

Speech Emotion Recognition Using Deep Learning

Dataset: RAVDESS

Isuru Dananjaya

(06th May 2020)

Table of Content

Chapter 1: Introduction	4
Chapter 2: Overview.....	5
2.1 Customers	5
2.2 Platform.....	6
2.3 Development Responsibility	6
Chapter 3: Research and Reference Material.....	7
Chapter 4: Pipeline of The Methodology.....	9
4.1 Dataset Selection.....	9
4.2 Idea Generation	10
4.3 Base Proceeding and Methodology	10
Chapter 5: Code Base Explanations of Important Sections.....	12
5.1 Main.py	12
5.2 Data_loading.py	15
5.3 Classes_and_adjustments.py.....	16
5.4 Data_cleaning.py	17
5.5 Configurations.py.....	18
5.6 Build_features.py	19
5.7 Models.py.....	21

Table of Figures

Figure 5.1: main.py, part 01	14
Figure 5.2: main.py, part 02.....	14
Figure 5.3: data_loading.py	15
Figure 5.4: classes_and_adjustments.py	16
Figure 5.5: data_cleaning.py	17
Figure 5.6: envelope function	18
Figure 5.7: configurations.py	19
Figure 5.8: build_features.py 01	20
Figure 5.9: build_features.py 02	20
Figure 5.10: models.py	21

List of Acronyms and Abbreviation

CNN	Convolutional Neural Network
FFT	Fast Fourier Transform
MFCC	Mel Frequency Cepstral Coefficients

Chapter 1: Introduction

This document lays out information on the project background and plan for the development of ‘mlcs-speech_emotion_recognition’ open source repository system by Isuru Dananjaya.

The intended readers of this document are current and future developers who are interested in audio classification, speech recognition, speech emotion recognition and the use and application of machine learning to achieve those said areas. This plan will, first, include, an overview of the project and what is the end goal, the directed audience, platform and development responsibility. Then this document will present the research and reference materials used to understand and develop this program. Next it will state information about the selected dataset, how the idea came by and the overall implementation methodology. Finally, a brief explanation is given on scripts and function which were considered to be important.

Chapter 2: Overview

Note: Overview covers the proposed system which is yet to be achieved. This is the first step towards the final objective.

In today's world, mental stress is considered a major factor that affects the human health and causes detriment to it. Mental stress can be imposed on a person, mostly, due to demands placed upon them which they cannot achieve either due to the lack of resources at hand or the lack of motivation to achieve them. Long term exposure to high levels of stress will unfailingly introduce complications with time, such as emotional, behavioral and physical complications. Addressing stress in this digital era can be challenging but still not impossible.

The first step in any task is information gathering. Accordingly, to treat stress, first it should be identified. There are some techniques available at present which are highly effective at this. These vary from the use of blood tests, heart rate variability, electroencephalogram, electrocardiogram, finger temperature etc. But the problem with stress is that with invasive mechanisms such as these used to identify stress may either be inconsistent enough to provide accurate readings or/and is invasive, mentally or physically, which in turn will cause the actual stress to be changed which will end up again in inaccurate results. So as even in cyber security, one should try to gather as much information as possible via passive/ noninvasive mechanisms. What this system aims to do is just that. The aim is to develop an end to end final product which can identify stress by analyzing speech emotion of the client.

The aim of this particular repository system is to analyze audio speech recording to identify whether the emotion being expressed in the audio is a positive or a negative emotion. At the end a person can upload an audio file through the web interface and a prediction based on that will be given to the user.

2.1 Customers

Current System: Anyone who needs to find whether the emotion he/she is having is either negative or positive.

Final Proposed System: Anyone who believes he/she is suffering from stress (ranging from child to an old-age person)

2.2 Platform

Programming Language: Python

Development: Development has been done on PyCharm IDE

Deployment: Deployment has been done as a local host web application

2.3 Development Responsibility

I, Isuru Dananjaya, would be developing the product/program/software and I am responsible for the creation of all things related.

Chapter 3: Research and Reference Material

This speech emotion recognition implementation is heavily dependent on audio classification, and in turn, heavily relies on the programmers understanding of most of the theories relating to audio. Please find below some of the most important papers and sites that I, Isuru Dananjaya, referred to get some understanding of the theories related to audio. (Please note that these alone, will not suffice to get a good understanding. I have only added a limited number of resources that I considered as the best to be shared with where by reading them you will be forced to search for some other keywords which, in turn, will lead to an overall better understanding of the subject area)

- 1) <https://www.intechopen.com/books/social-media-and-machine-learning/automatic-speech-emotion-recognition-using-machine-learning>
- 2) <https://medium.com/@SeoJaeDuk/automatic-speech-emotion-recognition-using-machine-learning-f324dd414ea4>
- 3) <https://github.com/MITESHPUTHRANNEU/Speech-Emotion-Analyzer>
- 4) <https://towardsdatascience.com/speech-emotion-recognition-with-convolution-neural-network-1e6bb7130ce3>
- 5) https://github.com/rezachu/emotion_recognition_cnn
- 6) <https://medium.com/analytics-vidhya/tackle-almost-any-audio-classification-challenge-with-this-34a1d0ac82b9>
- 7) <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>
- 8) <https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520>
- 9) <https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd>
- 10) <https://medium.com/@LeonFedden/comparative-audio-analysis-with-wavenet-mfccs-umap-t-sne-and-pca-cb8237bfce2f>
- 11) <https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html>
- 12) <http://datagenetics.com/blog/november32012/index.html>
- 13) <https://github.com/seth814/Audio-Classification>

Note that, out of these 3, 5 and 13 repositories helped immensely in understanding how to implement such a system and credit is due for these three authors.

Chapter 4: Pipeline of The Methodology

4.1 Dataset Selection

The dataset selected for this project was ‘The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)’ (find it [here](#)). This dataset contains files which represents 24 professional actors (12 females and 12 males), vocalizing two lexically-matched statements in a neutral North American accent. Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions, and song contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. All conditions are available in three modality formats: Audio-only (16bit, 48kHz .wav), Audio-Video (720p H.264, AAC 48kHz, .mp4), and Video-only (no sound). Note, there are no song files for Actor_18. For this project, ‘Audio-only’ files were selected.

Audio-only files of all actors (01-24), which were used for this project, are available in the site as two separate zip files (~200 MB each):

Speech file (Audio_Speech_Actors_01-24.zip, 215 MB) contains 1440 files:

$$60 \text{ trials per actor} \times 24 \text{ actors} = 1440.$$

Song file (Audio_Song_Actors_01-24.zip, 198 MB) contains 1012 files:

$$44 \text{ trials per actor} \times 23 \text{ actors} = 1012.$$

File naming convention:

Each file of the RAVDESS files has a unique filename. A filename consists of a 7-part numerical identifier (e.g., 03-01-02-01-01-01-01.wav).

These identifiers define the stimulus characteristics:

- 1st Identifier: Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
- 2nd Identifier: Vocal channel (01 = speech, 02 = song).
- 3rd Identifier: Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- 4th Identifier: Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.

