



University of Tehran
Faculty of Engineering
School of ECE

Voice Emotion Recognition

Report Sheet

Niloufar Mortazavi – 220701096
Alborz Mahmoudian – 810101514
Mahdy Mokhtari - 810101515

Introduction

In recent years, the ability of machines to understand and respond to human emotions has become a significant area of research, particularly in the field of Human-Computer Interaction (HCI). Among the various modalities for emotion detection, speech stands out due to its naturalness, availability, and rich emotional content. Speech Emotion Recognition (SER) refers to the process of automatically identifying the emotional state of a speaker from their voice signal. This technology has wide-ranging applications, including virtual assistants, call center analytics, mental health monitoring, smart tutoring systems, and more.

The goal of our project is to develop an effective pipeline for detecting emotions from audio data using machine learning and signal processing techniques.

Starting from raw speech recordings, we aim to preprocess the data, extract relevant acoustic features, and classify the emotional content using advanced statistical and machine learning methods.

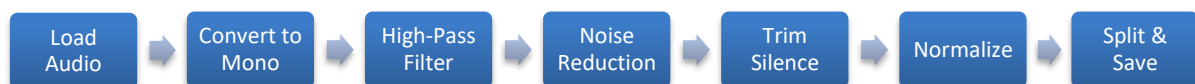
Dataset

CREMA-D (**Crowd-sourced Emotional Multimodal Actors Dataset**) is a widely used dataset for **Speech Emotion Recognition (SER)** research. It contains audio recordings of actors expressing different emotions, helping researchers analyze how speech characteristics correlate with emotions.

CREMA-D plays a crucial role in SER applications advancing these fields by providing a diverse and labeled dataset for training and evaluating machine learning models.

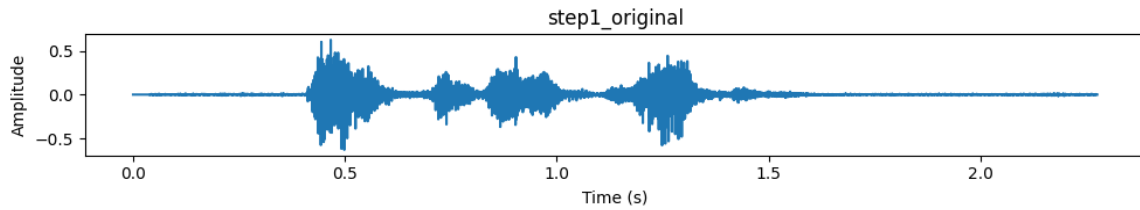
Preprocess Flow

The preprocessing flow in our project is designed to clean and standardize raw audio recordings by removing unwanted noise, normalizing volume levels, and segmenting the data into usable chunks. This step plays a crucial role in improving the performance of downstream tasks such as feature extraction and emotion classification by ensuring that the input data is both clean and uniform across all samples.



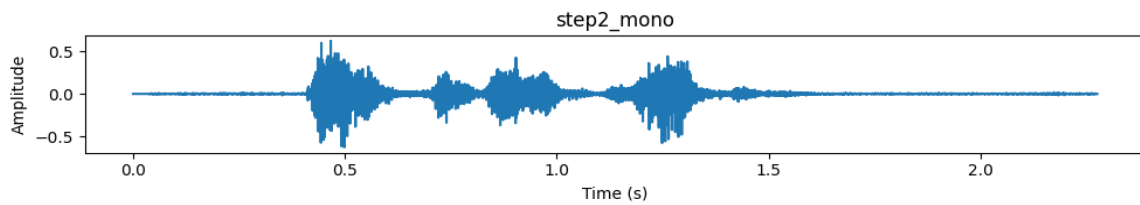
1. Loading Audio

We began by loading each audio file and resampling it to a standard sampling rate 16000 Hz. This ensured consistency across all files regardless of their original sample rates.



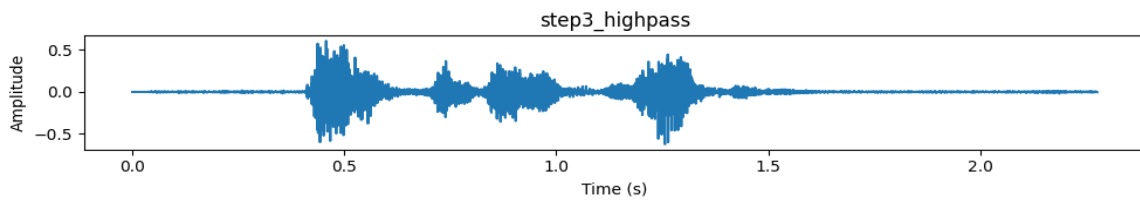
2. Converting to Mono

All audio signals were converted to mono-channel to simplify processing and reduce dimensionality, as stereo signals do not offer additional benefit for emotion classification.



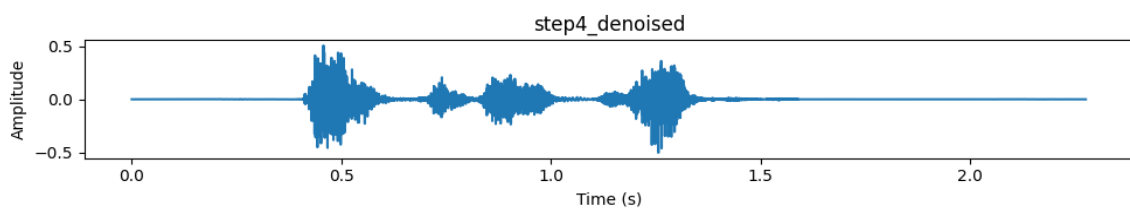
3. High-Pass Filtering

A high-pass filter was applied to eliminate low-frequency noise such as microphone rumble or background hum. This enhances the clarity of speech frequencies.



