Similarly, **t-SNE** was applied to the testing data to **visualize** it in 3D using its top **3 principal components**
- The clusters appear **indistinct**, indicating that incorporating more components could provide separation



ELBOW PLOT

- We used an **elbow plot** to find the **optimal** number of principal components for t-SNE, which **stabilizes** at the 4th or 5th component
- This suggests that using **4 or 5 components** would improve **cluster clarity** by better capturing **data variance**



EVALUATING GIVEN DATA

- Imported the provided **MFCC files**, processed the data, and **reshaped** it to match the model's expected input dimensions
- Passed the processed data through the model to obtain **category probabilities**
- Logged the probabilities for each category and identified the **predicted category** based on the **highest probability** value from the **sigmoid output**

```python
X = []
for i in tqdm(range(1, 117)):
    song = pd.read_csv(f"MFCC_T/{i:02d}-MFCC.csv")
    X.append(transform(song))
```

✓ 1m 31.0s

```
100%|██████████| 116/116 [01:31<00:00,  1.27it/s]
```

```python
X1 = torch.stack(X)
X1 = X1.squeeze(1)
X1.shape
```

✓ 0.0s

```
torch.Size([116, 1, 5000, 20])
```

```python
# Print the header with artist names and aligned columns
header = f"{'Song Number':<12} | " + " | ".join([f"Prob {artist:<4}" for artist in artist_short]) + " | Predicted Artist"
print(header)
print("-" * len(header))  # Separator line

# Print each song's probabilities and predicted artist
for i in range(1, 117):
    # Format probabilities with 2 decimal places and percent sign
    probabilities = " | ".join([f"{pred_probs[i-1][j] * 100:8.2f}%" for j in range(6)])
    predicted_artist = "    "+artist_short[pred_labels[i-1]]

    # Print the row with aligned columns
    print(f"{i:<12} | {probabilities} | {predicted_artist}")
```

✓ 0.0s

Song Number	Prob AB	Prob MJ	Prob KK	Prob INA	Prob MB	Prob ML	Predicted Artist
1	20.25%	3.10%	28.27%	40.32%	3.93%	4.13%	INA
2	35.64%	5.78%	9.15%	37.00%	8.75%	3.68%	INA
3	0.21%	99.61%	0.02%	0.08%	0.00%	0.07%	MJ
4	92.03%	0.00%	0.01%	0.01%	7.82%	0.13%	AB
5	11.01%	0.01%	83.96%	0.01%	2.34%	2.67%	KK
6	0.10%	0.01%	0.00%	0.04%	1.34%	98.51%	ML
7	58.04%	0.02%	9.12%	0.03%	14.21%	18.58%	AB
8	0.60%	98.06%	0.05%	1.12%	0.02%	0.15%	MJ
9	36.73%	0.00%	47.81%	0.00%	10.66%	4.80%	KK
10	93.96%	0.99%	0.56%	0.35%	1.17%	2.98%	AB
11	0.10%	0.00%	0.05%	0.00%	99.76%	0.08%	MB
12	93.23%	0.01%	0.42%	0.09%	5.12%	1.13%	AB
13	0.10%	0.00%	0.00%	0.00%	99.67%	0.23%	MB
14	9.04%	0.07%	0.99%	0.06%	23.52%	66.32%	ML
15	96.92%	0.01%	0.08%	0.01%	2.19%	0.80%	AB
16	14.12%	0.44%	44.84%	5.12%	28.67%	6.82%	KK