- 5th Identifier: Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- 6th Identifier: Repetition (01 = 1st repetition, 02 = 2nd repetition).
- 07th Identifier: Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

For the example: 03-01-02-01-01-01-01.wav

1st identifier (03): Modality - audio-only

2nd identifier (01): Vocal channel - speech

3rd identifier (02): Emotion - calm

4th identifier (01): Emotional intensity - normal

5th identifier (01): Statement - "Kids are talking by the door"

6th identifier (01): Repetition - 1st repetition

7th identifier (01): Actor – 01 is an odd number representing a male

4.2 Idea Generation

As mentioned before, this repository contains the first step towards stress detection which is emotion recognition through human speech. The idea was to develop a robust program which will take the RAVDESS dataset and train itself to identify the emotion depicted in an audio file given by a customer. Note the word ‘robust’, for due to which the program has become much more complicated along the way.

In order to make the initial step a little bit easier, the program has been designed to only consider one gender the programmer specifies. But this has been programmed in such a way that in the end, with a much more developed system, the customer will be able to select his/her gender and the program will choose the relevant dataset and the training will happen according to that.

In order to introduce an end-to-end product, a front end has also been designed with local hosting.

4.3 Base Proceeding and Methodology

The program will first read all the audio file names and extract the necessary information from the identifiers such as the gender, emotion no etc. Next based on a class condition, data will be filtered

to assign a relevant emotion (If the condition is 2, two classes will be made as positive and negative). Then the filtered data will be adjusted to only preserve the data relating to the preferred gender (male or female). Then the 'emotion label' will be assigned as '<gender>_<emotion>'. The data will be then cleaned by resampling the rate and by removing excess durations where there is no sound detected. This cleaned data will be saved as audio files to a new directory and next time the identifier information will be collected from the audio files in the new directory rather than the original resources directory. Then after clean data has been loaded, signal time series, Fast Fourier Transform (FFT), Mel Filter Bank Coefficients and Mel-frequency cepstral coefficients (MFCC) features will be extracted to analyze the data further. After this section is done, the program will extract the MFCC based features from the data and build the feature set which will be then fed into a 2D Convolutional Neural Network (CNN) (No. of Epochs = 60, Batch Size = 2048) (Batch size was made larger so that the gradient descent will not fluctuate excessively). After the model has been trained and saved, a verification prediction will take place upon the same data used to train the model. These predictions will be saved to a csv file and checked manually later. After these are over, a separate prediction function is there to predict the emotional label of any new audio file. A front end has been designed using flask which is hosted as local host, where using this front end a customer can upload an audio file and check which emotion is being depicted from the audio signal of the uploaded file.

Chapter 5: Code Base Explanations of Important Sections

5.1 Main.py

The Figure 5.1 and Figure 5.2 showcase the main python script used to diagnose and build the application by the programmer. Note that this is not the one that will be running if a customer is to upload an audio file of his/her own.

In this main.py the line shown below,

```
data_info_df = dl.load_data_intel(fromwhere='original')
```

will call for 'load_data_intel' function implemented in the 'data_loading.py' script shown by the Figure 5.3. After loading the data into a dataframe, the next few lines are the attempts of trying to understand the dataframe and the signals of an audio file better.

```
# pc.plot_single_audio_wave(signal)
# pc.plot_single_audio_amplitude(signal, rate)
# pc.plot_single_audio_fft(signal, rate)
# plt.show()
```

Here, few functions are called in which were implemented to display some plots and charts in order to analyze the data.

```
df1 = ca.assign_emotion(data_info_df, 2)
# print(df1)
df1 = ca.isolate_by_gender(df1, gender='male')
# print(df1)
df1 = ca.remove_none_emotion(df1)
# print(df1)
df1 = ca.assign_classes(df1)
# print(len(df1))
# print(df1.head())
```

Here, the dataframe is being filtered and adjusted first to assign emotions such as either negative or positive (indicated by the 2) or else any other specification done in the 'classes_and_adjustments.py' script.

```
dc.data_cleaning(df1)
```

Here, the data is being cleaned where first they will be resampled to a lower sample rate and will be shortened by removing the unnecessary audio sections.

```
mconf = conf.ModelConfig(mode='convolutional')
rfpconf = conf.RandFeatParams(df1)
```

This part is to declare the configuration variables that are needed through out the program later on.

```
X, y = bf.build_rand_feat(randfeatparams=rfpconf, modelconfig=mconf)
```

This will build the final features, from the dataset, which will be sent through to the model to be trained.

```
model.fit(X, y, epochs=60, batch_size=2048, shuffle=True, class_weight=class_weight,
validation_split=0.1, callbacks=[checkpoint])
```

Here, the model will be trained with 60 epochs and with a batch size of 2048.

```
vp.verification_predict(df1)
```

Here, a verification will be carried out predicting on the same dataset that was used to train the model.

```
p.predict(modelconfig, loaded_model, classes)
```

Here, the function 'predict' is called which has being written to make predictions on any new audio file provided.