4. Noise Reduction

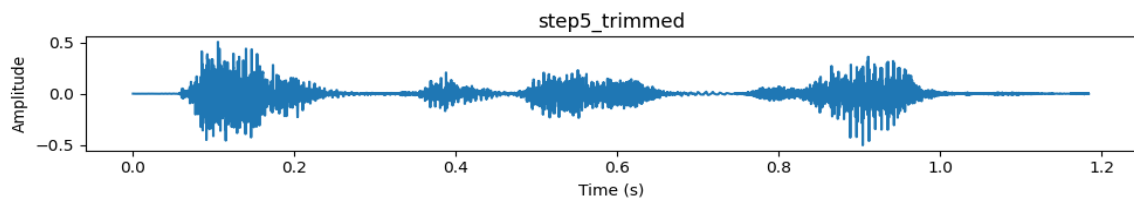
We applied noise reduction techniques to suppress background noise while preserving the main speech signal. This step significantly improves the signal-to-noise ratio.



5. Trim Silence

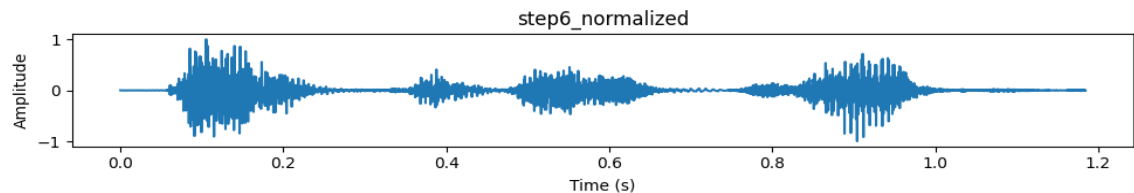
Silence at the beginning and end of audio files was trimmed to focus on the actual spoken content. This prevents the model from learning irrelevant pauses.

Why Trim Only Start/End Silence? Silence within speech is often meaningful — like pauses for breath, emphasis, or emotion. Removing those could distort natural patterns and affect emotion recognition accuracy. A pause in a sad voice feels different from a pause in an angry one. Removing internal silence could delete emotional cues that are critical for model performance



6. Normalizing

Finally, the audio signal was amplitude-normalized to ensure a consistent loudness level across all files. This is important for fair comparison during feature extraction. Models can get biased toward louder samples if amplitudes vary widely. Normalization removes this bias.



7. Split & Save

In our preprocessing pipeline, we divide each processed audio file into equal-length chunks of 0.8 seconds, with an overlap of 0.4 seconds.

Why We Split Audio into Fixed-Size Chunks?

the most important reason we split the audio into uniform chunks is to ensure that all feature vectors have the same dimensionality.

Many of our features are computed frame by frame. If chunks were of varying lengths, the resulting feature matrices would have inconsistent shapes, making it difficult to combine all data into one dataset and train machine learning models that require fixed-size input vector.

Why Chunk Duration = 0.8 Seconds?

Based on linguistic research, the average spoken word duration is ~0.4 seconds. To capture two full words per chunk, we selected 0.8 seconds to provide enough context for emotional expression, ensure the chunk isn't too short (losing meaning), or too long and to help the model learn better from meaningful units of speech.

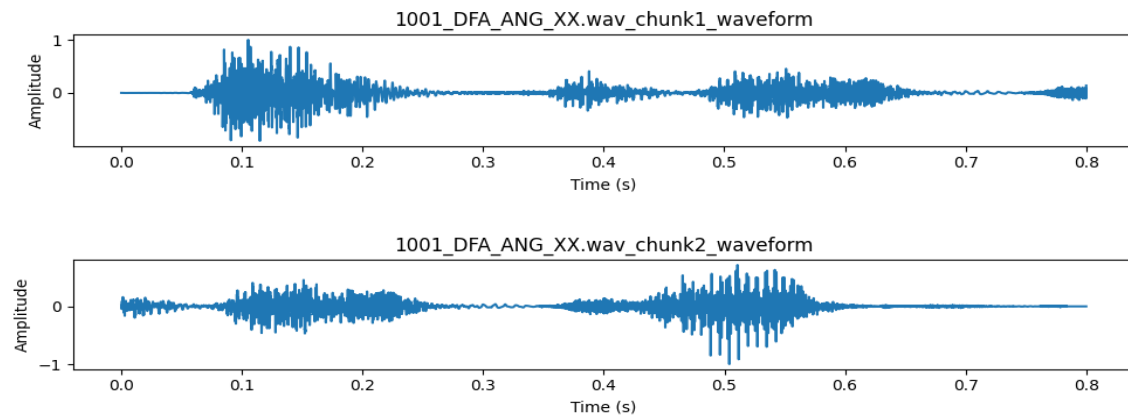
Why Overlap = 0.4 Seconds?

Instead of dividing the audio into non-overlapping chunks we deliberately use overlapping windows so that:

- The last word of one chunk becomes the first word of the next,
- The second chunk contains words 2 and 3 (not 3 and 4),
- This overlap helps preserve the relationship between consecutive words,
- Emotional cues, which often span across word boundaries, aren't lost,
- It captures transitions and emotional patterns that may occur between adjacent words.

This overlapping window helps in modeling the temporal dependency and emotional transitions between words, which are crucial for speech emotion recognition. If the last chunk of the audio is not enough long (0.8 s), we use zero padding at the end of the chunk.

This 1.2s audio is trimmed into 2 chunks, first from 0 to 0.8 seconds and the second from 0.4 to 1.2.



Feature Extraction

Feature Extraction Strategy

In our speech emotion recognition pipeline, we extract features frame-by-frame using a sliding window approach, instead of analyzing the whole audio clip at once. This design choice is critical for accurately capturing the temporal characteristics of speech.

Frame and Overlap Settings

- **Frame Size = 512 samples**

This corresponds to 32 milliseconds of audio at a sampling rate of 16,000 Hz.

- **Hop Length = 205 samples**

About 12.8 milliseconds between the start of consecutive frames. This results in ~60% overlap between frames.

So each 0.8 second audio chunk is divided into approximately 63 frames

$$= (0.8 \text{ sec} \times 16000 \text{ samples/sec} - 512) / 205 + 1$$

Why Frame-Wise?

Speech is non-stationary, meaning its characteristics (like pitch, loudness, frequency content) change quickly over time. Frame-wise extraction captures local patterns in energy, tone, and rhythm that are crucial for identifying emotions.

What If We Used Whole-Audio Features?

If we tried to compute features from the entire audio signal we would lose time-related changes, which are essential in speech. For example, two clips with the same average pitch but different emotional dynamics would look identical. Emotion is often expressed through change over time, not static values.

Why Do We Use Overlapping Frames?