Results

Song Number	Prob AB	Prob MJ	Prob KK	Prob INA	Prob MB	Prob ML	Predicted Artist
1	20.25%	3.10%	28.27%	40.32%	3.93%	4.13%	INA
2	35.64%	5.78%	9.15%	37.00%	8.75%	3.68%	INA
3	0.21%	99.61%	0.02%	0.08%	0.00%	0.07%	MJ
4	92.03%	0.00%	0.01%	0.01%	7.82%	0.13%	AB
5	11.01%	0.01%	83.96%	0.01%	2.34%	2.67%	KK
6	0.10%	0.01%	0.00%	0.04%	1.34%	98.51%	ML
7	58.04%	0.02%	9.12%	0.03%	14.21%	18.58%	AB
8	0.60%	98.06%	0.05%	1.12%	0.02%	0.15%	MJ
9	36.73%	0.00%	47.81%	0.00%	10.66%	4.80%	KK
10	93.96%	0.99%	0.56%	0.35%	1.17%	2.98%	AB
11	0.10%	0.00%	0.05%	0.00%	99.76%	0.08%	MB
12	93.23%	0.01%	0.42%	0.09%	5.12%	1.13%	AB
13	0.10%	0.00%	0.00%	0.00%	99.67%	0.23%	MB
14	9.04%	0.07%	0.99%	0.06%	23.52%	66.32%	ML
15	96.92%	0.01%	0.08%	0.01%	2.19%	0.80%	AB
16	14.12%	0.44%	44.84%	5.12%	28.67%	6.82%	KK
17	92.27%	0.45%	4.08%	0.84%	1.52%	0.84%	AB
18	77.48%	0.00%	13.89%	0.02%	7.61%	1.00%	AB
19	0.75%	0.00%	0.04%	0.00%	3.05%	96.16%	ML
20	6.53%	77.78%	2.64%	8.86%	1.53%	2.66%	MJ
21	13.84%	0.00%	13.92%	0.00%	35.20%	37.05%	ML
22	91.69%	0.38%	6.09%	0.31%	0.49%	1.04%	AB
23	78.50%	0.00%	4.04%	0.04%	17.16%	0.26%	AB
24	19.65%	11.38%	11.84%	4.72%	14.58%	37.84%	ML
25	42.77%	0.00%	4.45%	0.03%	43.73%	9.02%	MB
26	6.00%	0.14%	79.03%	4.20%	7.21%	3.41%	KK
27	8.35%	0.58%	7.64%	71.12%	8.25%	4.05%	INA
28	11.56%	0.00%	83.00%	0.00%	2.97%	2.47%	KK
29	3.69%	0.00%	94.47%	0.01%	1.46%	0.37%	KK
30	25.92%	0.10%	26.92%	0.58%	20.10%	26.37%	KK
31	0.02%	0.03%	0.01%	0.06%	0.05%	99.82%	ML
32	42.82%	0.04%	9.31%	0.17%	37.52%	10.13%	AB
33	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	MB
34	14.65%	7.31%	7.53%	26.16%	5.51%	38.84%	ML
35	1.00%	1.01%	0.13%	96.49%	1.13%	0.24%	INA
36	51.49%	0.04%	5.21%	0.12%	26.48%	16.65%	AB
37	35.58%	0.01%	45.65%	0.01%	14.48%	4.26%	KK
38	13.87%	0.00%	0.57%	8.27%	76.39%	0.89%	MB
39	98.31%	0.00%	0.00%	0.00%	1.47%	0.22%	AB
40	11.69%	0.00%	0.22%	0.01%	86.78%	1.30%	MB
41	83.25%	0.03%	0.57%	0.11%	15.45%	0.58%	AB
42	14.86%	0.09%	0.09%	1.68%	48.85%	34.43%	MB
43	73.78%	0.00%	0.28%	0.03%	23.55%	2.36%	AB
44	0.12%	99.65%	0.04%	0.18%	0.00%	0.01%	MJ
45	0.47%	98.37%	0.11%	0.49%	0.01%	0.56%	MJ
46	3.91%	0.00%	90.54%	0.00%	5.25%	0.30%	KK
47	17.91%	3.45%	0.04%	4.77%	3.73%	70.10%	ML
48	30.29%	0.04%	28.60%	0.01%	8.15%	32.91%	ML
49	39.08%	0.00%	0.07%	0.03%	59.35%	1.47%	MB
50	0.83%	0.00%	98.58%	0.00%	0.44%	0.15%	KK
51	17.90%	0.00%	80.20%	0.00%	1.61%	0.29%	KK
52	0.14%	0.01%	0.01%	0.06%	0.42%	99.35%	ML
53	0.86%	98.21%	0.08%	0.66%	0.03%	0.16%	MJ
54	85.46%	0.01%	5.52%	0.01%	5.68%	3.33%	AB
55	35.51%	12.94%	9.49%	14.01%	11.46%	16.60%	AB
56	0.01%	0.00%	0.00%	0.00%	99.91%	0.08%	MB
57	0.83%	0.00%	0.55%	0.00%	98.55%	0.07%	MB