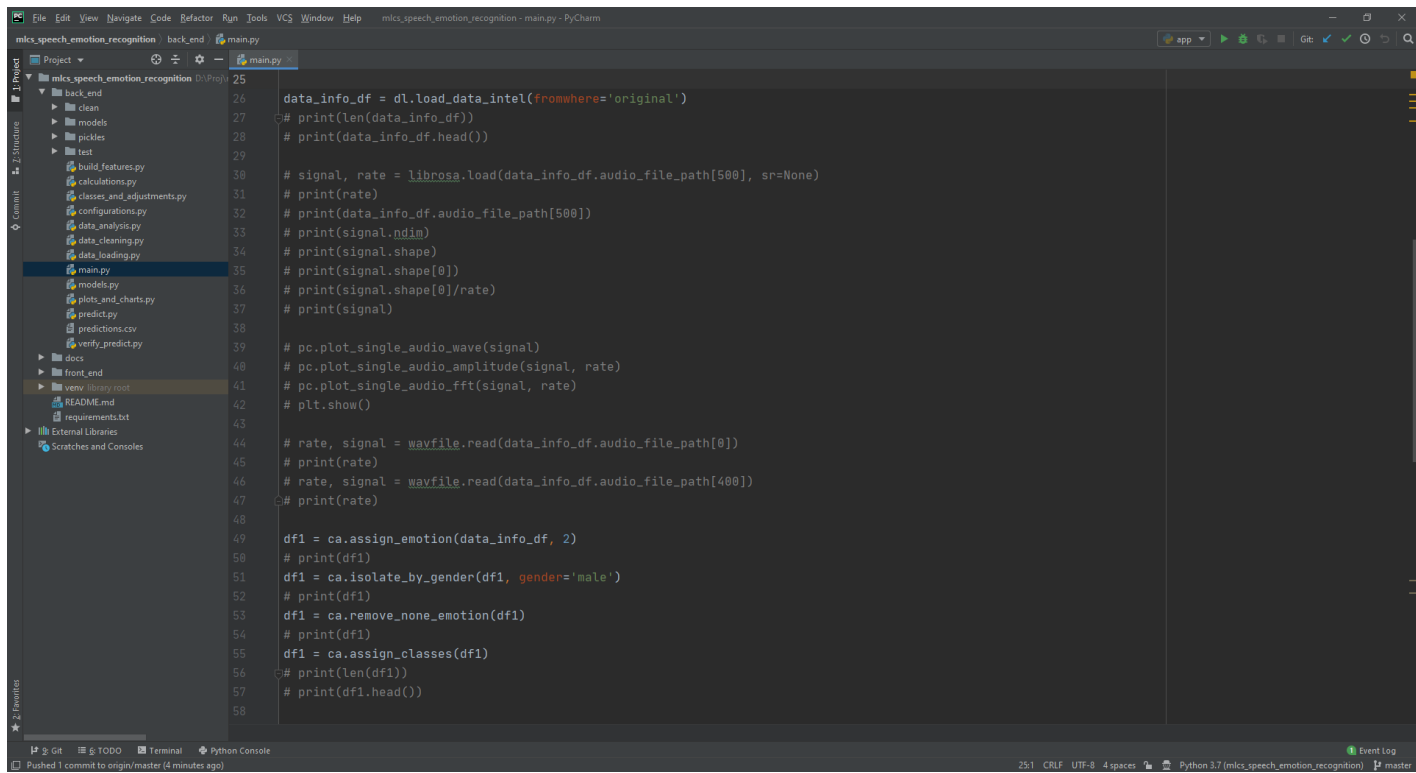


Figure 5.1: main.py, part 01

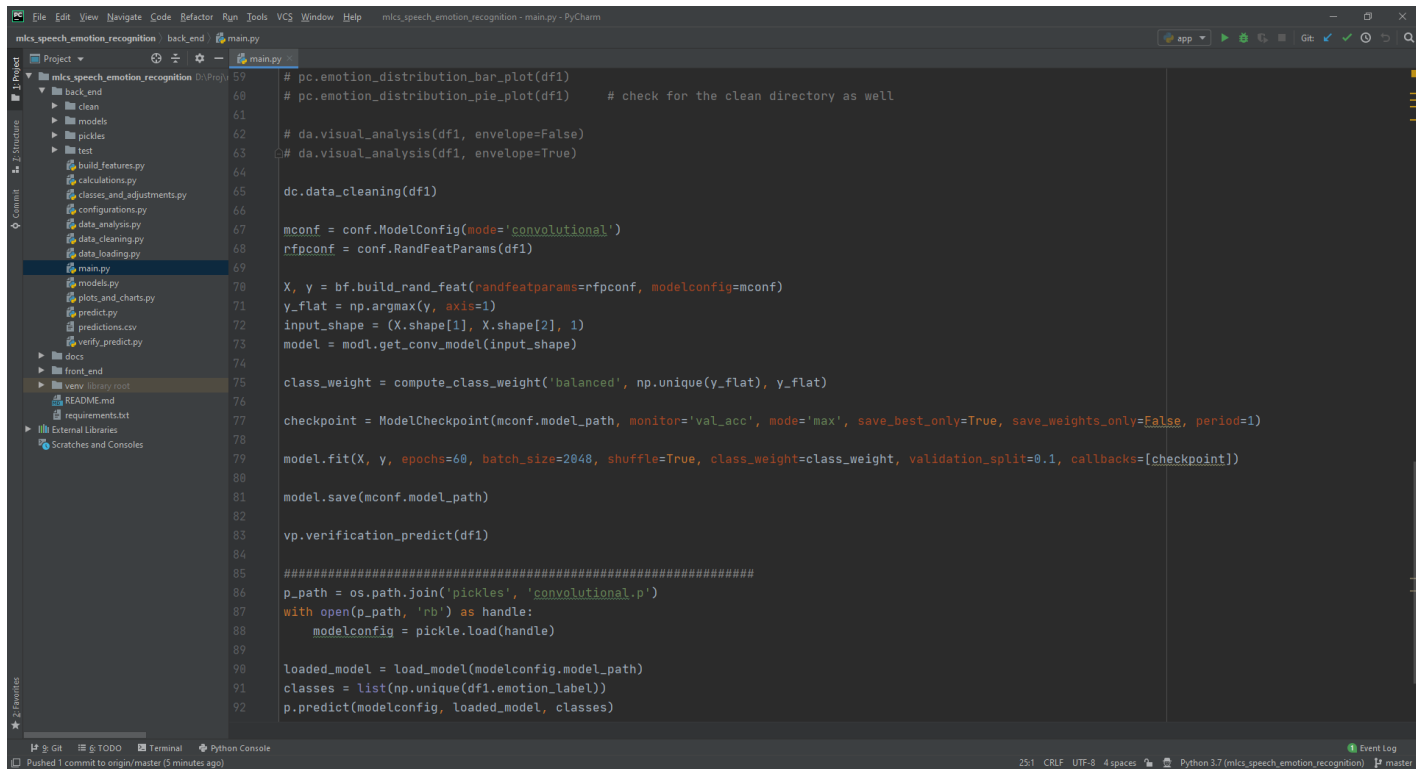


Figure 5.2: main.py, part 02

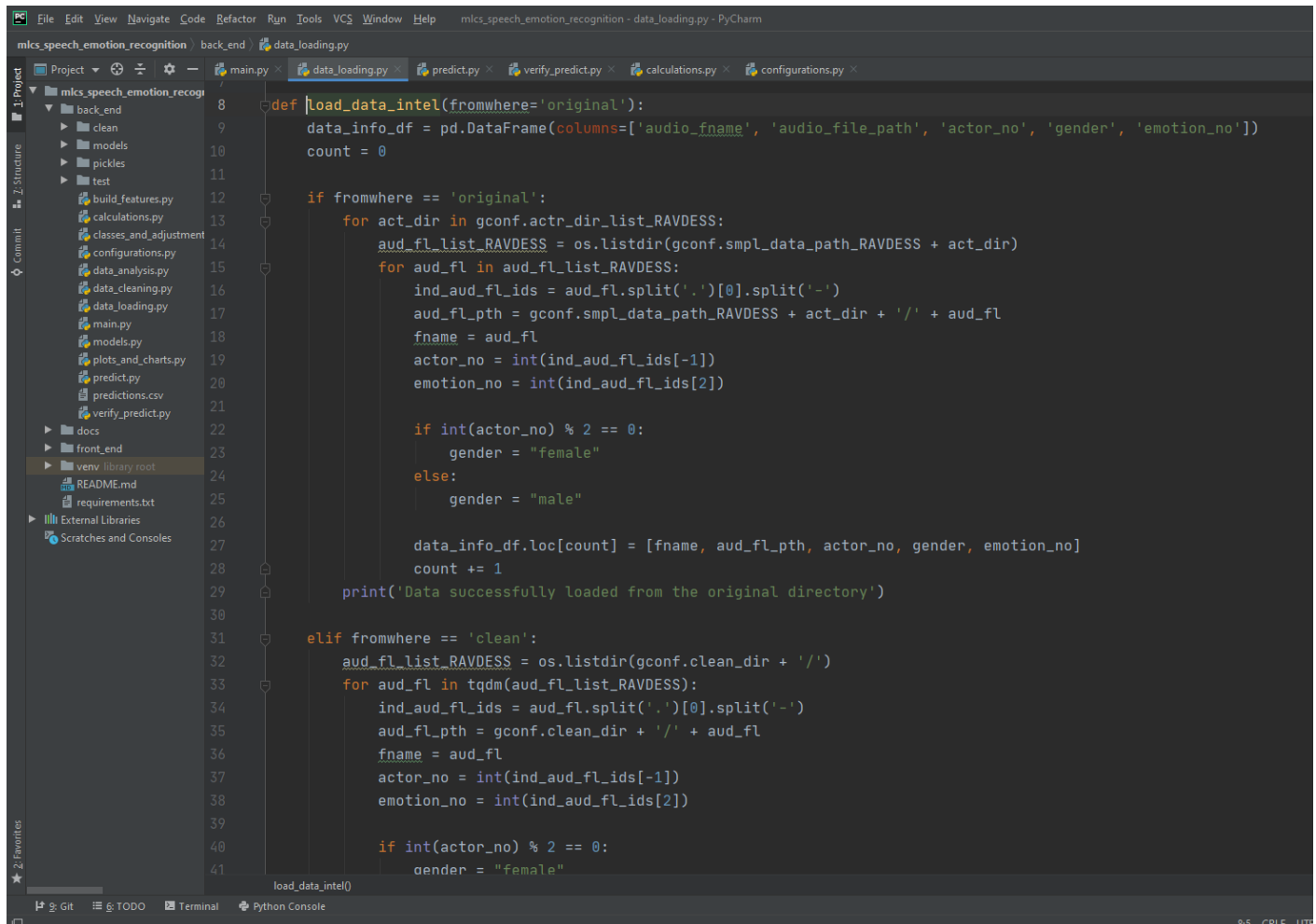
5.2 Data_loading.py

The Figure 5.3 showcase the function that has being built to load data from audio file identifiers to a dataframe.

```
if fromwhere == 'original':
```

```
elif fromwhere == 'clean':
```

The if conditions depicted will divert the program to either load information from the original audio file or from the cleaned audio files. The for loops inside will iterate through each audio file and derive the information needed.



```
def load_data_intel(fromwhere='original'):
    data_info_df = pd.DataFrame(columns=['audio_fname', 'audio_file_path', 'actor_no', 'gender', 'emotion_no'])
    count = 0

    if fromwhere == 'original':
        for act_dir in gconf.actr_dir_list_RAVDESS:
            aud_fl_list_RAVDESS = os.listdir(gconf.smpl_data_path_RAVDESS + act_dir)
            for aud_fl in aud_fl_list_RAVDESS:
                ind_aud_fl_ids = aud_fl.split('.')[0].split('-')
                aud_fl_pth = gconf.smpl_data_path_RAVDESS + act_dir + '/' + aud_fl