Overlap between frames ensures that no important signal transition is missed. Without overlap, small but crucial shifts in speech (like sudden pitch or intensity changes) could fall between frames and be ignored.

Overlap provides a smoother and more continuous representation, which helps improve feature quality and downstream model accuracy.

Features

1) MFCCs (Mel-Frequency Cepstral Coefficients)

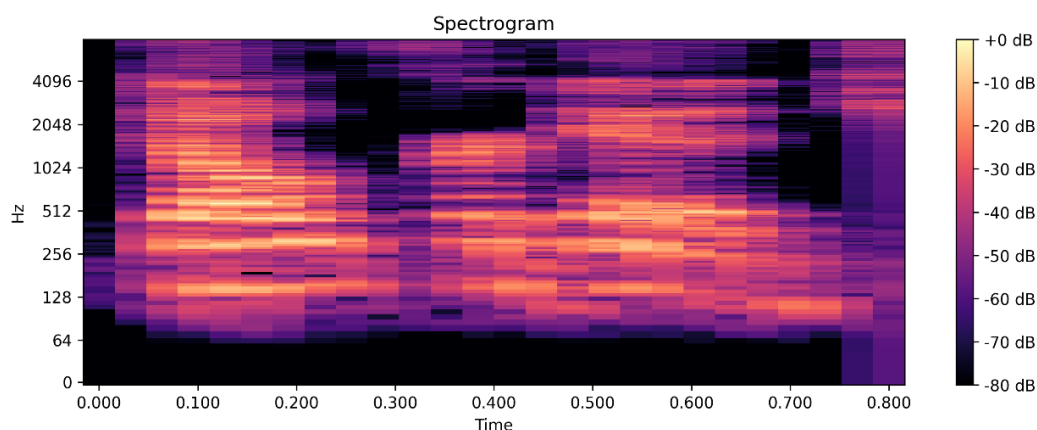
MFCCs are the most widely used features in speech and audio processing because they effectively model the shape of the vocal tract, which changes with different emotions. They capture timbre and overall spectral envelope of the sound.

How to calculate MFCC:



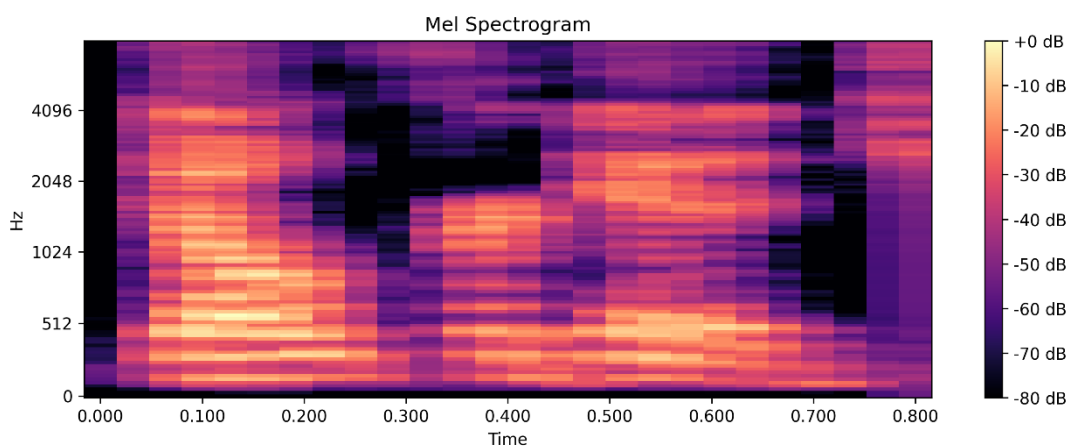
1) Short-Time Fourier Transform (STFT):

First, the audio signal is divided into short overlapping frames. For each frame, we compute its frequency spectrum using STFT. This results in a spectrogram, which shows how the frequency content of the signal changes over time.



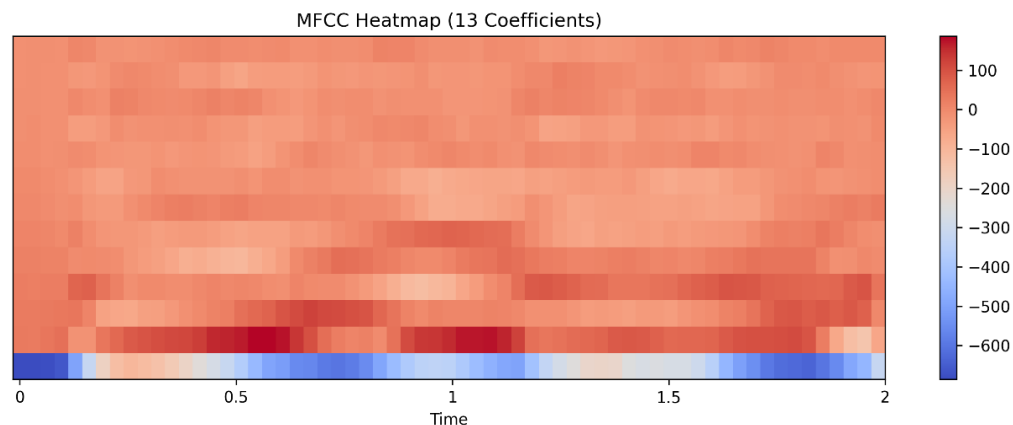
2) Mel Filter Bank Application:

Human hearing is more sensitive to lower frequencies than higher ones. So, instead of using a linear frequency scale, we apply a Mel filter bank to the spectrogram, which maps the frequencies to the Mel scale — a scale that better reflects human auditory perception.



3) Logarithmic Compression and DCT:

To match how humans perceive loudness, we apply a logarithm to the Mel spectrogram values. We then apply the DCT to the log-Mel values. This transforms the features into a set of coefficients that are less correlated and more compact. These are the MFCCs. Typically, only the first 13 MFCCs are kept, as they contain the most useful information about the shape and structure of the sound. Higher-order coefficients tend to capture noise and fine details that are not helpful for tasks like emotion recognition.