58	1.77%	0.01%	0.81%	0.02%	96.55%	0.85%	MB
59	17.96%	0.00%	77.23%	0.00%	2.88%	1.92%	KK
60	95.20%	0.01%	0.15%	0.01%	3.83%	0.81%	AB
61	1.03%	0.87%	7.71%	89.93%	0.34%	0.11%	INA
62	77.32%	0.00%	0.22%	0.06%	22.16%	0.24%	AB
63	25.17%	0.03%	64.42%	0.40%	8.37%	1.61%	KK
64	6.79%	0.00%	0.00%	0.02%	92.16%	1.03%	MB
65	41.77%	0.00%	10.71%	0.00%	1.33%	46.19%	ML
66	12.30%	0.02%	80.70%	0.09%	4.90%	1.99%	KK
67	2.08%	0.06%	95.42%	0.30%	1.15%	0.99%	KK
68	50.61%	1.48%	16.84%	2.28%	25.38%	3.42%	AB
69	1.49%	93.76%	0.28%	3.66%	0.10%	0.70%	MJ
70	0.20%	0.00%	0.00%	0.00%	2.82%	96.98%	ML
71	3.52%	0.01%	95.24%	0.01%	0.93%	0.29%	KK
72	60.17%	0.00%	0.00%	0.00%	39.29%	0.54%	AB
73	3.87%	0.07%	0.00%	74.48%	16.75%	4.82%	INA
74	37.30%	8.53%	2.03%	16.72%	22.94%	12.46%	AB
75	8.65%	1.17%	67.73%	5.87%	14.39%	2.19%	KK
76	1.89%	0.00%	0.31%	0.01%	97.53%	0.25%	MB
77	78.67%	0.00%	0.06%	0.00%	20.66%	0.61%	AB
78	4.84%	86.19%	1.86%	3.54%	0.83%	2.75%	MJ
79	25.25%	0.00%	0.00%	0.00%	73.85%	0.89%	MB
80	78.20%	0.00%	6.67%	0.00%	14.76%	0.37%	AB
81	97.50%	0.00%	0.81%	0.03%	1.64%	0.03%	AB
82	73.44%	0.00%	21.61%	0.00%	3.41%	1.53%	AB
83	0.38%	0.00%	99.43%	0.00%	0.12%	0.07%	KK
84	18.89%	0.00%	66.41%	0.00%	9.80%	4.90%	KK
85	0.01%	0.00%	0.00%	0.00%	99.98%	0.02%	MB
86	0.16%	99.70%	0.02%	0.10%	0.00%	0.01%	MJ
87	2.40%	28.48%	0.51%	66.06%	1.95%	0.60%	INA
88	47.93%	0.55%	2.37%	0.96%	22.47%	25.72%	AB
89	5.68%	8.60%	2.47%	72.13%	7.02%	4.10%	INA
90	11.33%	7.34%	8.49%	68.10%	3.44%	1.30%	INA
91	0.01%	0.00%	0.00%	0.00%	99.99%	0.00%	MB
92	21.92%	0.01%	7.49%	0.03%	50.24%	20.31%	MB
93	0.54%	0.00%	97.66%	0.00%	1.06%	0.75%	KK
94	15.07%	0.00%	0.94%	0.06%	48.51%	35.42%	MB
95	0.71%	0.03%	0.01%	99.04%	0.19%	0.02%	INA
96	3.85%	0.01%	94.18%	0.08%	1.61%	0.28%	KK
97	10.68%	0.04%	84.65%	0.13%	2.91%	1.59%	KK
98	1.10%	96.18%	0.16%	2.38%	0.03%	0.14%	MJ
99	11.37%	0.00%	0.01%	0.00%	36.01%	52.61%	ML
100	24.01%	0.00%	71.89%	0.00%	3.40%	0.70%	KK
101	28.00%	0.02%	6.93%	0.02%	10.31%	54.73%	ML
102	7.81%	0.00%	5.29%	0.00%	86.68%	0.22%	MB
103	4.50%	58.23%	3.49%	7.78%	1.47%	24.54%	MJ
104	22.32%	23.48%	2.32%	26.16%	9.15%	16.56%	INA
105	86.07%	0.00%	0.04%	0.02%	9.87%	4.00%	AB
106	41.42%	0.56%	0.19%	4.22%	30.84%	22.76%	AB
107	9.32%	10.98%	4.63%	61.20%	6.68%	7.20%	INA
108	1.40%	0.74%	2.31%	79.81%	14.24%	1.51%	INA
109	0.50%	0.00%	0.04%	0.00%	0.73%	98.72%	ML
110	55.56%	0.00%	0.13%	0.00%	4.42%	39.89%	AB
111	3.75%	0.00%	91.25%	0.00%	4.75%	0.25%	KK
112	10.55%	0.00%	0.21%	0.00%	86.46%	2.77%	MB
113	15.14%	0.31%	0.65%	0.55%	78.44%	4.91%	MB
114	2.32%	89.25%	1.08%	6.65%	0.15%	0.56%	MJ
115	0.33%	0.00%	0.01%	0.02%	98.75%	0.90%	MB
116	17.86%	0.08%	8.85%	3.59%	18.57%	51.04%	ML

