

```

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'ravdess-emotional-speech-audio:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F107620%2F256618%2Fbundle%2Far'

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
    os.symlink(KAGGLE_INPUT_PATH, os.path.join(".", 'input'), target_is_directory=True)
except FileExistsError:
    pass
try:
    os.symlink(KAGGLE_WORKING_PATH, os.path.join(".", 'working'), target_is_directory=True)
except FileExistsError:
    pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f'\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded')
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
                with ZipFile(tfile) as zfile:
                    zfile.extractall(destination_path)
            else:
                with tarfile.open(tfile.name) as tarfile:
                    tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')

```

```

➡ Downloading ravdess-emotional-speech-audio, 450102890 bytes compressed
[=====] 450102890 bytes downloaded
Downloaded and uncompressed: ravdess-emotional-speech-audio
Downloading toronto-emotional-speech-set-tess, 448572034 bytes compressed
[=====] 448572034 bytes downloaded
Downloaded and uncompressed: toronto-emotional-speech-set-tess
Downloading cremad, 473324524 bytes compressed
[=====] 473324524 bytes downloaded

```

```
Downloaded and uncompressed: cremad
Downloading surrey-audiovisual-expressed-emotion-savee, 112690765 bytes compressed
[=====] 112690765 bytes downloaded
Downloaded and uncompressed: surrey-audiovisual-expressed-emotion-savee
Data source import complete.
```

✓ Speech Emotion Recognition

I am going to build a speech emotion detection classifier.

But first we need to learn about what is speech recognition (SER) and why are we building this project? Well, few of the reasons are-

First, let's define SER i.e. Speech Emotion Recognition.

- Speech Emotion Recognition, abbreviated as SER, is the act of attempting to recognize human emotion and affective states from speech. This is capitalizing on the fact that voice often reflects underlying emotion through tone and pitch. This is also the phenomenon that animals like dogs and horses employ to be able to understand human emotion.

Why we need it?

1. Emotion recognition is the part of speech recognition which is gaining more popularity and need for it increases enormously. Although there are methods to recognize emotion using machine learning techniques, this project attempts to use deep learning to recognize the emotions from data.
2. SER(Speech Emotion Recognition) is used in call center for classifying calls according to emotions and can be used as the performance parameter for conversational analysis thus identifying the unsatisfied customer, customer satisfaction and so on.. for helping companies improving their services
3. It can also be used in-car board system based on information of the mental state of the driver can be provided to the system to initiate his/her safety preventing accidents to happen

Datasets used in this project

- Crowd-sourced Emotional Multimodal Actors Dataset (Crema-D)
- Ryerson Audio-Visual Database of Emotional Speech and Song (Ravdess)
- Surrey Audio-Visual Expressed Emotion (Savee)
- Toronto emotional speech set (Tess)

✓ Importing Libraries

```

import pandas as pd
import numpy as np

import os
import sys

# librosa is a Python library for analyzing audio and music. It can be used to extract the data from the audio files we will see it later.
import librosa
import librosa.display
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# to play the audio files
from IPython.display import Audio

import keras
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint

import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning)

```

✓ Data Preparation

- As we are working with four different datasets, so i will be creating a dataframe storing all emotions of the data in dataframe with their paths.
- We will use this dataframe to extract features for our model training.

```

# Paths for data.
Ravdess = "/kaggle/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
Crema = "/kaggle/input/cremad/AudioWAV/"
Tess = "/kaggle/input/toronto-emotional-speech-set-tess/tess toronto emotional speech set data/TESS Toronto emotional speech set data/"
Savee = "/kaggle/input/surrey-audiovisual-expressed-emotion-savee/ALL/"

```

✓ 1. Ravdess Dataframe

Here is the filename identifiers as per the official RAVDESS website:

- Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
- Vocal channel (01 = speech, 02 = song).
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
- Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- Repetition (01 = 1st repetition, 02 = 2nd repetition).
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

So, here's an example of an audio filename. 02-01-06-01-02-01-12.mp4 This means the meta data for the audio file is:

