RECOGNIZE SPEECH EMOTIONS WITH LIBROSA

NOVEMBER 9

COMMUNITY INNOVATION WORKSPACE

About the Assembly

- A smart lab based out of In5 since Dec 2014
- Over 250 free workshops done
- ASSEMBLY: HACK - Embedded systems, iOT and hardware
- ASSEMBLY: CODE - Software projects - APIs, frameworks, apps
- Age range: 16-60 - students, professionals, entrepreneurs
- Focus on smart technology and practical applications
- Forum: members.theassembly.ae

TAG US ON OUR SOCIAL MEDIA

FACEBOOK The Assembly (@MakeSmartThings)

TWITTER @MakeSmartThings

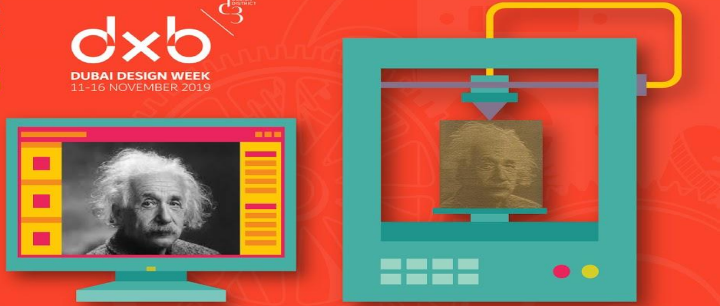
INSTAGRAM @MakeSmartThings

YOUTUBE The Assembly

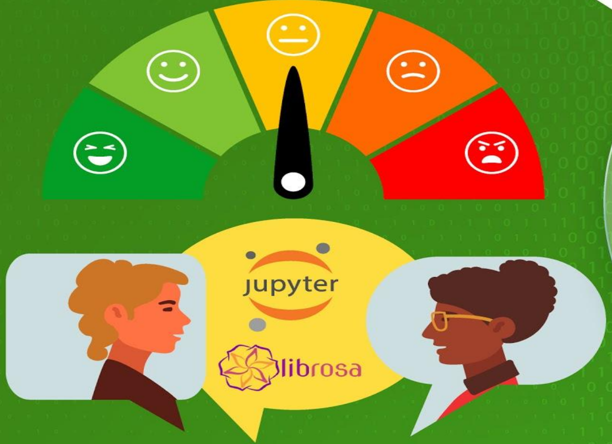
3D PRINT YOUR PHOTOS AS LITHOPHANES

NOV 16 - in5 DESIGN, D3

dxb
DUBAI DESIGN WEEK
11-16 NOVEMBER 2019



NOVEMBER 2019



NOV 9 - in5 TECH

RECOGNIZE SPEECH EMOTIONS WITH LIBROSA & JUPYTERLAB

NOV 23 - in5 TECH

MANAGE TASKS WITH RFID & GOOGLE SHEETS

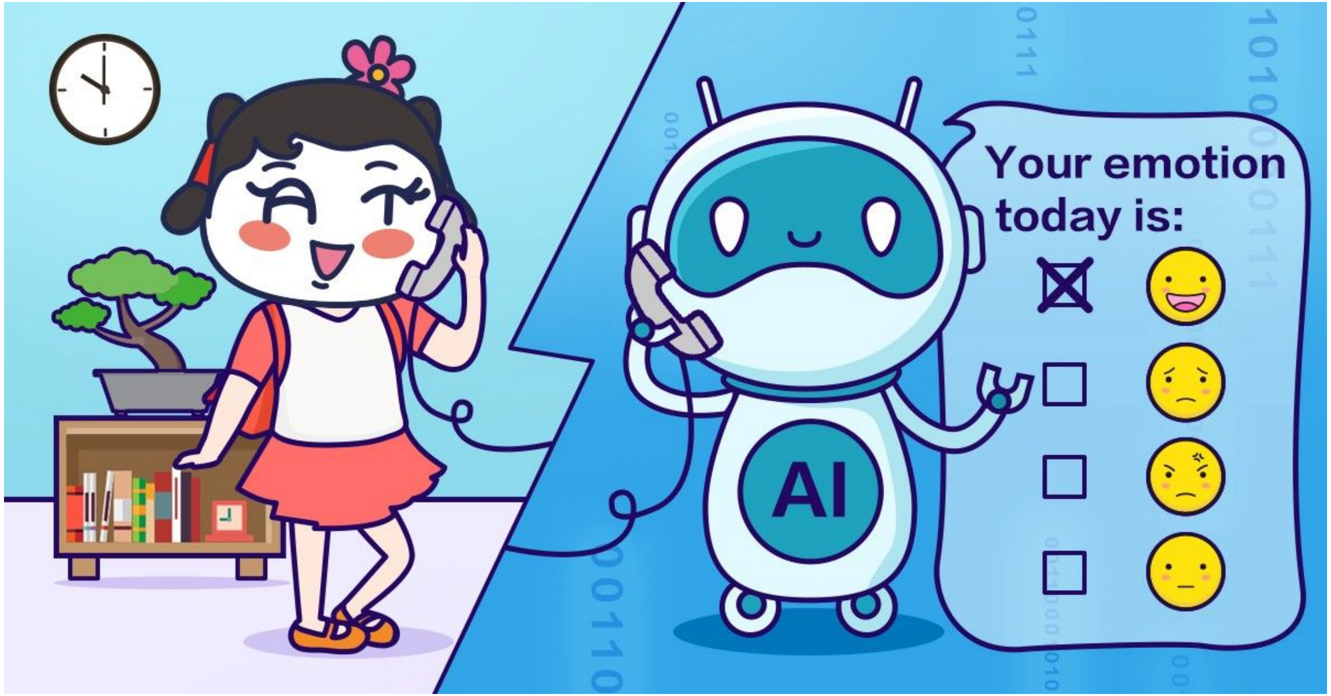
Introduction

COMMUNITY INNOVATION WORKSPACE

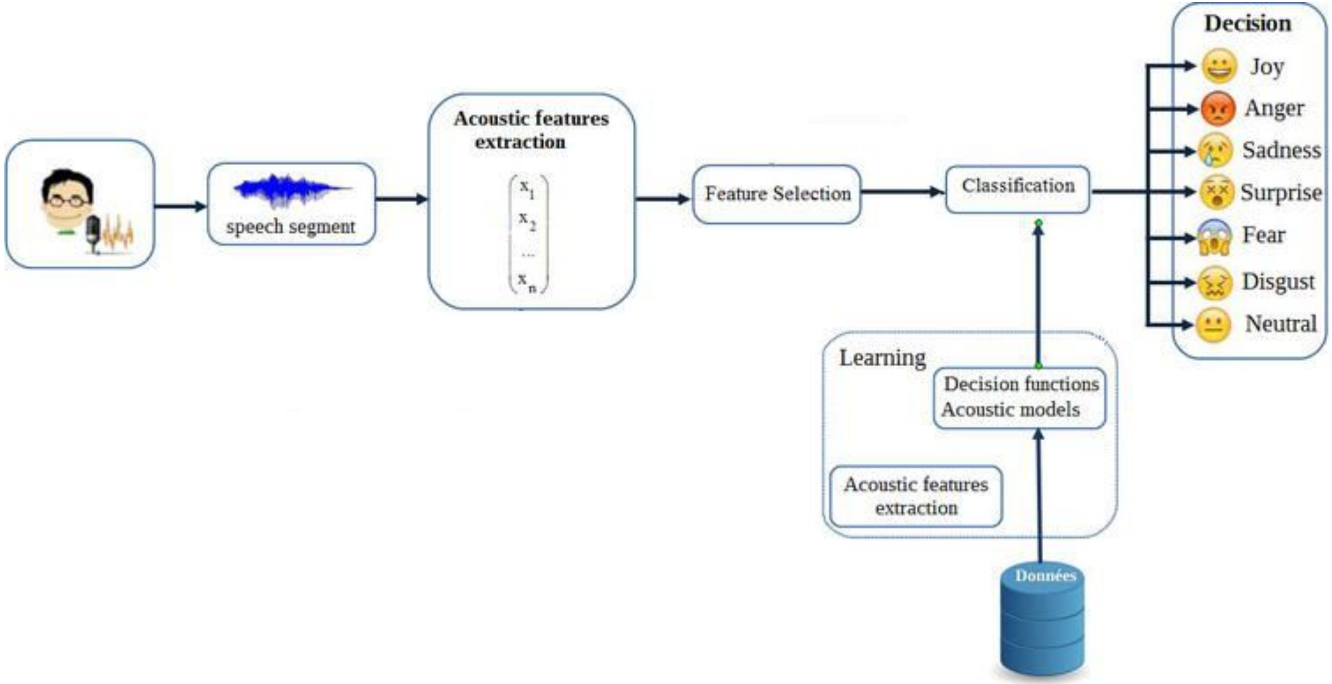
In this workshop we are going to:

- Learn about SER (Speech Emotion Recognition)
- Discuss some of the use cases for SER
- Talk about Librosa and Scikit Learn libraries
- Build and train an SER model
- Test out the model by feeding our own input

Speech Emotion Recognition (SER)



Traditional SER Algorithm



Use cases for SER

The Dataset

We’ll use the RAVDESS dataset; this is the Ryerson Audio-Visual Database of Emotional Speech and Song dataset, and is free to download. This dataset has 7356 files rated by 247 individuals 10 times on emotional validity, intensity, and genuineness.

How To Download The Dataset?

The following link will give the download link;

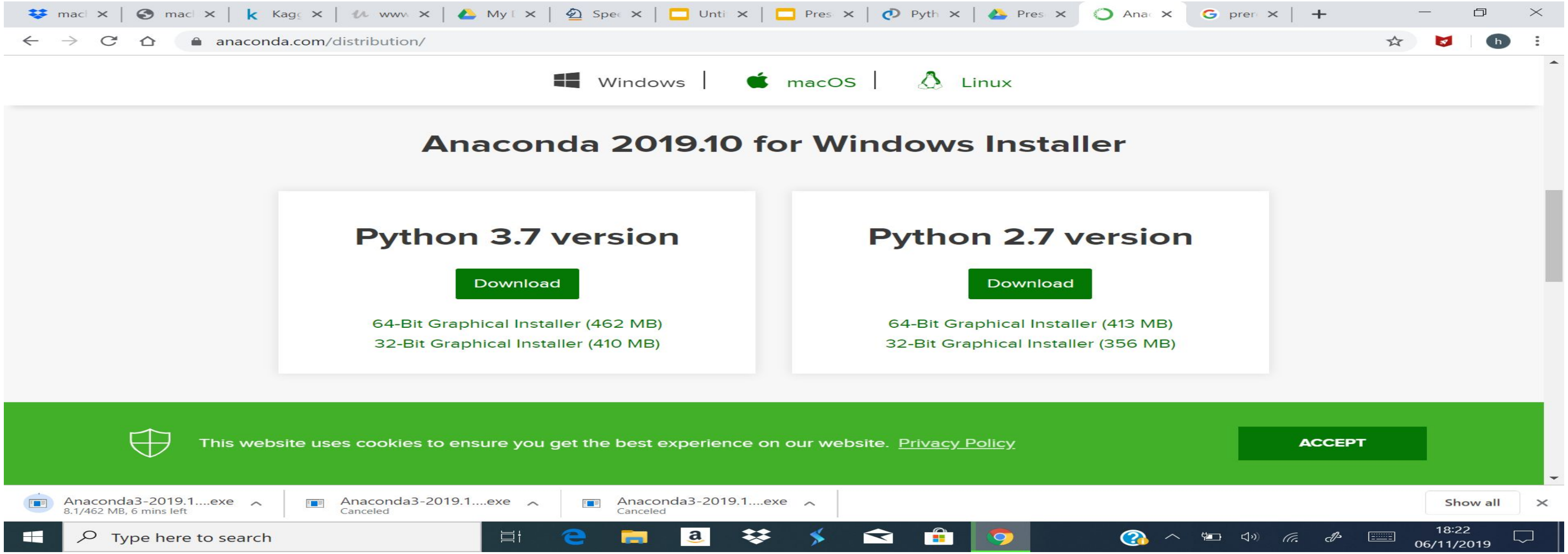
https://drive.google.com/file/d/1wWsrN2Ep7x6lWqOXfr4rpKGYrJhWc8z7/view