RESULT ANALYSIS

- We observe high certainty for certain songs, likely because they **coincidentally** appear in the **training dataset** and represent popular hits by each artist
- For example, many **Michael Jackson** songs are **confidently classified** as his due to their distinct style
- The National Anthem shows **low certainty** since it's unlikely that the exact version in our dataset appears in the training data

Prob AB	Prob MJ	Prob KK	Prob INA	Prob MB	Prob ML	Predicted
20.25%	3.10%	28.27%	40.32%	3.93%	4.13%	INA
35.64%	5.78%	9.15%	37.00%	8.75%	3.68%	INA
0.21%	99.61%	0.02%	0.08%	0.00%	0.07%	MJ
92.03%	0.00%	0.01%	0.01%	7.82%	0.13%	AB
11.01%	0.01%	83.96%	0.01%	2.34%	2.67%	KK
0.10%	0.01%	0.00%	0.04%	1.34%	98.51%	ML
58.04%	0.02%	9.12%	0.03%	14.21%	18.58%	AB
0.60%	98.06%	0.05%	1.12%	0.02%	0.15%	MJ
36.73%	0.00%	47.81%	0.00%	10.66%	4.80%	KK
93.96%	0.99%	0.56%	0.35%	1.17%	2.98%	AB
0.10%	0.00%	0.05%	0.00%	99.76%	0.08%	MB
93.23%	0.01%	0.42%	0.09%	5.12%	1.13%	AB
0.10%	0.00%	0.00%	0.00%	99.67%	0.23%	MB
9.04%	0.07%	0.99%	0.06%	23.52%	66.32%	ML
96.92%	0.01%	0.08%	0.01%	2.19%	0.80%	AB
14.12%	0.44%	44.84%	5.12%	28.67%	6.82%	KK
92.27%	0.45%	4.08%	0.84%	1.52%	0.84%	AB
77.48%	0.00%	13.89%	0.02%	7.61%	1.00%	AB
0.75%	0.00%	0.04%	0.00%	3.05%	96.16%	ML
6.53%	77.78%	2.64%	8.86%	1.53%	2.66%	MJ
13.84%	0.00%	13.92%	0.00%	35.20%	37.05%	ML
91.69%	0.38%	6.09%	0.31%	0.49%	1.04%	AB
78.50%	0.00%	4.04%	0.04%	17.16%	0.26%	AB
19.65%	11.38%	11.84%	4.72%	14.58%	37.84%	ML
42.77%	0.00%	4.45%	0.03%	43.73%	9.02%	MB
6.00%	0.14%	79.03%	4.20%	7.21%	3.41%	KK
8.35%	0.58%	7.64%	71.12%	8.25%	4.05%	INA
11.56%	0.00%	83.00%	0.00%	2.97%	2.47%	KK
3.69%	0.00%	94.47%	0.01%	1.46%	0.37%	KK
25.92%	0.10%	26.92%	0.58%	20.10%	26.37%	KK
0.02%	0.03%	0.01%	0.06%	0.05%	99.82%	ML
42.82%	0.04%	9.31%	0.17%	37.52%	10.13%	AB
0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	MB
14.65%	7.31%	7.53%	26.16%	5.51%	38.84%	ML
1.00%	1.01%	0.13%	96.49%	1.13%	0.24%	INA
51.49%	0.04%	5.21%	0.12%	26.48%	16.65%	AB
35.58%	0.01%	45.65%	0.01%	14.48%	4.26%	KK
13.87%	0.00%	0.57%	8.27%	76.39%	0.89%	MB
98.31%	0.00%	0.00%	0.00%	1.47%	0.22%	AB
11.69%	0.00%	0.22%	0.01%	86.78%	1.30%	MB
83.25%	0.03%	0.57%	0.11%	15.45%	0.58%	AB