- Video-only (02)
- Speech (01)
- Fearful (06)
- Normal intensity (01)
- Statement "dogs" (02)
- 1st Repetition (01)
- 12th Actor (12) - Female (as the actor ID number is even)

```

ravdess_directory_list = os.listdir(Ravdess)


file_emotion = []
file_path = []
for dir in ravdess_directory_list:
    # as their are 20 different actors in our previous directory we need to extract files for each actor.
    actor = os.listdir(Ravdess + dir)
    for file in actor:
        part = file.split('.')[0]
        part = part.split('-')
        # third part in each file represents the emotion associated to that file.
        file_emotion.append(int(part[2]))
        file_path.append(Ravdess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Ravdess_df = pd.concat([emotion_df, path_df], axis=1)

# changing integers to actual emotions.
Ravdess_df.Emotions.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5:'angry', 6:'fear', 7:'disgust', 8:'surprise'}, inplace=True)
Ravdess_df.head()

```



	Emotions	Path
0	angry	/kaggle/input/ravdess-emotional-speech-audio/a...
1	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
2	sad	/kaggle/input/ravdess-emotional-speech-audio/a...
3	happy	/kaggle/input/ravdess-emotional-speech-audio/a...
4	sad	/kaggle/input/ravdess-emotional-speech-audio/a...

2. Crema DataFrame

```

crema_directory_list = os.listdir(Crema)


file_emotion = []
file_path = []

for file in crema_directory_list:
    # storing file paths
    file_path.append(Crema + file)
    # storing file emotions
    part=file.split('_')
    if part[2] == 'SAD':
        file_emotion.append('sad')
    elif part[2] == 'ANG':
        file_emotion.append('angry')
    elif part[2] == 'DIS':
        file_emotion.append('disgust')
    elif part[2] == 'FEA':
        file_emotion.append('fear')
    elif part[2] == 'HAP':
        file_emotion.append('happy')
    elif part[2] == 'NEU':
        file_emotion.append('neutral')
    else:
        file_emotion.append('Unknown')

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Crema_df = pd.concat([emotion_df, path_df], axis=1)
Crema_df.head()

```



	Emotions	Path
0	angry	/kaggle/input/cremad/AudioWAV/1014_IEO_ANG_LO.wav
1	angry	/kaggle/input/cremad/AudioWAV/1044_IEO_ANG_MD.wav
2	happy	/kaggle/input/cremad/AudioWAV/1011_TSI_HAP_XX.wav
3	fear	/kaggle/input/cremad/AudioWAV/1059_IOM_FEA_XX.wav
4	angry	/kaggle/input/cremad/AudioWAV/1051_ITS_ANG_XX.wav

3. TESS dataset


```
tess_directory_list = os.listdir(Tess)

file_emotion = []
file_path = []

for dir in tess_directory_list:
    directories = os.listdir(Tess + dir)
    for file in directories:
        part = file.split('.')[0]
        part = part.split('_')[2]
        if part=='ps':
            file_emotion.append('surprise')
        else:
            file_emotion.append(part)
        file_path.append(Tess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Tess_df = pd.concat([emotion_df, path_df], axis=1)
Tess_df.head()
```



	Emotions	Path
0	angry	/kaggle/input/toronto-emotional-speech-set-tes...
1	angry	/kaggle/input/toronto-emotional-speech-set-tes...
2	angry	/kaggle/input/toronto-emotional-speech-set-tes...
3	angry	/kaggle/input/toronto-emotional-speech-set-tes...
4	angry	/kaggle/input/toronto-emotional-speech-set-tes...

4. CREMA-D dataset

The audio files in this dataset are named in such a way that the prefix letters describes the emotion classes as follows:

- 'a' = 'anger'
- 'd' = 'disgust'
- 'f' = 'fear'
- 'h' = 'happiness'
- 'n' = 'neutral'
- 'sa' = 'sadness'
- 'su' = 'surprise'

```

savee_directory_list = os.listdir(Savee)


file_emotion = []
file_path = []

for file in savee_directory_list:
    file_path.append(Savee + file)
    part = file.split('_')[1]
    ele = part[:-6]
    if ele=='a':
        file_emotion.append('angry')
    elif ele=='d':
        file_emotion.append('disgust')
    elif ele=='f':
        file_emotion.append('fear')
    elif ele=='h':
        file_emotion.append('happy')
    elif ele=='n':
        file_emotion.append('neutral')
    elif ele=='sa':
        file_emotion.append('sad')
    else:
        file_emotion.append('surprise')

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Savee_df = pd.concat([emotion_df, path_df], axis=1)
Savee_df.head()