PREREQUISITES

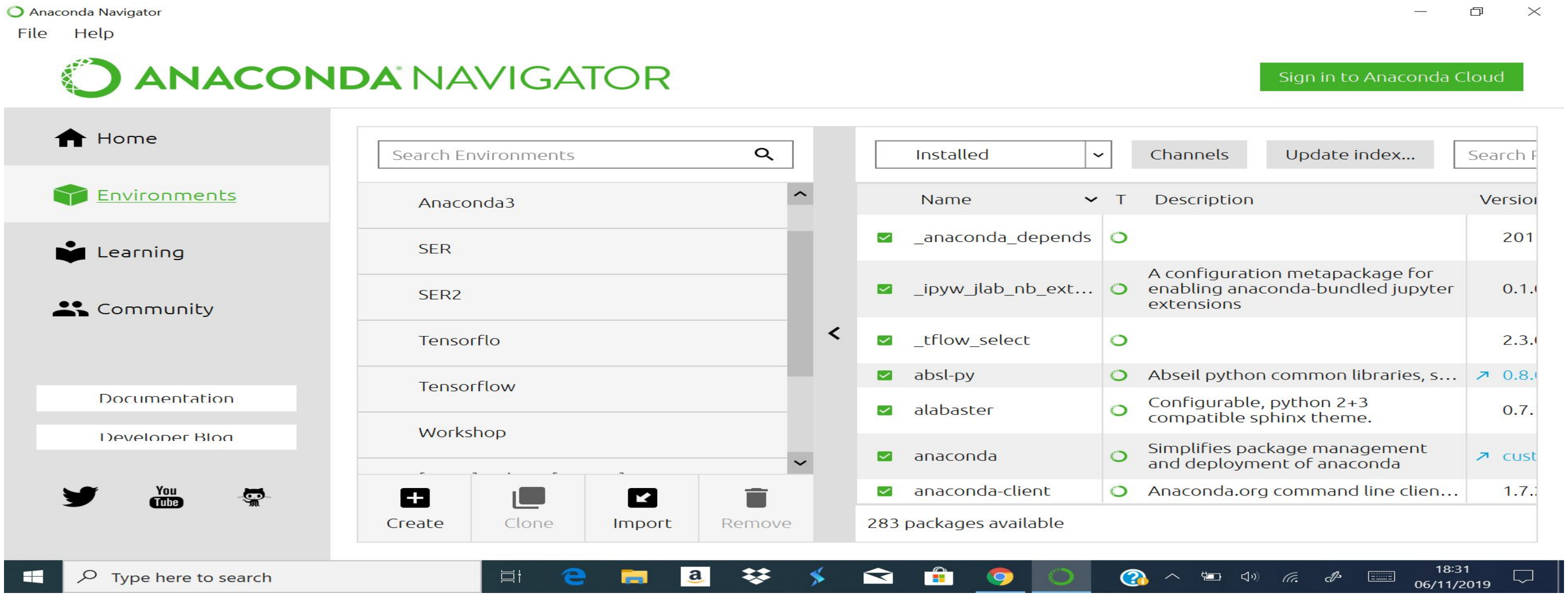
1. We need to install the Programming language Python first.
2. https://www.python.org/doc/essays/blurb/
3. Go to Downloads,Based on your Device (Mac OS or Windows)
4. Scroll Down, Download Python Version (3.6.9)