                fname = aud_fl
                actor_no = int(ind_aud_fl_ids[-1])
                emotion_no = int(ind_aud_fl_ids[2])

                if int(actor_no) % 2 == 0:
                    gender = "female"
                else:
                    gender = "male"

                data_info_df.loc[count] = [fname, aud_fl_pth, actor_no, gender, emotion_no]
                count += 1
            print('Data successfully loaded from the original directory')

    elif fromwhere == 'clean':
        aud_fl_list_RAVDESS = os.listdir(gconf.clean_dir + '/')
        for aud_fl in tqdm(aud_fl_list_RAVDESS):
            ind_aud_fl_ids = aud_fl.split('.')[0].split('-')
            aud_fl_pth = gconf.clean_dir + '/' + aud_fl
            fname = aud_fl
            actor_no = int(ind_aud_fl_ids[-1])
            emotion_no = int(ind_aud_fl_ids[2])

            if int(actor_no) % 2 == 0:
                gender = "female"
```

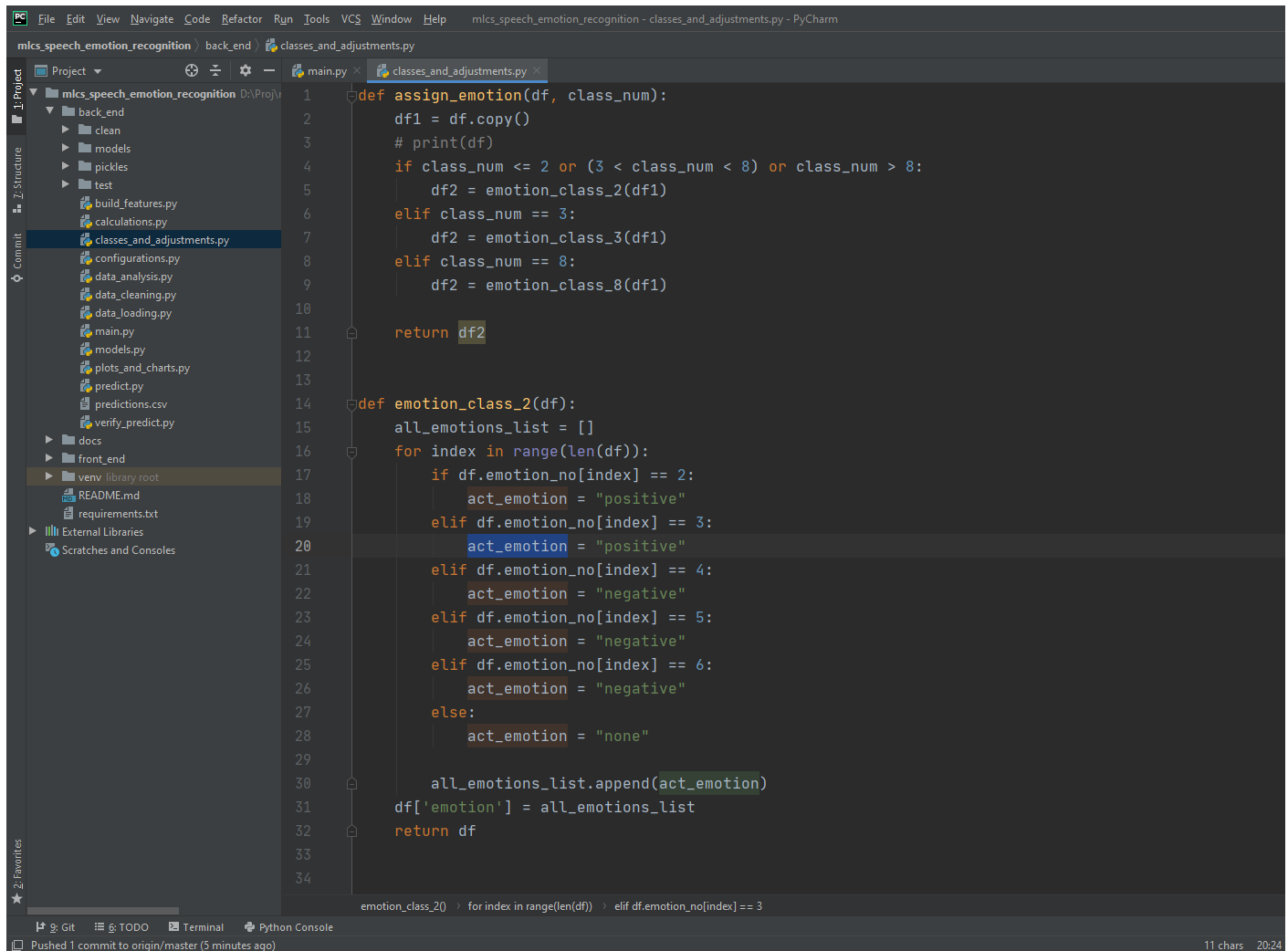
Figure 5.3: data_loading.py

5.3 Classes_and_adjustments.py

The Figure 5.4 showcase the python script written to assign emotions based on an argument provided.

```
def assign_emotion(df, class_num):
```