```




	Emotions	Path
0	surprise	/kaggle/input/surrey-audiovisual-expressed-emo...
1	happy	/kaggle/input/surrey-audiovisual-expressed-emo...
2	neutral	/kaggle/input/surrey-audiovisual-expressed-emo...
3	disgust	/kaggle/input/surrey-audiovisual-expressed-emo...
4	neutral	/kaggle/input/surrey-audiovisual-expressed-emo...

```

# creating Dataframe using all the 4 dataframes we created so far.
data_path = pd.concat([Ravdess_df, Crema_df, Tess_df, Savee_df], axis = 0)
data_path.to_csv("data_path.csv", index=False)
data_path.head()

```



	Emotions	Path
0	angry	/kaggle/input/ravdess-emotional-speech-audio/a...
1	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
2	sad	/kaggle/input/ravdess-emotional-speech-audio/a...
3	happy	/kaggle/input/ravdess-emotional-speech-audio/a...
4	sad	/kaggle/input/ravdess-emotional-speech-audio/a...

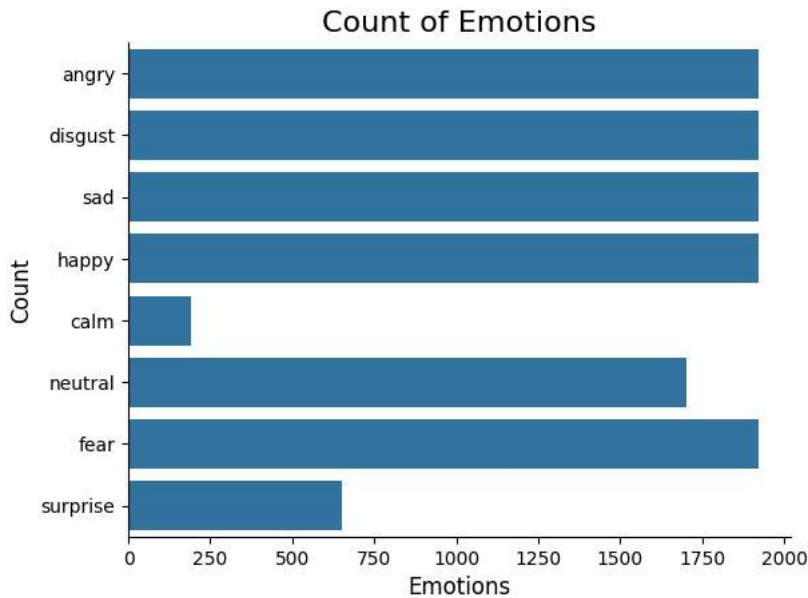
▼ Data Visualisation and Exploration

First let's plot the count of each emotions in our dataset.

```

plt.title('Count of Emotions', size=16)
sns.countplot(data_path.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()