PREREQUISITES

Link; https://www.anaconda.com/distribution/

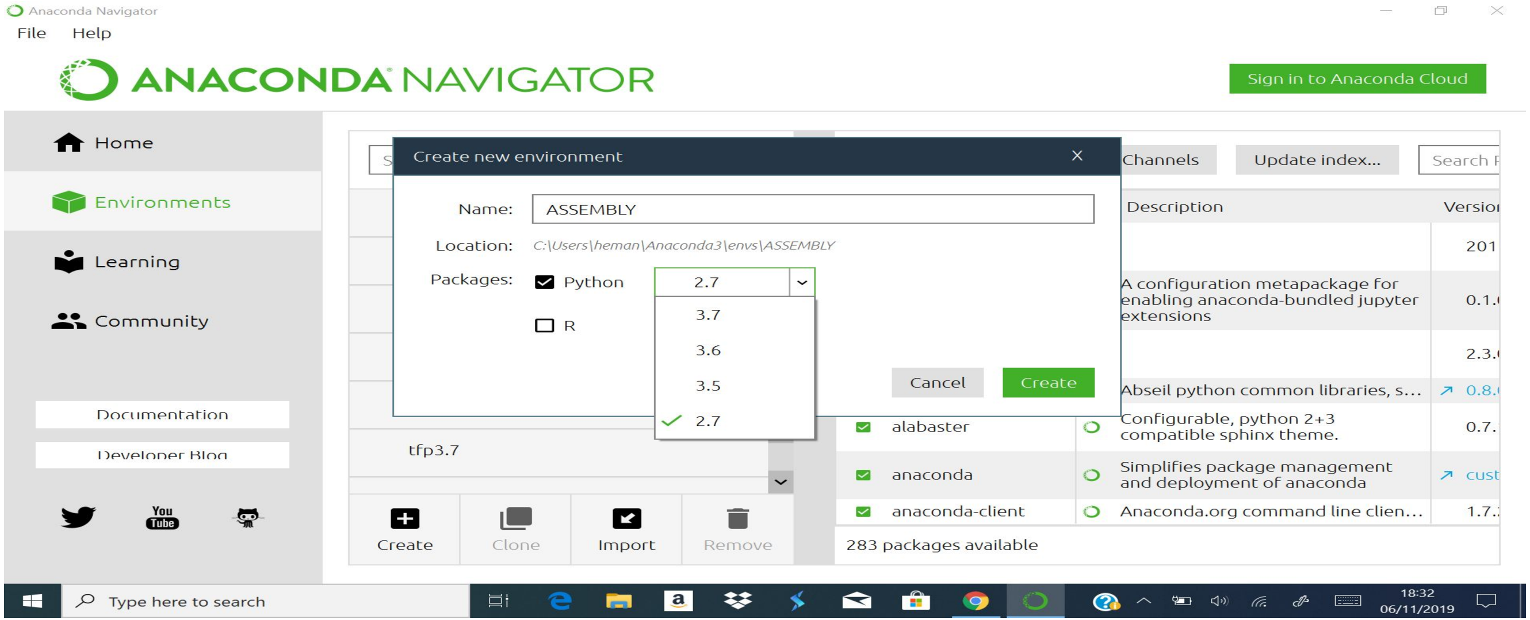


CONDA ENVIRONMENT



COMMUNITY
INNOVATION
WORKSPACE

CREATING A CONDA ENVIRONMENT



LIBRARIES

https://github.com/The-Assembly/emotion_speech_recognition/

Go to Command Prompt (CD)-

pip install -r REQUIRMENTS.txt

Librosa Audio and Music Signal Analysis

Librosa is a *Python library* for analyzing audio and music. It has a flatter package layout, standardized interfaces and names, backwards compatibility, modular functions, and readable code.

SCIKIT-LEARN

Scikit-learn (formerly **scikits.learn** and also known as **sklearn**) is a free software machine learning library for the Python programming language.It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

COMMUNITY
INNOVATION
WORKSPACE

PYAUDIO

PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms, such as GNU/Linux, Microsoft Windows, and Apple Mac OS X / macOS.

What is Jupyterlab?

JupyterLab is an open-source, web-based UI for Project Jupyter and it has all basic functionalities of the Jupyter Notebook, like notebooks, terminals, text editors, file browsers, rich outputs, and more. However, it also provides improved support for third party extensions.