Here, the 'class_num = 2', effectively informs the function to divide data among two classes; positive and negative. Similarly, there are three ways of class divisions implemented on the same script.



```
1 def assign_emotion(df, class_num):
2     df1 = df.copy()
3     # print(df)
4     if class_num <= 2 or (3 < class_num < 8) or class_num > 8:
5         df2 = emotion_class_2(df1)
6     elif class_num == 3:
7         df2 = emotion_class_3(df1)
8     elif class_num == 8:
9         df2 = emotion_class_8(df1)
10
11     return df2
12
13
14 def emotion_class_2(df):
15     all_emotions_list = []
16     for index in range(len(df)):
17         if df.emotion_no[index] == 2:
18             act_emotion = "positive"
19         elif df.emotion_no[index] == 3:
20             act_emotion = "positive"
21         elif df.emotion_no[index] == 4:
22             act_emotion = "negative"
23         elif df.emotion_no[index] == 5:
24             act_emotion = "negative"
25         elif df.emotion_no[index] == 6:
26             act_emotion = "negative"
27         else:
28             act_emotion = "none"
29
30         all_emotions_list.append(act_emotion)
31     df['emotion'] = all_emotions_list
32     return df
33
34
```

Figure 5.4: classes_and_adjustments.py

5.4 Data_cleaning.py

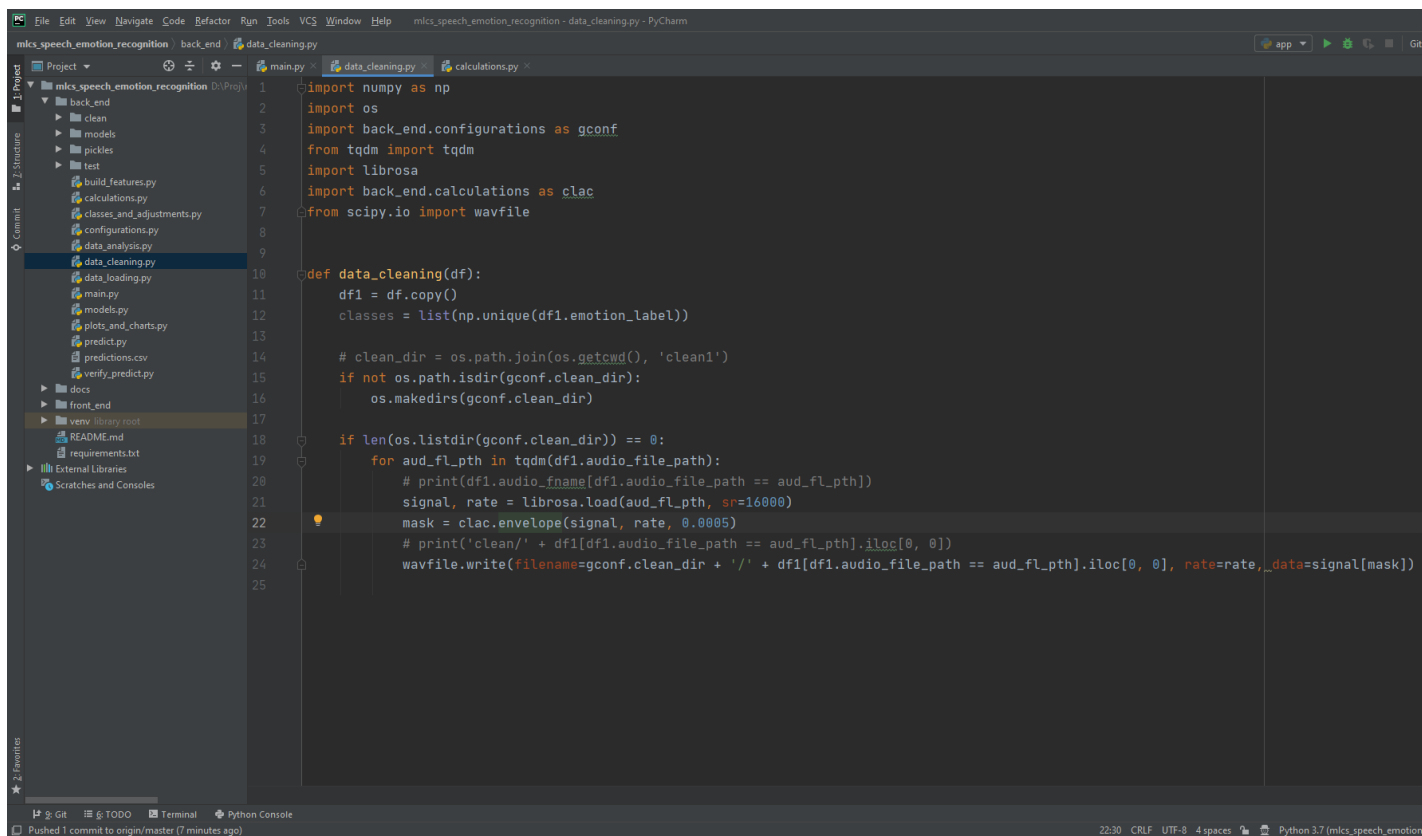
The Figure 5.5 showcase the data cleaning stage on the program where it will load audio files one by one, resample them to a lower rate to remove unnecessary high frequency ranges, (with speech and most audio, most changes happen at low frequencies),

```
signal, rate = librosa.load(aud_fl_pth, sr=16000)
```

and then will remove the extra seconds before and after the actual sound has being observed by using the 'envelope' function. Figure 5.6 show case the implementation of the 'envelope' function.

```
mask = clac.envelope(signal, rate, 0.0005)
```