```



We can also plot waveplots and spectrograms for audio signals

- Waveplots - Waveplots let us know the loudness of the audio at a given time.
- Spectrograms - A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It's a representation of frequencies changing with respect to time for given audio/music signals.

```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveshow(data, sr=sr)

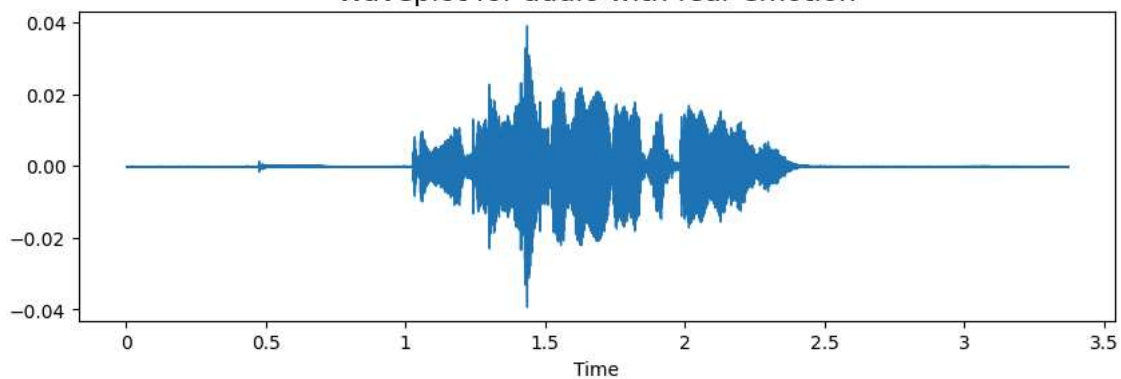
    plt.show()

def create_spectrogram(data, sr, e):
    # stft function converts the data into short term fourier transform
    X = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(12, 3))
    plt.title('Spectrogram for audio with {} emotion'.format(e), size=15)
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
    #librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar()

emotion='fear'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

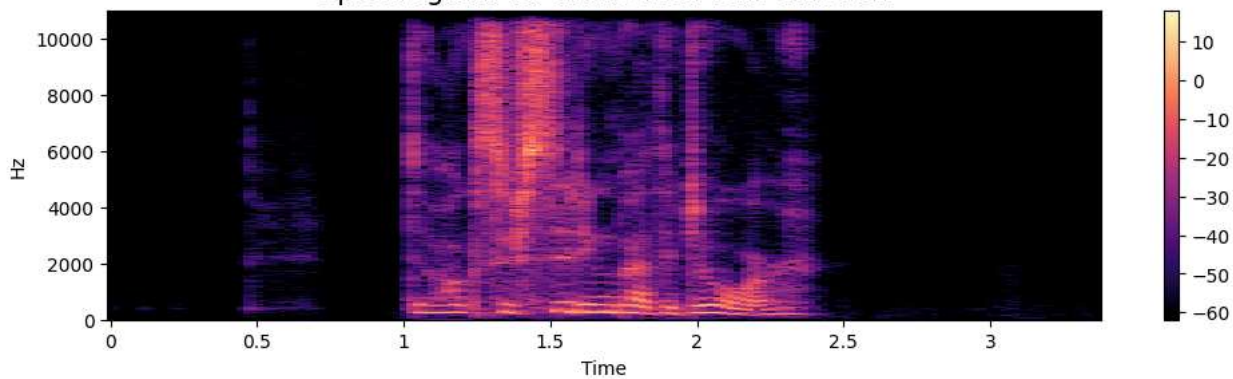


Waveplot for audio with fear emotion



0:00 / 0:03

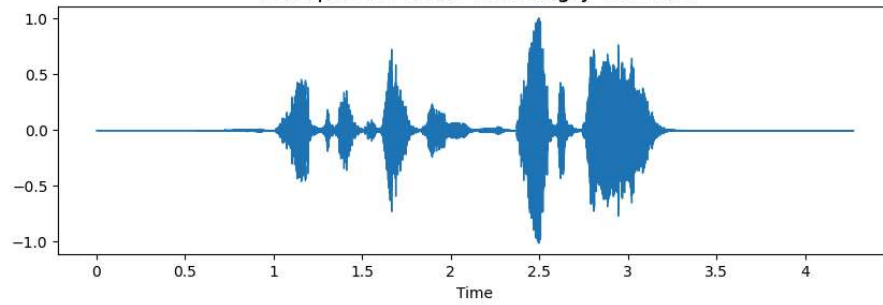
Spectrogram for audio with fear emotion