SONG DISTRIBUTION ACROSS GROUPS

```python
songs_by_artist = {artist: [] for artist in artist_short}

for i in range(1, 117):
    predicted_artist = artist_short[pred_labels[i - 1]]
    songs_by_artist[predicted_artist].append(i)

for artist, songs in songs_by_artist.items():
    print(f"{artist}: {songs}")

```

[34] ✓ 0.0s

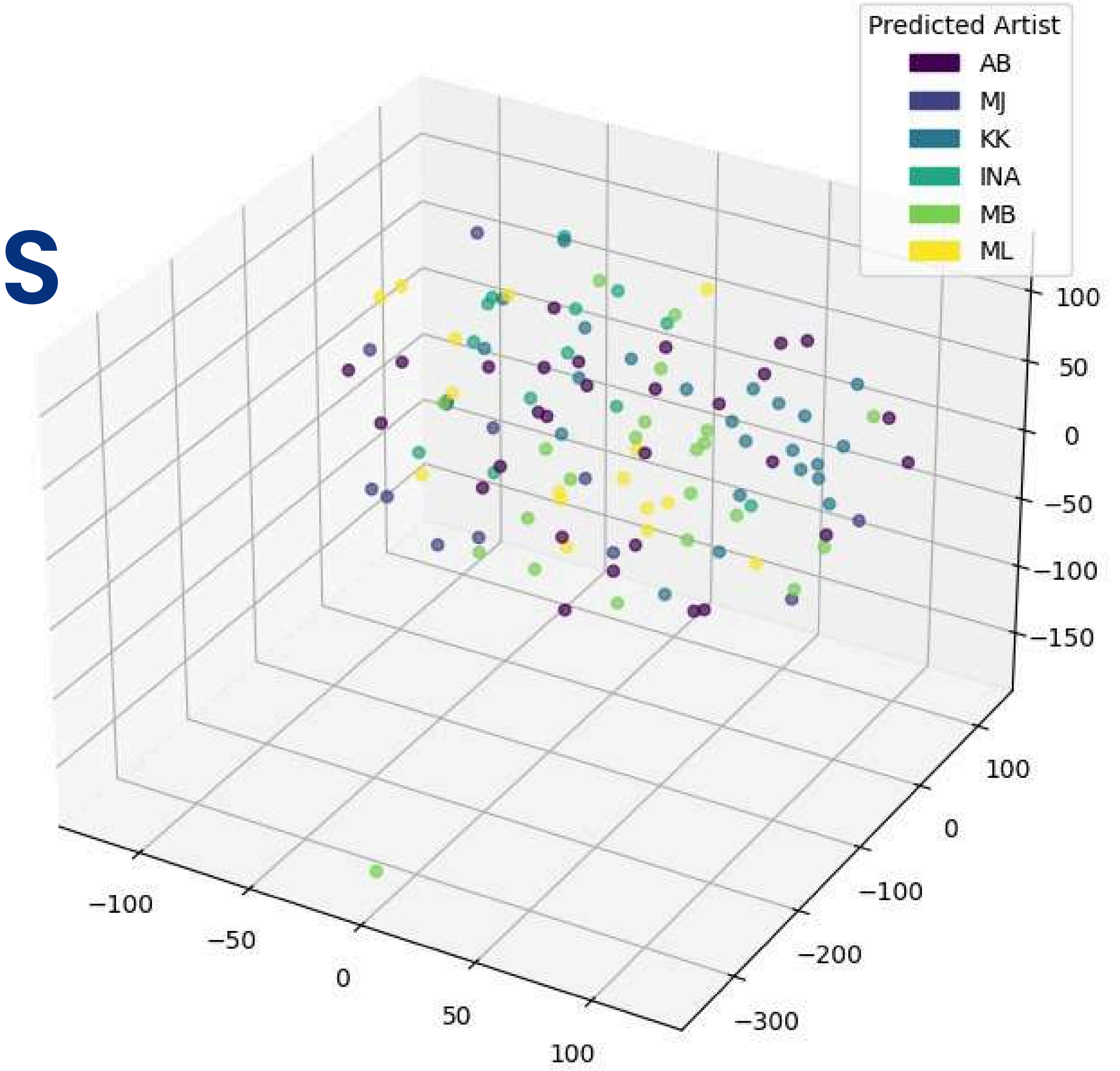
```
AB: [4, 7, 10, 12, 15, 17, 18, 22, 23, 32, 36, 39, 41, 43, 54, 55, 60, 62, 68, 72, 74, 77, 80, 81, 82, 88, 105, 106, 110]
MJ: [3, 8, 20, 44, 45, 53, 69, 78, 86, 98, 103, 114]
KK: [5, 9, 16, 26, 28, 29, 30, 37, 46, 50, 51, 59, 63, 66, 67, 71, 75, 83, 84, 93, 96, 97, 100, 111]
INA: [1, 2, 27, 35, 61, 73, 87, 89, 90, 95, 104, 107, 108]
MB: [11, 13, 25, 33, 38, 40, 42, 49, 56, 57, 58, 64, 76, 79, 85, 91, 92, 94, 102, 112, 113, 115]
ML: [6, 14, 19, 21, 24, 31, 34, 47, 48, 52, 65, 70, 99, 101, 109, 116]
```