After the data has been cleaned, they will be stored in a new directory called 'clean'.



```
1 import numpy as np
2 import os
3 import back_end.configurations as gconf
4 from tqdm import tqdm
5 import librosa
6 import back_end.calculations as clac
7 from scipy.io import wavfile
8
9
10
11 def data_cleaning(df):
12     df1 = df.copy()
13     classes = list(np.unique(df1.emotion_label))
14
15     # clean_dir = os.path.join(os.getcwd(), 'clean1')
16     if not os.path.isdir(gconf.clean_dir):
17         os.makedirs(gconf.clean_dir)
18
19     if len(os.listdir(gconf.clean_dir)) == 0:
20         for aud_fl_pth in tqdm(df1.audio_file_path):
21             # print(df1.audio_fname[df1.audio_file_path == aud_fl_pth])
22             signal, rate = librosa.load(aud_fl_pth, sr=16000)
23             mask = clac.envelope(signal, rate, 0.0005)
24             # print('clean/' + df1[df1.audio_file_path == aud_fl_pth].iloc[0, 0])
25             wavfile.write(filename=gconf.clean_dir + '/' + df1[df1.audio_file_path == aud_fl_pth].iloc[0, 0], rate=rate, data=signal[mask])
```

Figure 5.5: data_cleaning.py

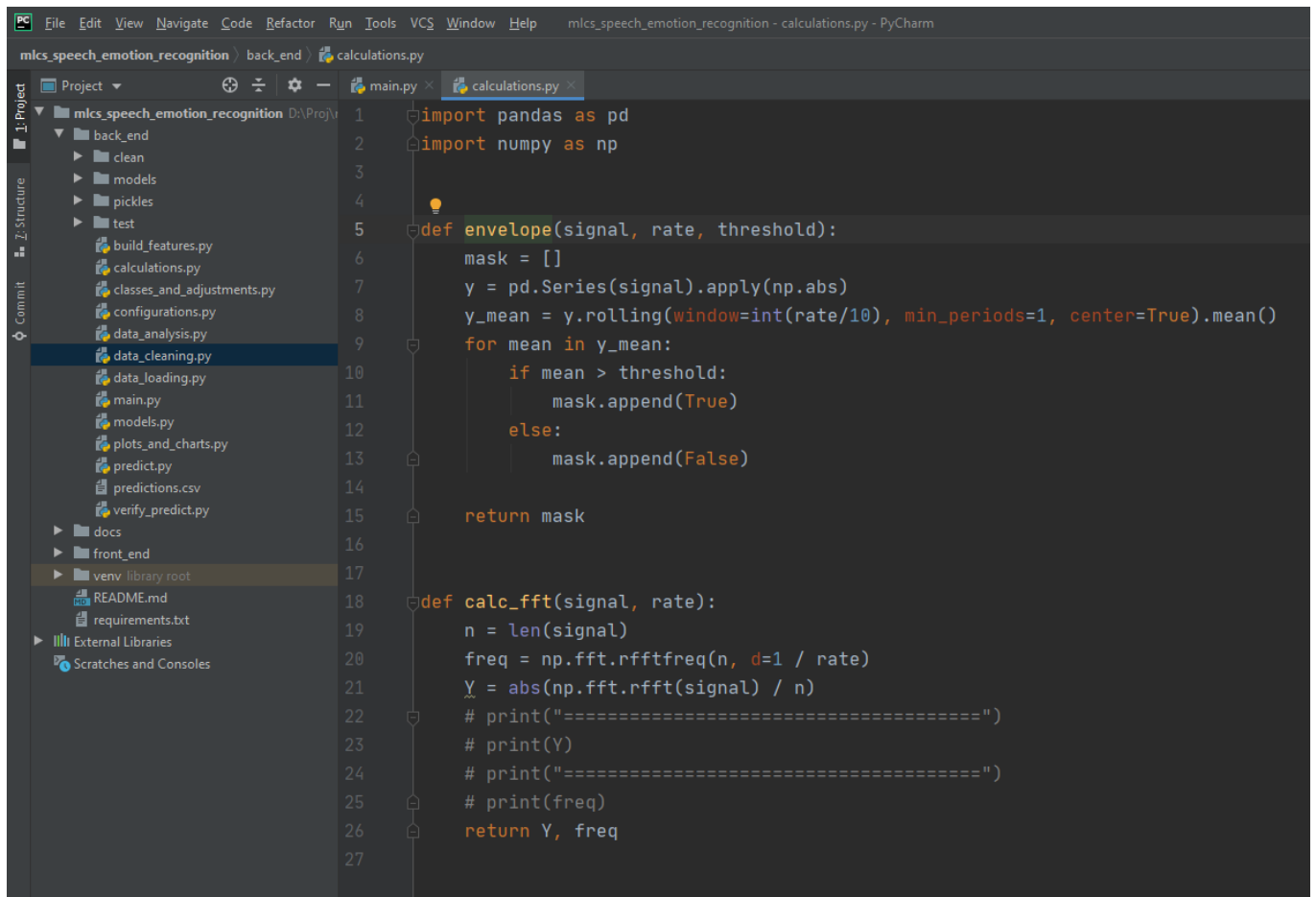


Figure 5.6: envelope function

5.5 Configurations.py

Figure 5.7 showcase the configurations script that is designed to hold all important variables that might be required by other function throughout the program repetitively.

```

smpl_data_path_RAVDESS = '../Resources/Datasets/RAVDESS_Audio/'
actr_dir_list_RAVDESS = os.listdir(smpl_data_path_RAVDESS)
clean_dir = 'clean'
tmp_test_dir = 'test'
uploads_dir = '../front_end/uploads'

```

These are the global variable which holds the paths and names of directories used through out the program.