```
emotion='angry'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

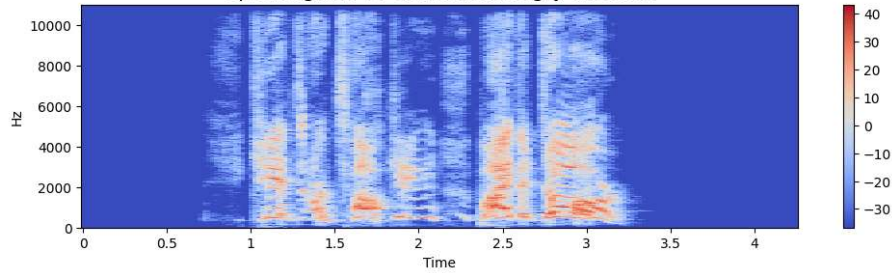



Waveplot for audio with angry emotion

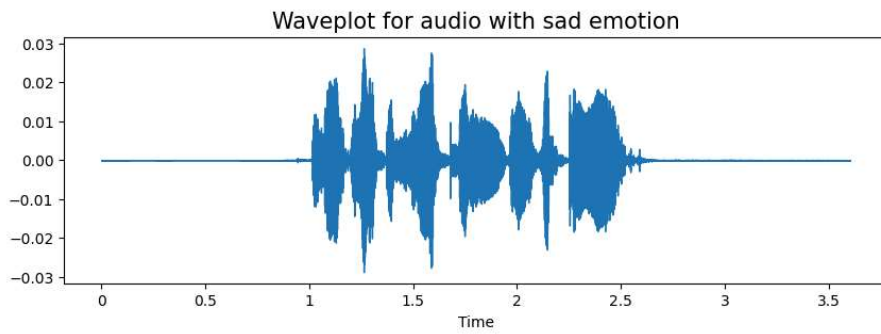


0:00 / 0:04

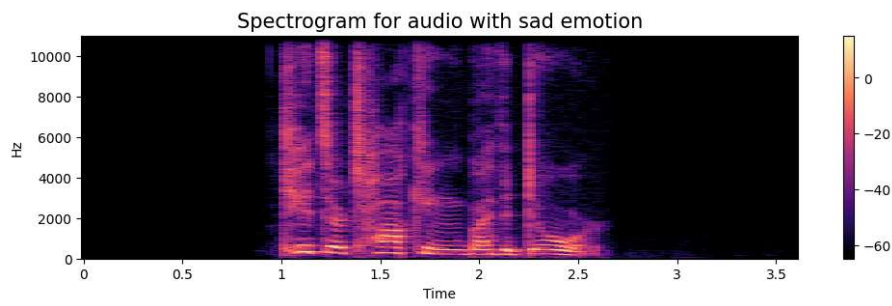
Spectrogram for audio with angry emotion



```
emotion='sad'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



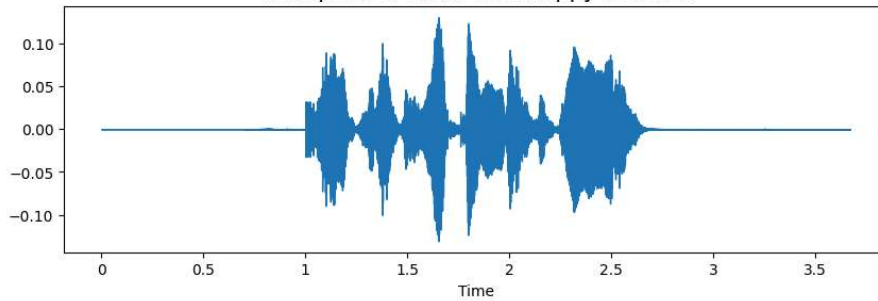
0:00 / 0:03



```
emotion='happy'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

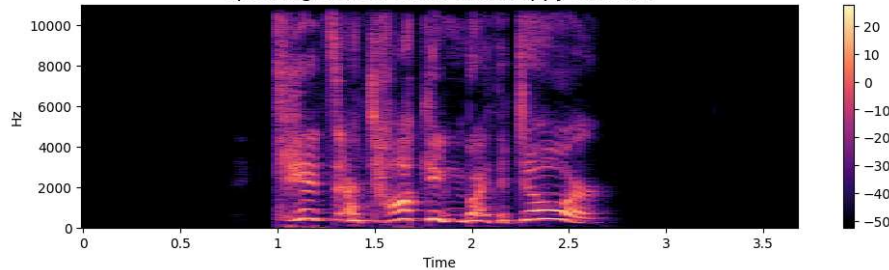


Waveplot for audio with happy emotion



0:00 / 0:03

Spectrogram for audio with happy emotion



✓ Data Augmentation

- Data augmentation is the process by which we create new synthetic data samples by adding small perturbations on our initial training set.
- To generate syntactic data for audio, we can apply noise injection, shifting time, changing pitch and speed.
- The objective is to make our model invariant to those perturbations and enhance its ability to generalize.
- In order to this to work adding the perturbations must conserve the same label as the original training sample.
- In images data augmentation can be performed by shifting the image, zooming, rotating ...

First, let's check which augmentation techniques works better for our dataset.

```
def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate)

def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sampling_rate, pitch_factor)

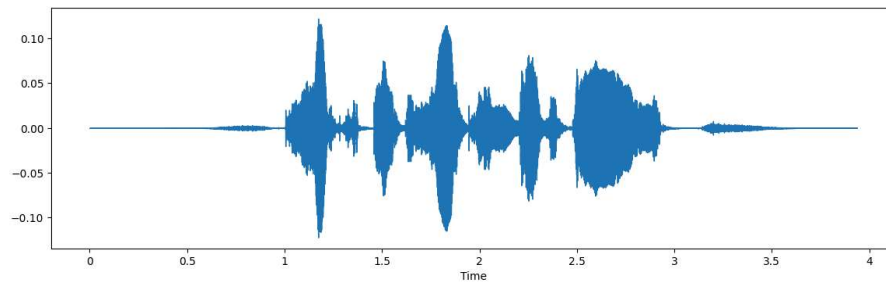
# taking any example and checking for techniques.
path = np.array(data_path.Path)[1]
data, sample_rate = librosa.load(path)
```