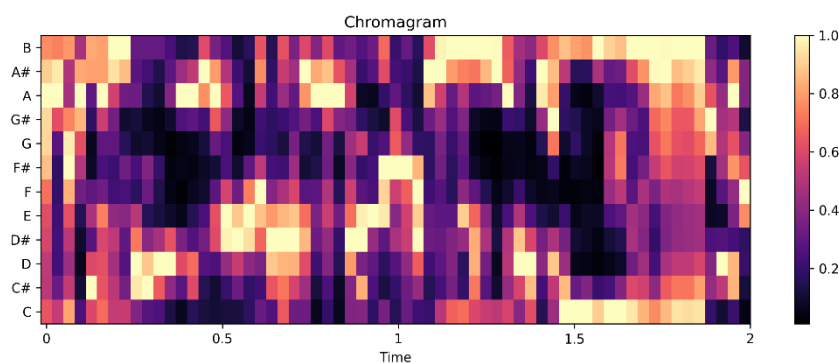
2. Chroma

Chroma feature (chroma energy) represent **how much energy** exists in each of the 12 pitch classes of music or sound: C, C#, D, D#, E, F, F#, G, G#, A, A#, B

But unlike musical notes (which care about pitch and octave), chroma features ignore the octave – they only care about which pitch class is present.

To calculate chroma, the audio signal is first converted into a frequency or spectrogram representation. Then, the energy in each frequency bin is mapped to one of the 12 chroma bins based on its pitch class.

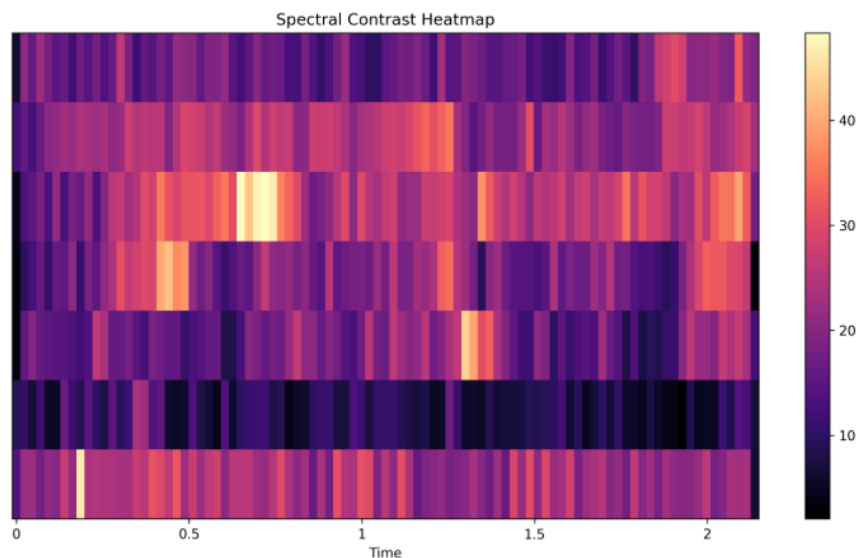
Finally, the result is a 12-dimensional feature vector that shows how much energy is present in each pitch class.



3. Spectral Contrast

Spectral contrast measures the difference between the loudest and quietest parts (peaks and valleys) in different frequency bands of a sound. It captures how “sharp” or “flat” the sound spectrum is in each part of the frequency range. In simple terms, Spectral tells you **how punchy or smooth** a sound is.

To calculate Spectral, each frame of your audio is split into several frequency bands (usually 7), and for each band: It finds the peak energy (strongest frequency) and the valley energy (weakest frequency), and then it calculates **contrast = peak – valley**



4. Zero-Crossing Rate (ZCR)

We use it because ZCR is a simple and fast way to detect signal noisiness or harshness. It captures the number of times the signal changes sign (crosses zero).

It is computed by counting zero crossings per frame. Higher ZCR often corresponds to **fricatives**, which are more common in emotions like **fear or anger**.

5. Root Mean Square (RMS) Energy

We use it because RMS gives a good estimate of **perceived loudness**. it captures the **energy level** of the audio.

It is computed by square root of the average of squared amplitude values in each frame.

Higher RMS values are common in **angry or excited** speech, while low RMS is typical in **sad or fearful** tones.

6. Spectral Centroid

We use it because it describes where the “**center of mass**” of the spectrum is.

It captures A proxy for **brightness or sharpness** of the sound.

It is computed using Weighted mean of frequencies present in the signal, weighted by their magnitude. Excited or angry speech often shifts the spectral centroid higher, indicating more high-frequency energy.

7. Spectral Bandwidth

We use it to understand the range of frequencies present in the signal. It captures Spread of the spectrum — tells us how much the signal is spread around its centroid.

It is Standard deviation of the spectral distribution. Wide bandwidth may reflect **emotional intensity**, while narrow bandwidth may indicate **calmness**.

8. Spectral Rolloff

We use It to mark the frequency below which a given percentage (typically 85%) of the total spectral energy lies. It Helps detect **voicing and noise content**.

It's computed Sum of spectral magnitudes is used to find the cutoff frequency.

Angry and energetic speech has **higher rolloff**, reflecting energy in higher frequencies.

Features Engineering

▪ Feature Aggregation and Dimensionality Reduction Strategy

After the splitting process, our dataset grew significantly. From the original ~7500 audio files, we obtained approximately 31,000 audio chunks, each with a fixed duration of 0.8 seconds. For each of these chunks, we extracted frame-wise features, where each audio segment resulted in a feature vector of size 37×63 — that is, 37 features computed over 63 time frames. While this approach preserves rich temporal information, it leads to a very high-dimensional dataset (over 2300 features per sample), which may increase computational cost, lead to overfitting and cause issues with memory and model generalization.

Our Solution: Smart Aggregation Based on Statistical Relevance

To reduce the dimensionality without losing important information, we decided to aggregate some of the features across frames — typically using the mean value. But instead of blindly aggregating all features, we used a statistical decision-based approach to determine which features should remain frame-wise and which should be aggregated.