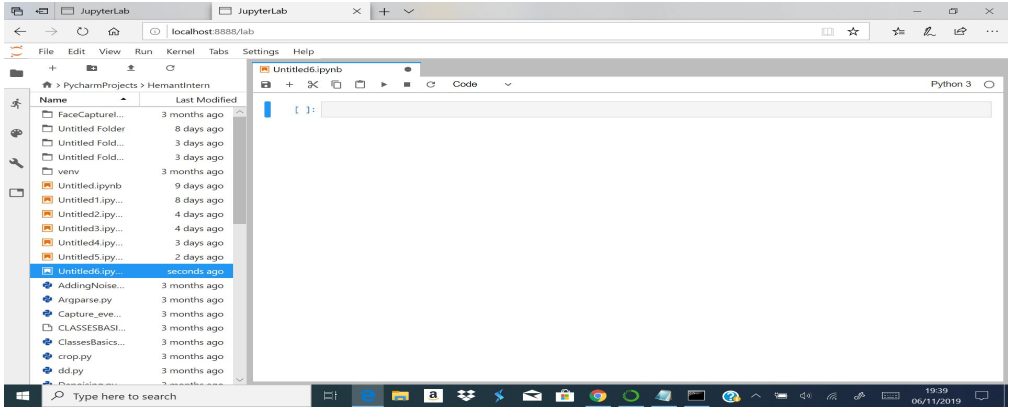
JUPYTER LAB

Can use it to explain their reasoning, show their work, and draw connections between their classwork and the world outside. Scientists, journalists, and researchers can use it to open up their data, share the stories behind their computations, and enable future collaboration and innovation.

JUPYTER NOTEBOOK

Go to Command Prompt (CD)- Jupyter Lab

*Once this command is written,Jupyter Lab will open in a new Internet Tab



CODE STRUCTURE

1) Import the required libraries

CODE STRUCTURE

2) Extracting the features from the librosa library;We have to extract the mfcc,chroma and mel features from the sound file.This function takes 4 Parameters – the file name and three Boolean parameters for the three features.

- mfcc: Mel Frequency Cepstral Coefficient, represents the short-term power spectrum of a sound.
- chroma: Pertains to the 12 different pitch classes.
- mel: Mel Spectrogram Frequency.

CODE STRUCTURE

- Extraction of the features from each sound file happens.
- After Extraction we store in np.hstack(Horizontal)
- np.mean is used to get the arithmetic mean of the features which is added to the hstack

CODE STRUCTURE

Screenshot:

```
[2]: #DataFlair - Extract features (mfcc, chroma, mel) from a sound file
     def extract_feature(file_name, mfcc, chroma, mel):
         with soundfile.SoundFile(file_name) as sound_file:
             X = sound_file.read(dtype="float32")
             sample_rate=sound_file.samplerate
             if chroma:
                 stft=np.abs(librosa.stft(X))
             result=np.array([])
             if mfcc:
                 mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
                 result=np.hstack((result, mfccs))
             if chroma:
                 chroma=np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
                 result=np.hstack((result, chroma))
             if mel:
                 mel=np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
                 result=np.hstack((result, mel))
         return result
```

CODE STRUCTURE

Let's define a *dictionary* to hold numbers and the emotions available in the RAVDESS dataset, and a list to hold those we want- calm, happy, fearful, disgust.

Screenshot:

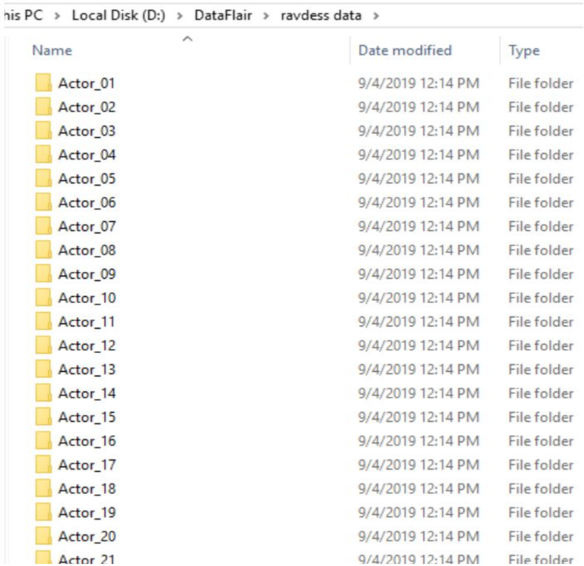
```
[3]: #DataFlair - Emotions in the RAVDESS dataset
     emotions={
          '01':'neutral',
          '02':'calm',
          '03':'happy',
          '04':'sad',
          '05':'angry',
          '06':'fearful',
          '07':'disgust',
          '08':'surprised'
     }

     #DataFlair - Emotions to observe
     observed_emotions=['calm', 'happy', 'fearful', 'disgust']
```