✓ 1. Simple Audio

```
plt.figure(figsize=(14,4))  
librosa.display.waveshow(y=data, sr=sample_rate)  
Audio(path)
```



0:00 / 0:03

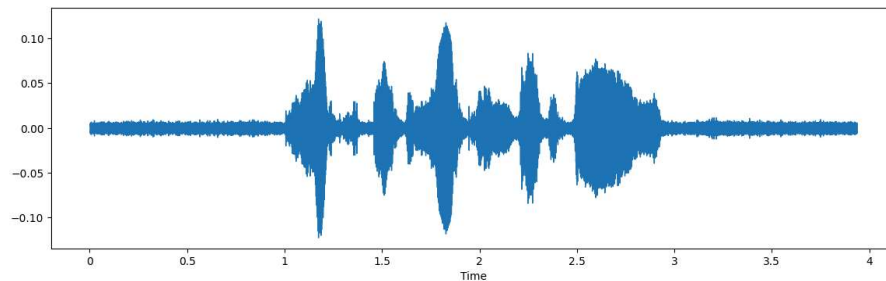


✓ 2. Noise Injection

```
x = noise(data)  
plt.figure(figsize=(14,4))  
librosa.display.waveshow(y=x, sr=sample_rate)  
Audio(x, rate=sample_rate)
```



0:00 / 0:03



We can see noise injection is a very good augmentation technique because of which we can assure our training model is not overfitted

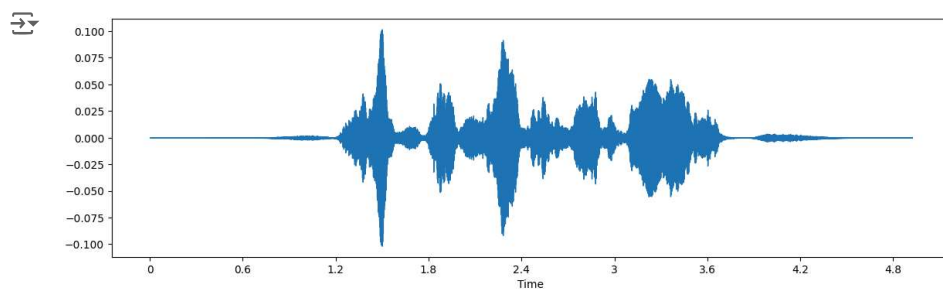
✓ 3. Stretching

```
import librosa
import matplotlib.pyplot as plt
from IPython.display import Audio

def stretch(data, rate=0.8):
    # Apply time stretching
    return librosa.effects.time_stretch(data, rate=rate)

# Example usage
x = stretch(data, rate=0.8)
plt.figure(figsize=(14, 4))
librosa.display.waveshow(y=x, sr=sample_rate)
plt.show()

# Play the stretched audio
Audio(x, rate=sample_rate)
```

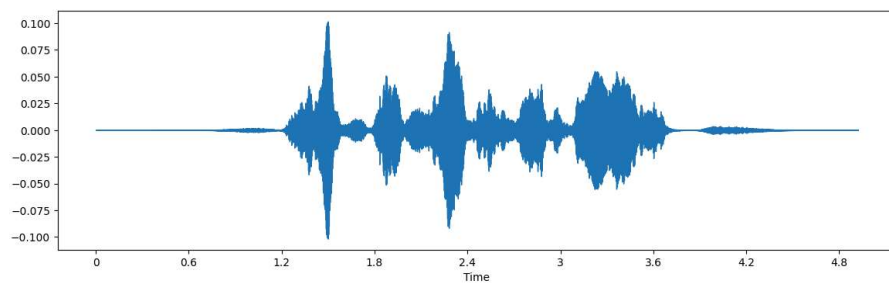


0:00 / 0:04

```
x = stretch(data)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```



0:00 / 0:04



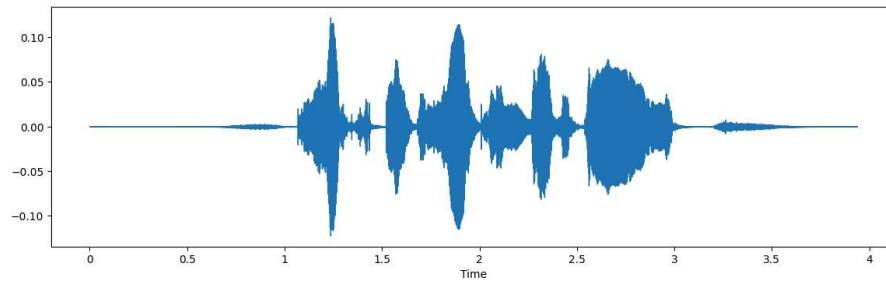
4. Shifting