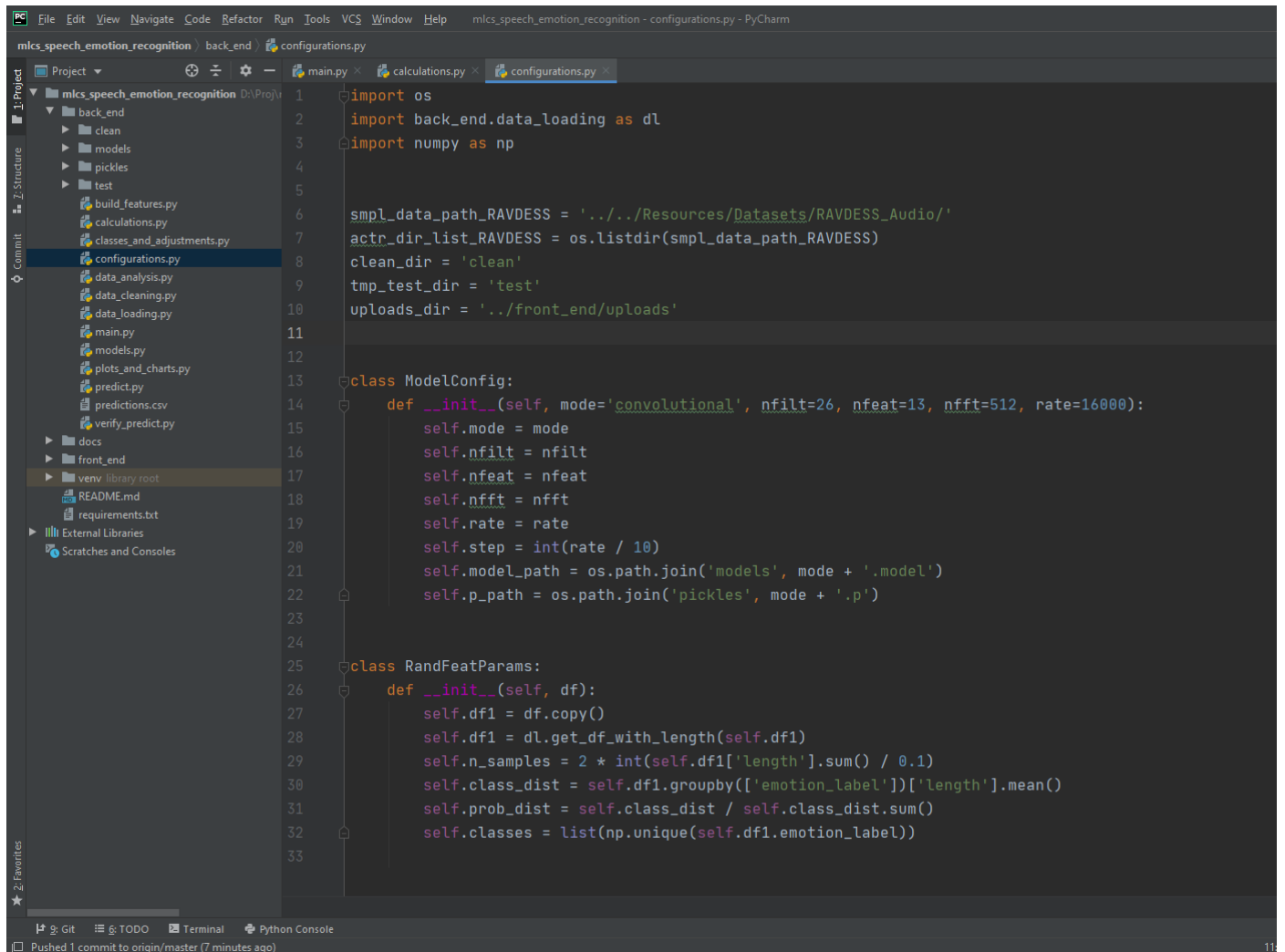


Figure 5.7: configurations.py

5.6 Build_features.py

The Figure 5.8 and Figure 5.9 showcase the python script that has being implemented to generate the features that will be passed to the CNN model.

```
rand_index = np.random.randint(0, signal.shape[0] - modelconfig.step)
```

The most important code line is here, where the samples are chosen randomly by dividing an audio file into 10th of a second chunks.