AB - Asha Bhosle
MJ - Michael Jackson
KK - Kishore Kumar

INA - Indian National Anthem
MB - Marathi Bhavgeet
ML - Marathi Lavani

t-SNE OF THE SONG PREDICTIONS

- The clusters appear **indistinct** due to the use of only 3 principal components, which **restricts** the data's **representational capacity**
- Increasing the dimensionality to 4 or 5 components could likely **enhance cluster separation**, making the groupings more distinct and well-defined



3 AUDIO FILE NUMBERS CORRESPONDING TO EACH SINGER

National Anthem	Asha Bhosale	Kishore Kumar	Michael Jackson
27	15	59	3
35	39	83	8
87	77	84	20

RESULTS

How many problems have been correctly solved?

Solved Problems 1, 2, 3

- **Problem 1** : Classified the 116 songs into the 6 groups as shown in slide 36
- **Problem 2** : National Anthem file numbers : 27, 35, 87
- **Problem 3** : Asha Bhosle file numbers : 15, 39, 77
  Kishore Kumar file numbers : 59, 83, 84
  Michael Jackson file numbers : 3, 8, 20

This is show in slide 34

RESULTS

Has there been any creative thinking and innovation while solving the problems?

- **Unique Dataset Creation** : Collected audio data by downloading songs from YouTube, creating a diverse dataset with different genres and sound qualities
- **CNN for Audio Classification** : Applied CNNs, which are usually used for images, to classify audio data, showing a new way to use deep learning for this task
- **3D Classification Plots** : Used 3D plots to visualize data where 2D plots wouldn’t provide enough information

RESULTS

Quality of Feature Engineering / Feature Creation in terms of relevance to the problem

- **Automatic Feature Learning** : The convolutional layers automatically learn relevant features from raw data, reducing the need for manual feature extraction
- **Relevance of Extracted Features** : The model captures both low-level and high-level features, ensuring they are suited for the classification task
- **Feature Extraction Flexibility** : The option to extract features allows for direct use in other tasks like visualization or clustering

MAJOR LEARNINGS AND EXPERIENCES

- Learnt about **data cleaning** and processing, converting **audio files into csv** files using **MFCC** coefficients
- Learnt the implementation of **CNNs** on a given dataset using Python and extracting relevant **metrics**
- Learnt about how different types of models (supervised and unsupervised) affect **performance on a given dataset** and how to choose between them

HURDLES

- One of the first challenges encountered was to download such **memory-heavy files** into our computers in an organized manner
- The next hurdle was the constant struggle of trying out **different models** and obtaining the performance metrics
- This often took a long time because of the **large amount of data**
- **Logistical problems** included **common time management** for team members and allocation of work in an **equitable** manner

LINKS TO OUR SOURCE CODE

Successful attempts in Supervised Learning

https://github.com/PanavShah1/DS203-Final-Project-2

Failed attempts in Unsupervised Learning

https://github.com/PanavShah1/DS203-Final-Project

Songs and Models

https://drive.google.com/drive/folders/18-IlkQX2EEeA2t9u_julzuEavNPxJK1x?usp=drive_link

Thank You!