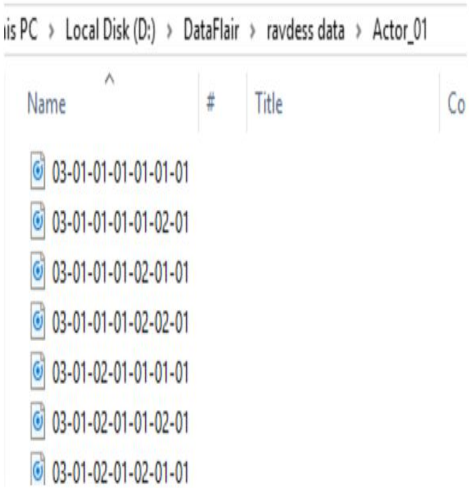
CODE STRUCTURE

Let's load the data with a function load_data() – this takes in the relative size of the test set as parameter. x and y are empty lists; we'll use the glob() function from the glob module to get all the pathnames for the sound files in our dataset. The pattern we use for this is: "directory location\\Actor_**.wav". This is because our dataset looks like this:

RAVDESS Dataset Folder


his PC › Local Disk (D:) › DataFlair › ravdess data ›

Name	Date modified	Type
Actor_01	9/4/2019 12:14 PM	File folder
Actor_02	9/4/2019 12:14 PM	File folder
Actor_03	9/4/2019 12:14 PM	File folder
Actor_04	9/4/2019 12:14 PM	File folder
Actor_05	9/4/2019 12:14 PM	File folder
Actor_06	9/4/2019 12:14 PM	File folder
Actor_07	9/4/2019 12:14 PM	File folder
Actor_08	9/4/2019 12:14 PM	File folder
Actor_09	9/4/2019 12:14 PM	File folder
Actor_10	9/4/2019 12:14 PM	File folder
Actor_11	9/4/2019 12:14 PM	File folder
Actor_12	9/4/2019 12:14 PM	File folder
Actor_13	9/4/2019 12:14 PM	File folder
Actor_14	9/4/2019 12:14 PM	File folder
Actor_15	9/4/2019 12:14 PM	File folder
Actor_16	9/4/2019 12:14 PM	File folder
Actor_17	9/4/2019 12:14 PM	File folder
Actor_18	9/4/2019 12:14 PM	File folder
Actor_19	9/4/2019 12:14 PM	File folder
Actor_20	9/4/2019 12:14 PM	File folder
Actor_21	9/4/2019 12:14 PM	File folder


is PC › Local Disk (D:) › DataFlair › ravdess data › Actor_01

Name	#	Title	Co
03-01-01-01-01-01-01			
03-01-01-01-01-02-01			
03-01-01-01-02-01-01			
03-01-01-01-02-02-01			
03-01-02-01-01-01-01			
03-01-02-01-01-02-01			
03-01-02-01-02-01-01			

CODE STRUCTURE

Using our emotions dictionary, this number is turned into an emotion, and our function checks whether this emotion is in our list of observed_emotions; if not, it continues to the next file. It makes a call to extract_feature and stores what is returned in 'feature'. Then, it appends the feature to x and the emotion to y. So, the list x holds the features and y holds the emotions. We call the function train_test_split with these, the test size, and a random state value, and return that.

```
def load_data(test_size=0.2):
    x,y=[],[]
    for file in glob.glob("D:\\DataFlair\\ravdess data\\Actor_*\\*.wav"):
        file_name=os.path.basename(file)
        emotion=emotions[file_name.split("-")[2]]
        if emotion not in observed_emotions:
            continue
        feature=extract_feature(file, mfcc=True, chroma=True, mel=True)
        x.append(feature)
        y.append(emotion)
    return train_test_split(np.array(x), y, test_size=test_size, random_state=9)
```

CODE STRUCTURE

Time to split the dataset into training and testing sets,keep the test set 25% of everything and use the load_data function for this.

```
#DataFlair - Split the dataset
x_train,x_test,y_train,y_test=load_data(test_size=0.25)
```