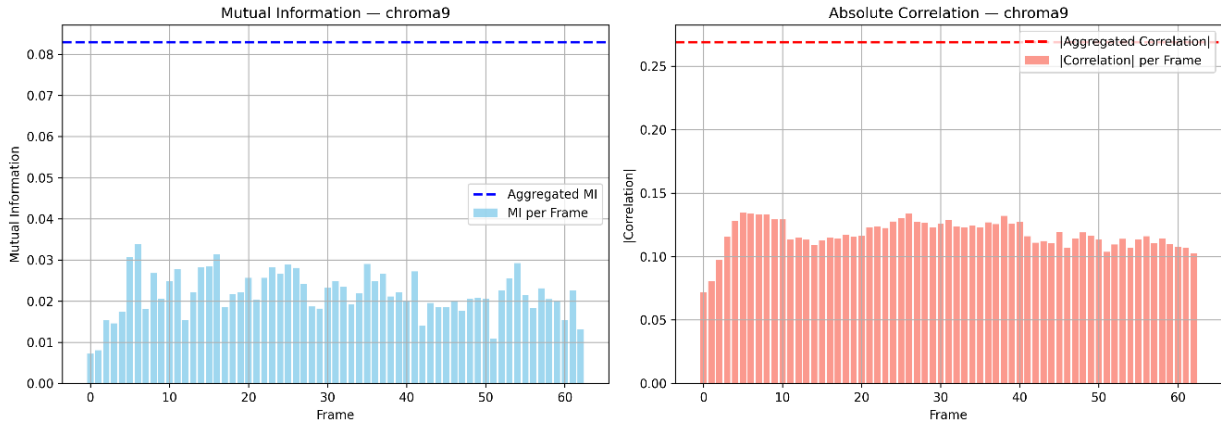
Step-by-Step Methodology:

1. For each feature, we considered its frame-wise values (63 values per chunk), and its aggregated version (e.g., mean over 63 frames)
2. For each frame of a feature, we calculated correlation with the target emotion label and Mutual Information (MI) with the target emotion label.
3. We also calculated the Correlation and MI between the aggregated version of the feature and the target label.
4. Decision Rule: If at least one frame of that feature had higher MI with the target than the aggregated version, we kept the frame-wise version of that feature. Otherwise, we replaced it with the aggregated version.

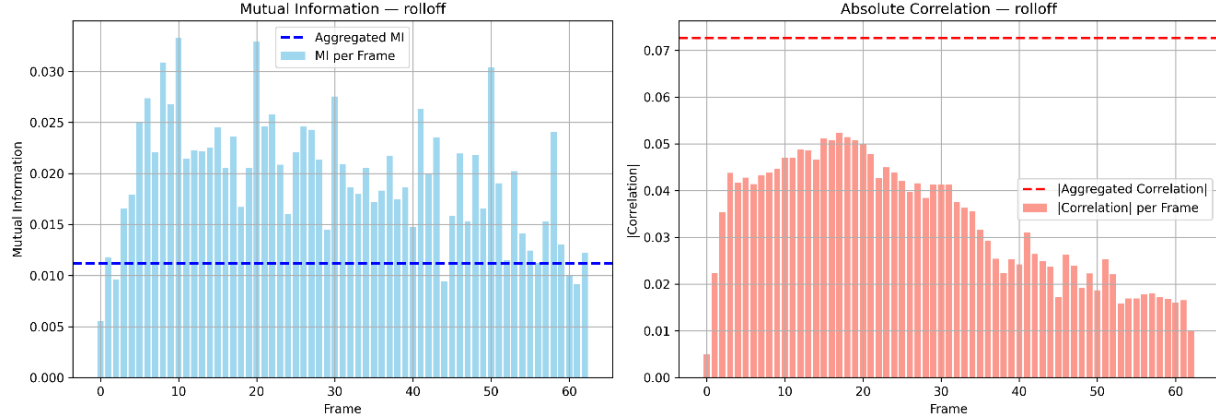
Why Mutual Information?

Mutual Information (MI) is a powerful measure that captures non-linear relationships between a feature and the target. This complements correlation, which captures only linear relationships. Using MI ensures we preserve subtle yet relevant frame-wise patterns that might be lost in aggregation

■ Rationale Behind Framewise vs. Aggregated Feature Retention



For the chroma9 feature, as shown in the top plots, none of the individual frames exhibit a Mutual Information value that exceeds the aggregated MI (dashed blue line), nor does any frame show a stronger correlation than the aggregated correlation (dashed red line). This suggests that the aggregated value of chroma9 carries more consistent and informative emotion-related signal than any single frame. Hence, we decided to **keep only the aggregated version of chroma9**.

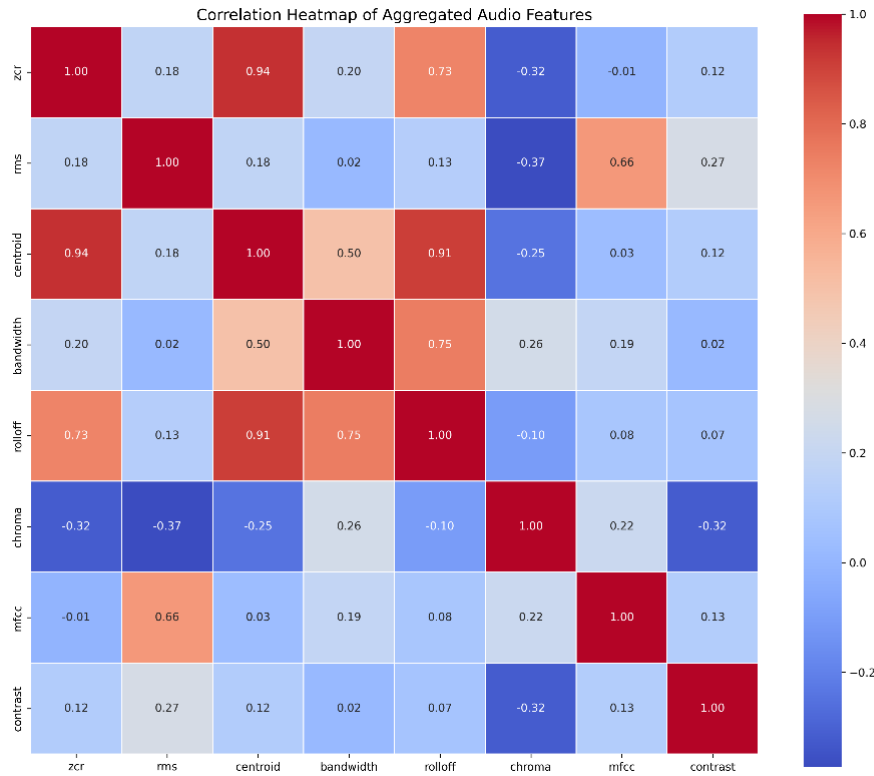


In contrast, the rolloff feature displays a different pattern. As the lower plots illustrate, multiple frames have higher MI than the aggregated version, and some even exceed the aggregated correlation. This indicates that certain frames capture emotion-specific patterns more effectively than a simple average. As a result, we chose to **retain the full framewise representation of rolloff** to preserve the temporal dynamics and capture richer information for classification.