```
x = shift(data)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
```

```
librosa.display.waveshow(y, sr=sample_rate,
Audio(x, rate=sample_rate)
```



0:00 / 0:03



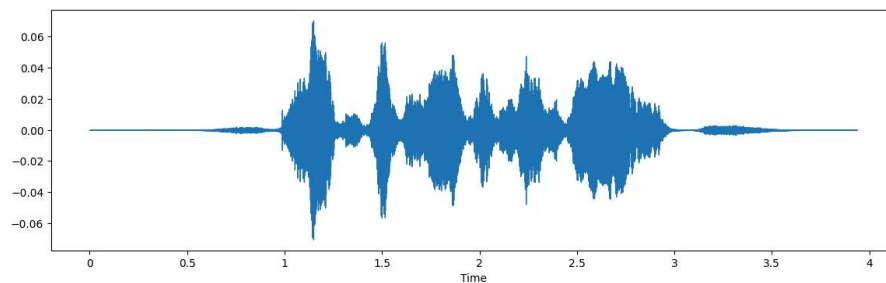
5. Pitch

```
import librosa
import matplotlib.pyplot as plt
from IPython.display import Audio

def pitch(data, sampling_rate, pitch_factor=0.7):
    # Apply pitch shift
    return librosa.effects.pitch_shift(data, sr=sampling_rate, n_steps=pitch_factor)

# Example usage
x = pitch(data, sample_rate)
plt.figure(figsize=(14, 4))
librosa.display.waveshow(y=x, sr=sample_rate)
plt.show()

# Play the pitch-shifted audio
Audio(x, rate=sample_rate)
```



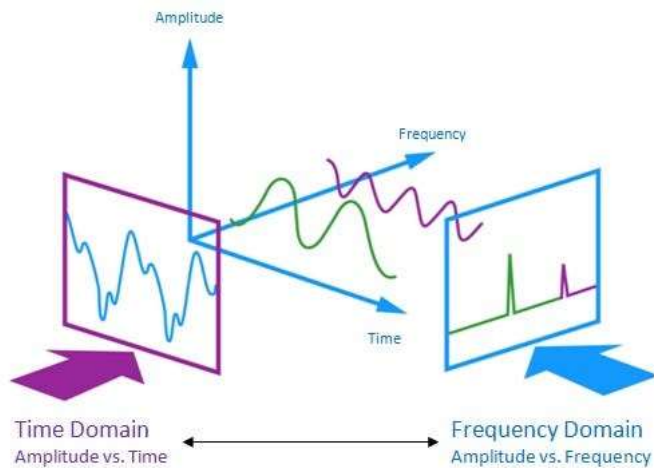
0:00 / 0:03

- From the above types of augmentation techniques i am using noise, stretching(ie. changing speed) and some pitching.

✓ Feature Extraction

- Extraction of features is a very important part in analyzing and finding relations between different things. As we already know that the data provided of audio cannot be understood by the models directly so we need to convert them into an understandable format for which feature extraction is used.

The audio signal is a three-dimensional signal in which three axes represent time, amplitude and frequency.



As stated there with the help of the sample rate and the sample data, one can perform several transformations on it to extract valuable features out of it.

1. Zero Crossing Rate : The rate of sign-changes of the signal during the duration of a particular frame.
2. Energy : The sum of squares of the signal values, normalized by the respective frame length.
3. Entropy of Energy : The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
4. Spectral Centroid : The center of gravity of the spectrum.
5. Spectral Spread : The second central moment of the spectrum.
6. Spectral Entropy : Entropy of the normalized spectral energies for a set of sub-frames.
7. Spectral Flux : The squared difference between the normalized magnitudes of the spectra of the two successive frames.
8. Spectral Rolloff : The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
9. MFCCs Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
10. Chroma Vector : A 12-element representation of the spectral energy where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).
11. Chroma Deviation : The standard deviation of the 12 chroma coefficients.

In this project i am not going deep in feature selection process to check which features are good for our dataset rather i am only extracting 5 features:

- Zero Crossing Rate
- Chroma_stft
- MFCC
- RMS(root mean square) value
- MelSpectrogram to train our model.