CODE STRUCTURE

Observe the shape of the training and testing datasets;

```
#DataFlair - Get the shape of the training and testing
datasets
print((x_train.shape[0], x_test.shape[0]))
```

CODE STRUCTURE

Get the number of features extracted;

Output Screenshot:

```
[7]: #DataFlair - Get the number of features extracted
     print(f'Features extracted: {x_train.shape[1]}')

     Features extracted: 180
```

COMMUNITY
INNOVATION
WORKSPACE

CODE STRUCTURE

- Initialize an MLPClassifier. This is a Multi-layer Perceptron Classifier.
- The MLPClassifier has an internal neural network for the purpose of classification. This is a feedforward ANN model.

```
#DataFlair - Initialize the Multi Layer Perceptron Classifier
model=MLPClassifier(alpha=0.01, batch_size=256, epsilon=1e-08, hidden_layer_sizes=(300,), learning_rate='adaptive', max_iter=500)
```

CODE STRUCTURE

Let's predict the values for the test set. This gives us y_pred (the predicted emotions for the features in the test set).

Screenshot:

```
[10]: #DataFlair - Predict for the test set
      y_pred=model.predict(x_test)
```

CODE STRUCTURE

To calculate the accuracy of our model, we'll call up the accuracy_score() function we imported from sklearn. Finally, we'll round the accuracy to 2 decimal places and print it out.

Output Screenshot:

```
[11]: #DataFlair - Calculate the accuracy of our model
      accuracy=accuracy_score(y_true=y_test, y_pred=y_pred)

      #DataFlair - Print the accuracy
      print("Accuracy: {:.2f}%".format(accuracy*100))

      Accuracy: 72.40%
```

CODE STRUCTURE

Create a directory called result and save the trained model as “ .model” file.

```
# now we save the model
# make result directory if doesn't exist yet
if not os.path.isdir("result"):
    os.mkdir("result")

pickle.dump(model, open("result/mlp_classifier.model", "wb"))
```

Installation: Pyaudio

Installing pyaudio from the github repository, open the file:

- PyAudio-0.2.11-cp37-cp37m-win_amd64.whl
- pip install PyAudio-0.2.11-cp37-cp37m-win_amd64.whl
- Also download the utils.py script

Testing the trained model : CODE STRUCTURE

- Testing the Trained Model,Install the library Pyaudio
- Install the other required libraries

```
import pyaudio
import os
import wave
import pickle
from sys import byteorder
from array import array
from struct import pack
from sklearn.neural_network import MLPClassifier

from utils import extract_feature

```

Testing the trained model : CODE STRUCTURE

- We are creating different functions which are used to clean the

```
    # Trim to the right
    snd_data.reverse()
    snd_data = _trim(snd_data)
    snd_data.reverse()
    return snd_data

def add_silence(snd_data, seconds):
    "Add silence to the start and end of 'snd_data' of length 'seconds' (float)"
    r = array('h', [0 for i in range(int(seconds*RATE))])
    r.extend(snd_data)
    r.extend([0 for i in range(int(seconds*RATE))])
    return r

def record():
    """
    Record a word or words from the microphone and
    return the data as an array of signed shorts.

    Normalizes the audio, trims silence from the
    start and end, and pads with 0.5 seconds of
    blank sound to make sure VLC et al can play
    it without getting chopped off.
    """
    p = pyaudio.PyAudio()
    stream = p.open(format=FORMAT, channels=1, rate=RATE,
        input=True, output=True,
        frames_per_buffer=CHUNK_SIZE)

    num_silent = 0
```

Testing the trained model : CODE STRUCTURE

- Once the testing of the trained model is done we give in the desired input

```
if __name__ == "__main__":
    # load the saved model (after training)
    model = pickle.load(open("result/mlp_classifier.model", "rb"))
    print("Please talk")
    filename = "test.wav"
    # record the file (start talking)
    record_to_file(filename)
    # extract features and reshape it
    features = extract_feature(filename, mfcc=True, chroma=True, mel=True).reshape(1, -1)
    # predict
    result = model.predict(features)[0]
    # show the result !
    print("result:", result)
```

```
Please talk
result: happy
```