This framewise vs. aggregate decision process was applied systematically across all features, helping us reduce dimensionality where possible, without sacrificing critical information.

▪ Feature Correlation

As a part of our project we should to analyze feature correlations and potentially remove or combine highly correlated ones based on statistical reasoning. The heatmap above illustrates the pairwise correlation between features. While it's evident that some features—like *zcr*, *centroid*, and *rolloff*—show strong linear relationships (e.g., correlations > 0.9), we made the conscious decision not to remove or merge any of them.



The reasoning lies in the nature of our features. Unlike traditional tabular datasets where features are often handcrafted or derived manually, our audio features are signal-engineered and grounded in well-established acoustic theory. Each feature captures a unique aspect of the signal—for example, *centroid* measures brightness, *rolloff* relates to spectral shape, and *zcr* to noisiness. Even if correlated, they might be offering complementary interpretations of the signal, especially in the context of emotional speech analysis.

Therefore, we preserved the full set of features, prioritizing signal richness and diversity over statistical redundancy.

▪ Handling of Missing Data

In our project, we did not encounter any missing or null values that required imputation or removal. This is because our pipeline was designed to start from a **clean and validated dataset**: the original CREMA-D dataset includes only valid, non-empty voice recordings. Furthermore, our preprocessing and feature extraction pipeline ensures that each audio segment is processed under controlled conditions.

As a result, all generated feature vectors are complete and contain no missing values. Thus, no additional handling of missing data was necessary or applicable in our case.

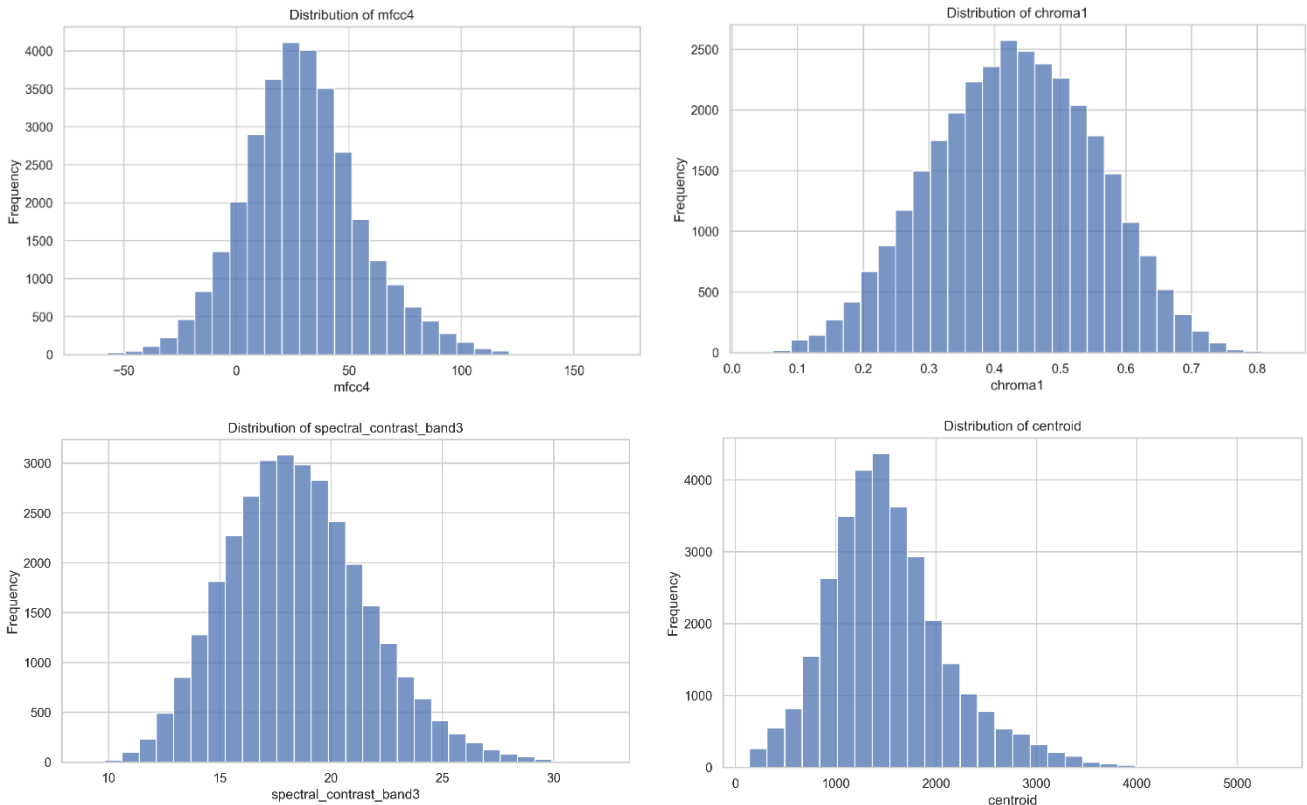
▪ Normalization and Standardization

In our project, normalization was applied to the audio waveforms before feature extraction to make all signals consistent in amplitude and avoid bias toward louder samples. Later, standardization is applied to feature vectors before feeding them into the model, to ensure all features contributed equally to learning. Actually, Standardization rescales the data to have:

- Mean = 0
- Standard deviation = 1

and we Use it when our data follows a Gaussian (normal-like) distribution.