```

def extract_features(data):
    # ZCR
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result=np.hstack((result, zcr)) # stacking horizontally

    # Chroma_stft
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft)) # stacking horizontally

    # MFCC
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mfcc)) # stacking horizontally

    # Root Mean Square Value
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms)) # stacking horizontally

    # MelSpectrogram
    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel)) # stacking horizontally

    return result

def get_features(path):
    # duration and offset are used to take care of the no audio in start and the ending of each audio files as seen above.
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)

    # without augmentation
    res1 = extract_features(data)
    result = np.array(res1)

    # data with noise
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2)) # stacking vertically

    # data with stretching and pitching
    new_data = stretch(data)
    data_stretch_pitch = pitch(new_data, sample_rate)
    res3 = extract_features(data_stretch_pitch)
    result = np.vstack((result, res3)) # stacking vertically

    return result

X, Y = [], []

for path, emotion in zip(data_path.Path, data_path.Emotions):
    # Extract the original feature
    original_feature = get_features(path)

    # Apply augmentation techniques (e.g., stretching, pitch shifting)
    stretch_feature = stretch(original_feature)
    pitch_feature = pitch(original_feature, sample_rate)

    # Append features to X
    X.extend([original_feature, stretch_feature, pitch_feature])

    # Append the corresponding emotion 3 times
    Y.extend([emotion] * 3)

```



```

/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:266: UserWarning: n_fft
warnings.warn(
/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:266: UserWarning: n_fft
warnings.warn(

```

```

-----
KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-24-e61ba7ef4b7f> in <cell line: 3>()
      3 for path, emotion in zip(data_path.Path, data_path.Emotions):
      4     # Extract the original feature
----> 5     original_feature = get_features(path)
      6
      7     # Apply augmentation techniques (e.g., stretching, pitch shifting)

```

```

----- 7 frames -----
/usr/local/lib/python3.10/dist-packages/numpy/fft/_pocketfft.py in _raw_fft(a, n, axis,
is_real, is_forward, inv_norm)
      71     else:
      72         a = swapaxes(a, axis, -1)
----> 73         r = pfi.execute(a, is_real, is_forward, fct)
      74         r = swapaxes(r, axis, -1)
      75     return r

```

```

KeyboardInterrupt:

```

```

X, Y = [], []
for path, emotion in zip(data_path.Path, data_path.Emotions):
    feature = get_features(path)
    for ele in feature:
        X.append(ele)
        # appending emotion 3 times as we have made 3 augmentation techniques on each audio file.
        Y.append(emotion)

```

```

/usr/local/lib/python3.10/dist-packages/librosa/core/pitch.py:101: UserWarning: Trying to estimate tuning from empty frequency set.
return pitch_tuning(

```

```
len(X), len(Y), data_path.Path.shape
```

```
(36486, 36486, (12162,))
```

```

Features = pd.DataFrame(X)
Features['labels'] = Y
Features.to_csv('features.csv', index=False)
Features.head()

```

```

0      0.269115   0.661842   0.638688   0.616759   0.661024   0.605353   0.567999   0.534179   0.603144
1      0.332004   0.723535   0.709408   0.723248   0.726153   0.677915   0.584769   0.617282   0.678248
2      0.234928   0.633777   0.658256   0.631592   0.651627   0.573642   0.565427   0.547067   0.592325
3      0.186325   0.512203   0.486625   0.508994   0.518580   0.524536   0.497192   0.530850   0.409797
4      0.223985   0.542818   0.524531   0.560396   0.572842   0.587055   0.546735   0.568327   0.448185
5 rows x 163 columns

```

- We have applied data augmentation and extracted the features for each audio files and saved them.

✓ Data Preparation

- As of now we have extracted the data, now we need to normalize and split our data for training and testing.

```

X = Features.iloc[:, :-1].values
Y = Features['labels'].values

# As this is a multiclass classification problem onehotencoding our Y.
encoder = OneHotEncoder()
Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()

```

```
# splitting data
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0, shuffle=True)
x_train.shape, y_train.shape, x_test.shape, y_test.shape

((27364, 162), (27364, 8), (9122, 162), (9122, 8))

# scaling our data with sklearn's Standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_train.shape, y_train.shape, x_test.shape, y_test.shape

((27364, 162), (27364, 8), (9122, 162), (9122, 8))

# making our data compatible to model.
x_train = np.expand_dims(x_train, axis=2)
```