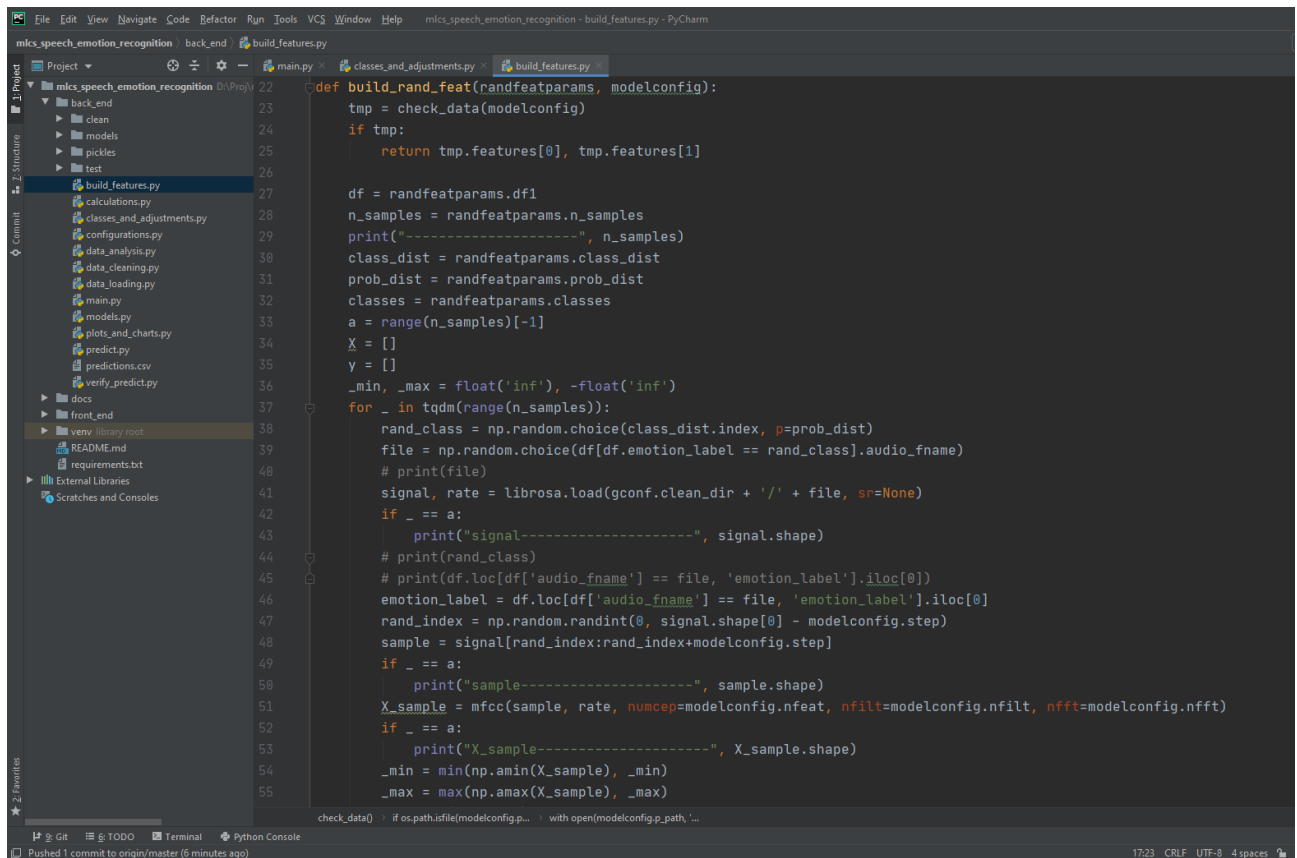


Figure 5.8: build_features.py 01

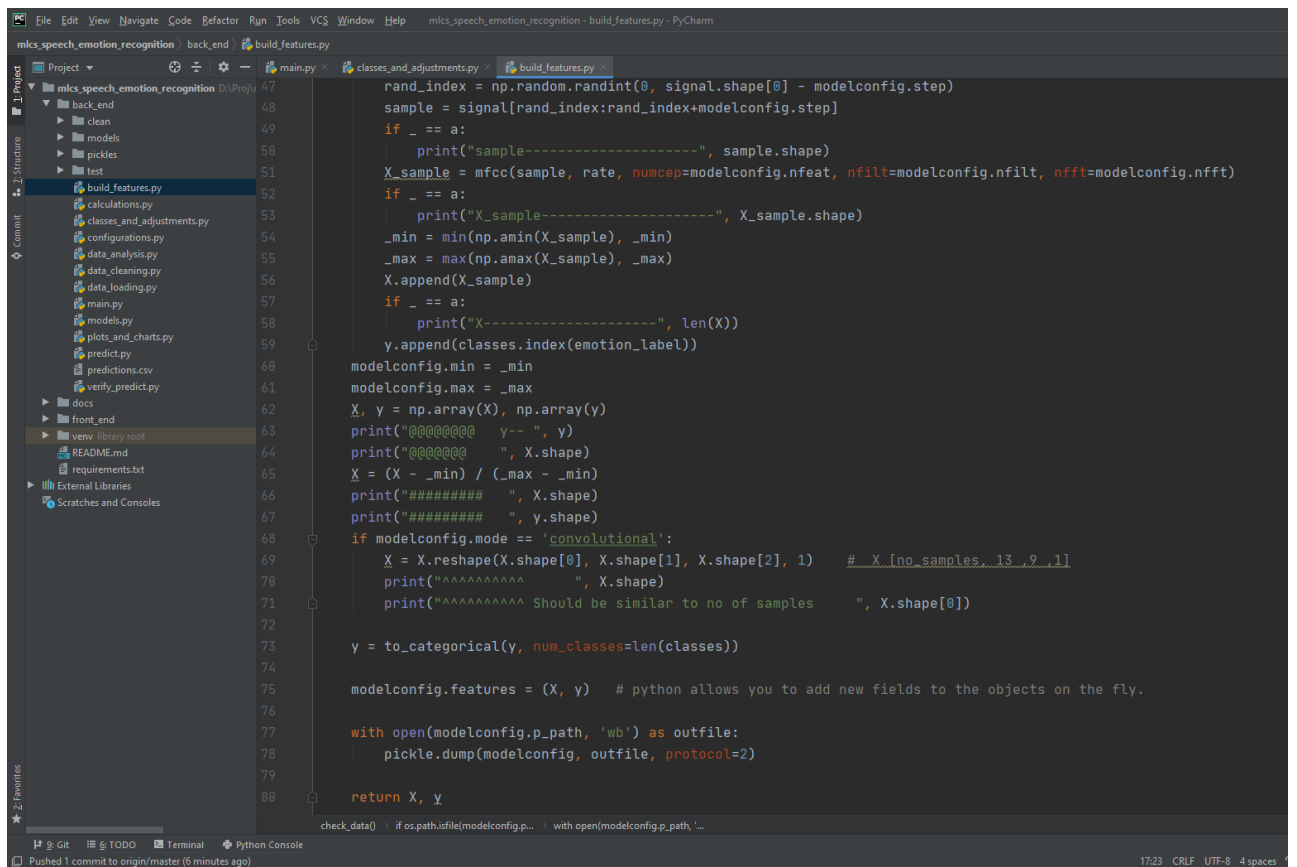
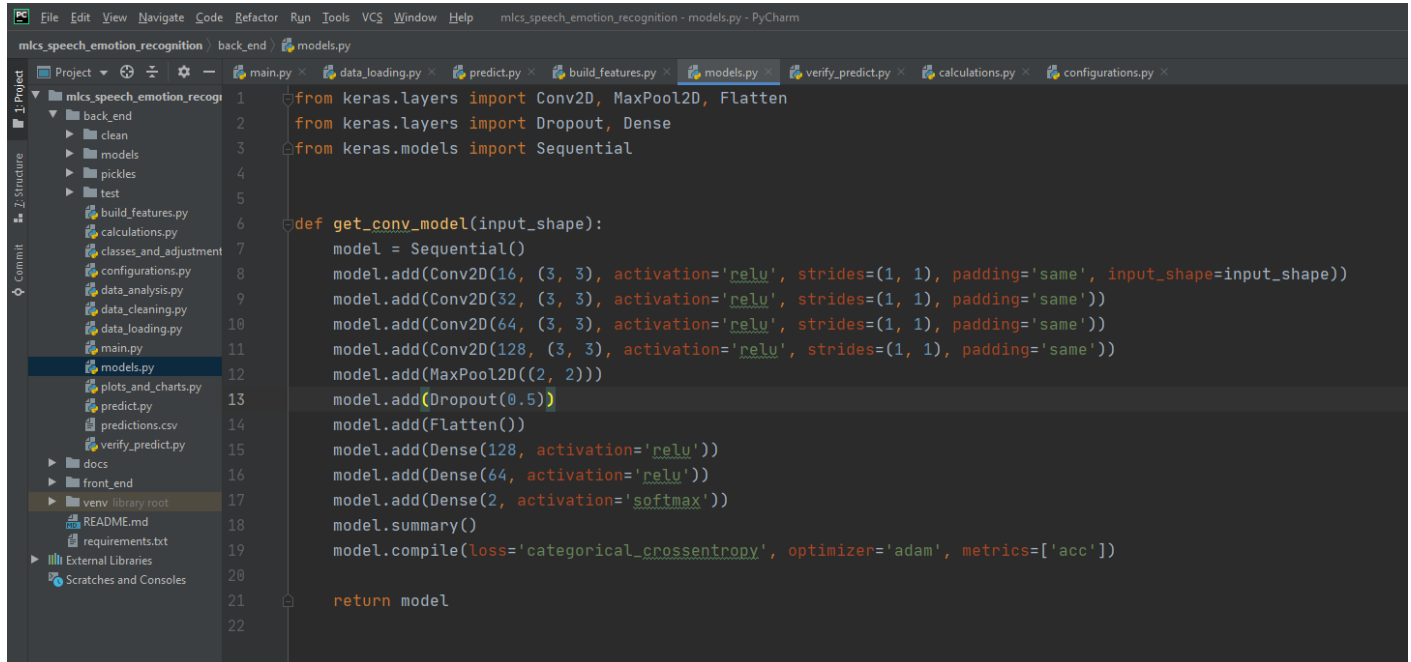


Figure 5.9: build_features.py 02

5.7 Models.py

The Figure 5.10 showcase the script where the CNN model has been defined.



```
1 from keras.layers import Conv2D, MaxPool2D, Flatten
2 from keras.layers import Dropout, Dense
3 from keras.models import Sequential
4
5
6 def get_conv_model(input_shape):
7     model = Sequential()
8     model.add(Conv2D(16, (3, 3), activation='relu', strides=(1, 1), padding='same', input_shape=input_shape))
9     model.add(Conv2D(32, (3, 3), activation='relu', strides=(1, 1), padding='same'))
10    model.add(Conv2D(64, (3, 3), activation='relu', strides=(1, 1), padding='same'))
11    model.add(Conv2D(128, (3, 3), activation='relu', strides=(1, 1), padding='same'))
12    model.add(MaxPool2D((2, 2)))
13    model.add(Dropout(0.5))
14    model.add(Flatten())
15    model.add(Dense(128, activation='relu'))
16    model.add(Dense(64, activation='relu'))
17    model.add(Dense(2, activation='softmax'))
18    model.summary()
19    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
20
21    return model
22
```

Figure 5.10: models.py