Most of our feature distributions follow normal shape which we show it for some features here:



Docker Screenshots

docker file:

```
1 FROM python:3.8-slim
2
3 WORKDIR /app
4 COPY . /app
5
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 # Ensure ffmpeg is installed for pydub (audio processing)
9 RUN apt-get update && apt-get install -y ffmpeg
10
11 CMD ["python", "pipeline.py"]
12
```

.dockerignore:

```
.dockerignore > # Exclude Python bytecode and cache files
1 # unnecessary directories
2 StatisticalImages/
3 NoteBooks/
4 Helper/
5 Analytics/
6
7 # Exclude documentation files
8 *.md
9 *.pdf
10
11 # Exclude environment configuration files (if sensitive data is inside)
12 .env
13
14 # Exclude Git-related files
15 .git/
16 .gitignore
17
18 # Exclude Python bytecode and cache files
19 *.pyc
20 __pycache__/_
21
```

docker run:

```
features are extracted from : 1088_MTI_ANG_XX.wav_chunk2.wav
features are extracted from : 1054_IWL_DIS_XX.wav_chunk5.wav
features are extracted from : 1073_TSI_FEA_XX.wav_chunk2.wav
features are extracted from : 1089_IEO_HAP_LO.wav_chunk3.wav
features are extracted from : 1090_TSI_SAD_XX.wav_chunk5.wav
features are extracted from : 1031_ITS_DIS_XX.wav_chunk1.wav
features are extracted from : 1014_ITH_DIS_XX.wav_chunk3.wav
features are extracted from : 1011_IEO_SAD_LO.wav_chunk4.wav
features are extracted from : 1027_IWL_FEA_XX.wav_chunk4.wav
features are extracted from : 1089_IWL_HAP_XX.wav_chunk2.wav
features are extracted from : 1070_IEO_DIS_MD.wav_chunk2.wav
features are extracted from : 1086_TSI_NEU_XX.wav_chunk3.wav
```

docker build:

```
PS C:\Users\NoteBook\Desktop\programming\Data Science\Uni project\final project\MahdyMokh7-Speech-Emotion-Recognition-en-> docker build -t ser_pipeline_new .
[+] Building 5.8s (5/9)
-> [internal] load build definition from Dockerfile
-> transferring dockerfile: 289B
-> [internal] load metadata for docker.io/library/python:3.8-slim
-> [auth] library/python:pull token for registry-1.docker.io
-> [internal] load .dockerignore
-> transferring context: 373B
-> [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beeb13a0e4d073280665ea7be7f69ce2382f29c5a613f
-> [internal] load build context
-> transferring context: 182.73MB
```

docker desktop:

The screenshot shows the Docker Desktop application window. The left sidebar contains navigation options: Containers (selected), Images, Volumes, Builds, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' and displays a table of running containers. The table has columns for Name, Container ID, Image, Port(s), CPU (%), Last state, and Actions. There are 12 containers listed, all with 'CPU (%)' at 0% and 'Last state' as '17 hours' or '16 hours'. The 'Actions' column includes icons for stopping, refreshing, and deleting each container.

	Name	Container ID	Image	Port(s)	CPU (%)	Last state	Actions
<input type="checkbox"/>	silly_wing	665768293c3f	ser_pipeline		0%	17 hours	
<input type="checkbox"/>	reverent_cori	5f43def7bc5f	ser_pipeline		0%	17 hours	
<input type="checkbox"/>	confident_joliot	6a906d121255	ser_pipeline		0%	17 hours	
<input type="checkbox"/>	fervent_feynmar	d6da231b9d9b	ser_pipeline		0%	17 hours	
<input type="checkbox"/>	keen_brahmagu	3ebf0ab691a4	ser_pipeline		0%	17 hours	
<input type="checkbox"/>	awesome_jang	0cb5cdb14203	ser_pipeline		0%	16 hours	
<input type="checkbox"/>	festive_vollhard	7b0b0a68d4a9	ser_pipeline		0%	16 hours	

Showing 12 items

SQL Query Outputs

Tables_in_ser_db
features

Field	Type	Null	Key	Default	Extra
Unnamed: 0	bigint	YES		NULL	
file_name	text	NO	PRI	NULL	
emotion	int	NO		NULL	
emotion_category	varchar(255)	NO		NULL	
mfcc1	double	YES		NULL	
mfcc2	double	YES		NULL	
mfcc3	double	YES		NULL	
mfcc4	double	YES		NULL	
mfcc5	double	YES		NULL	
mfcc6	double	YES		NULL	
mfcc7	double	YES		NULL	
mfcc8	double	YES		NULL	
rms	double	YES		NULL	
bandwidth	double	YES		NULL	
zcr	double	YES		NULL	

Result 22 ×

Output

file_name	emotion	emotion_category	mfcc1	mfcc2	mfcc3	mfcc4
1001_DFA_ANG_XX.wav_chunk1.wav	0	ANG	0.168133467877063	0.6905728393381252	1.7526632173767145	-0.8972213913248078
1001_DFA_ANG_XX.wav_chunk2.wav	0	ANG	-0.0008494944163108	-0.2254671412399312	-0.4180653009717411	0.8652470226228293
1001_DFA_DIS_XX.wav_chunk1.wav	1	DIS	0.9828695484967968	0.3107446698295734	0.1896959722281201	0.6908101174895847
1001_DFA_DIS_XX.wav_chunk2.wav	1	DIS	-0.4752519921242634	-1.4183581124540778	-0.5335567324395474	2.142991579145223
1001_DFA_FEA_XX.wav_chunk1.wav	2	FEA	1.469575376123614	-0.0502383103306846	-1.101076637006865	-1.362061398775759
1001_DFA_HAP_XX.wav_chunk1.wav	3	HAP	1.061626693299359	0.3508398734319353	-0.6939428571696992	-0.7343379330888589
1001_DFA_HAP_XX.wav_chunk2.wav	3	HAP	-1.899552089311316	-1.7771372488520536	-0.3964529540651041	-0.5810819978421736
1001_DFA_NEU_XX.wav_chunk1.wav	4	NEU	0.4956415358186649	1.4240829213501291	0.3465384628246278	-0.5350496425420994
1001_DFA_NEU_XX.wav_chunk2.wav	4	NEU	-0.9883370776395036	-0.6040041672588415	-1.0041089934630298	-0.1647652094293261
1001_DFA_SAD_XX.wav_chunk1.wav	5	SAD	-0.8138559681241118	1.2989646761858298	1.858305822686524	0.5754688537453851

file_name	emotion	emotion_category	mfcc1	mfcc2	mfcc3	mfcc4
1001_DFA_ANG_XX.wav_chunk1.wav	0	ANG	0.168133467877063	0.6905728393381252	1.7526632173767145	-0.8972213913248078
1001_DFA_ANG_XX.wav_chunk2.wav	0	ANG	-0.0008494944163108	-0.2254671412399312	-0.4180653009717411	0.8652470226228293
1001_DFA_DIS_XX.wav_chunk1.wav	1	DIS	0.9828695484967968	0.3107446698295734	0.1896959722281201	0.6908101174895847
1001_DFA_DIS_XX.wav_chunk2.wav	1	DIS	-0.4752519921242634	-1.4183581124540778	-0.5335567324395474	2.142991579145223
1001_DFA_FEA_XX.wav_chunk1.wav	2	FEA	1.469575376123614	-0.0502383103306846	-1.101076637006865	-1.362061398775759
1001_DFA_HAP_XX.wav_chunk1.wav	3	HAP	1.061626693299359	0.3508398734319353	-0.6939428571696992	-0.7343379330888589
1001_DFA_HAP_XX.wav_chunk2.wav	3	HAP	-1.899552089311316	-1.7771372488520536	-0.3964529540651041	-0.5810819978421736
1001_DFA_NEU_XX.wav_chunk1.wav	4	NEU	0.4956415358186649	1.4240829213501291	0.3465384628246278	-0.5350496425420994
1001_DFA_NEU_XX.wav_chunk2.wav	4	NEU	-0.9883370776395036	-0.6040041672588415	-1.0041089934630298	-0.1647652094293261

file_name	bandwidth
-----------	-----------

Result Grid			Filter Rows:
	emotion_category	Num_Audios	
▶	ANG	4517	
	DIS	6051	
	FEA	5307	
	HAP	4362	
	NEU	4663	
	SAD	6597	

Result Grid			Filter Rows:
	emotion_category	emotion	
▶	ANG	0	
	DIS	1	
	FEA	2	
	HAP	3	
	NEU	4	
	SAD	5	

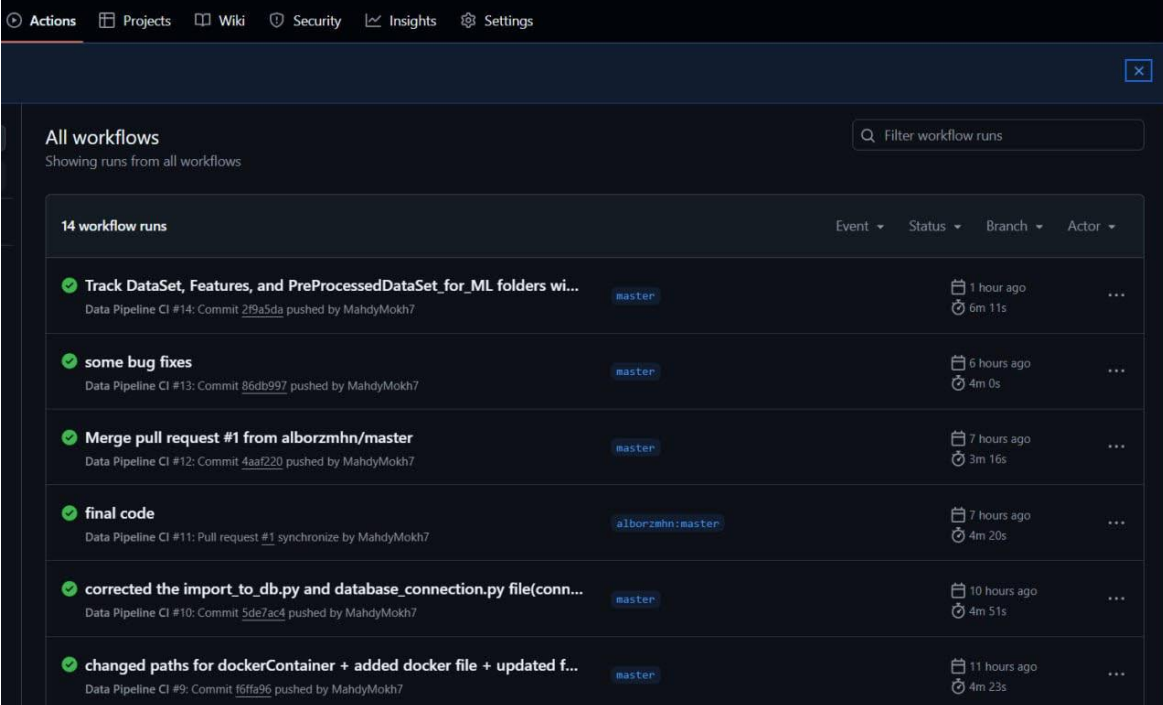
	Total_Num_audios
▶	31497

Result Grid			Filter Rows:	Export:
	emotion_category	avg_rms		
▶	HAP	0.13895535125749117		
	ANG	0.089801634859	0.08980163485940307	
	FEA	0.08850301110249122		
	NEU	-0.023928072199518206		
	SAD	-0.059158441477980106		
	DIS	-0.16189024420251932		

	emotion_category	avg_zcr
▶	ANG	0.447793659584514
	DIS	0.15669437089523713
	HAP	-0.009099448989954214
	NEU	-0.13259352242802055
	FEA	-0.13633898752503512
	SAD	-0.2409148401180736

Result Grid			Filter Rows:
	emotion_category	avg_chroma10	
▶	SAD	0.4846833341529695	
	ANG	-0.7122391013479231	

Github Actions Screenshots



The screenshot displays the GitHub Actions interface. At the top, there is a navigation bar with links for Actions, Projects, Wiki, Security, Insights, and Settings. Below this, the 'All workflows' section is visible, with a search bar labeled 'Filter workflow runs'. The main content area shows a list of 14 workflow runs. Each run is represented by a row with a green checkmark icon, a title, a description, a branch name, and a status (1 hour ago, 6m 11s, etc.).

Event	Status	Branch	Actor
Track DataSet, Features, and PreProcessedDataSet_for_ML folders wi...	1 hour ago	6m 11s	...
some bug fixes	6 hours ago	4m 0s	...
Merge pull request #1 from alborzmhn/master	7 hours ago	3m 16s	...
final code	7 hours ago	4m 20s	...
corrected the import_to_db.py and database_connection.py file(conn...	10 hours ago	4m 51s	...
changed paths for dockerContainer + added docker file + updated f...	11 hours ago	4m 23s	...