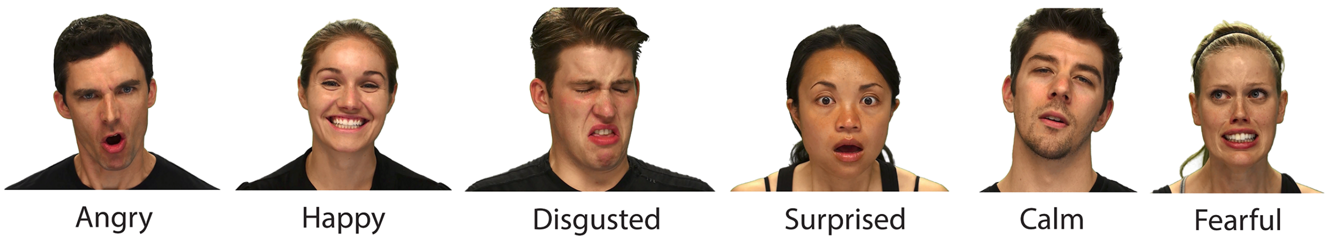
AUDIO SENTIMENT ANALYSIS By Sonic SARK

To detect Emotion of the user from audio file



Importing necessary modules

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import os
import numpy as np
import pandas as pd


import matplotlib.pyplot as plt
import seaborn as sns

import librosa as lr

import librosa
import speech_recognition as sr

import IPython.display as ipd
import librosa.display
```

1.Loading Dataset

In [4]:

```
# loading audio files of 24 actors and their respective path from Ravdess Dataset with p
import os
os.listdir(path='Dataset')

def getListOfFiles(dirName):
    listOfFile=os.listdir(dirName)
    allFiles=list()
    for entry in listOfFile:
        fullPath=os.path.join(dirName, entry)
        if os.path.isdir(fullPath):
            allFiles=allFiles + getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles

dirName = 'Dataset'
listOfFiles = getListOfFiles(dirName)
print("Total Number of Audio Files is ",len(listOfFiles))
```

```
Total Number of Audio Files is  1440
```

In [5]:

```
listOfFiles
```

```
['Dataset\\Actor_01\\03-01-01-01-01-01-01.wav',
 'Dataset\\Actor_01\\03-01-01-01-01-02-01.wav',
 'Dataset\\Actor_01\\03-01-01-01-02-01-01.wav',
 'Dataset\\Actor_01\\03-01-01-01-02-02-01.wav',
 'Dataset\\Actor_01\\03-01-02-01-01-01-01.wav',
 'Dataset\\Actor_01\\03-01-02-01-01-02-01.wav',
 'Dataset\\Actor_01\\03-01-02-01-02-01-01.wav',
 'Dataset\\Actor_01\\03-01-02-01-02-02-01.wav',
 'Dataset\\Actor_01\\03-01-02-02-01-01-01.wav',
 'Dataset\\Actor_01\\03-01-02-02-01-02-01.wav',
 'Dataset\\Actor_01\\03-01-02-02-02-01-01.wav',
 'Dataset\\Actor_01\\03-01-02-02-02-02-01.wav',
 'Dataset\\Actor_01\\03-01-03-01-01-01-01.wav',
 'Dataset\\Actor_01\\03-01-03-01-01-02-01.wav',
 'Dataset\\Actor_01\\03-01-03-01-02-01-01.wav',
 'Dataset\\Actor_01\\03-01-03-01-02-02-01.wav',
 'Dataset\\Actor_01\\03-01-03-02-01-01-01.wav',
 'Dataset\\Actor_01\\03-01-03-02-01-02-01.wav',
 'Dataset\\Actor_01\\03-01-03-02-02-01-01.wav',
```

2.Separating & Labelling Emotions from Dataset

The filenames are formatted in such a way that third part (5th and 6th value) from the filename represents the emotion of the audio.

For example--> 02-01-**06**-01-02-01-12.wav here 06 represents **fear** emotion.

In [6]:

```
import pandas as pd
file_emotion = []
file_path = []

for i in listOfFiles:
        part = i.split('.')[0]
        part = part.split('-')
        file_emotion.append(int(part[2]))
        file_path.append(i)

# Storing 3rd part of every file into emotion_df dataframe
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# Concatenating file_emotion and file_path into Ravdess_df
path_df = pd.DataFrame(file_path, columns=['Path'])
Ravdess_df = pd.concat([emotion_df, path_df], axis=1)

# Labeling integers into their respective classes
Ravdess_df.Emotions.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5:'angry', 6:'fe
```

In [7]:

```
emotion_df.head()
```

Out[7]:

	Emotions
0	1
1	1
2	1
3	1
4	2

In [8]:

```
Ravdess_df.head()
```

Out[8]:

	Emotions	Path
0	neutral	Dataset\Actor_01\03-01-01-01-01-01-01.wav
1	neutral	Dataset\Actor_01\03-01-01-01-01-02-01.wav
2	neutral	Dataset\Actor_01\03-01-01-01-02-01-01.wav
3	neutral	Dataset\Actor_01\03-01-01-01-02-02-01.wav
4	calm	Dataset\Actor_01\03-01-02-01-01-01-01.wav

In [9]:

```
Ravdess_df.shape
```

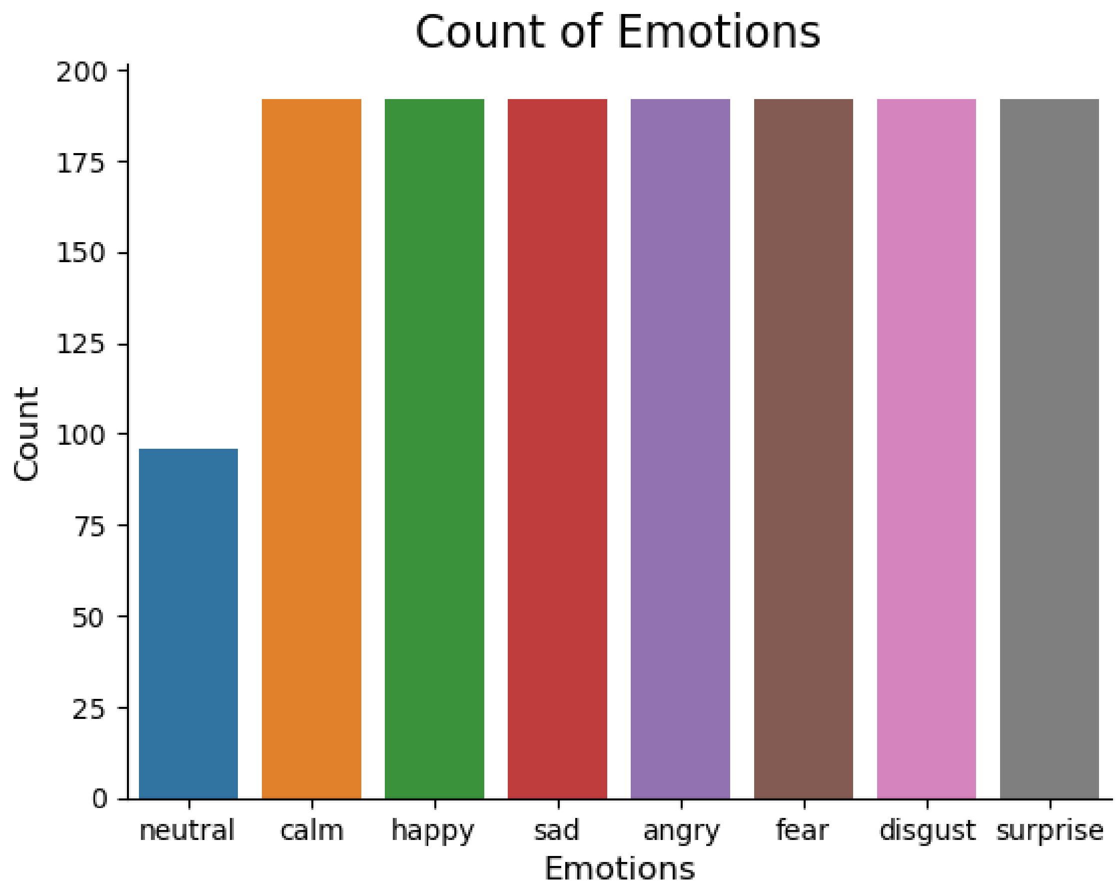
Out[9]:

```
(1440, 2)
```

3.Basic Visualization

In [10]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import math
plt.title('Count of Emotions', size=16)
sns.countplot(x=Ravdess_df.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```

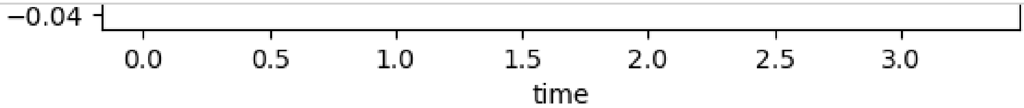


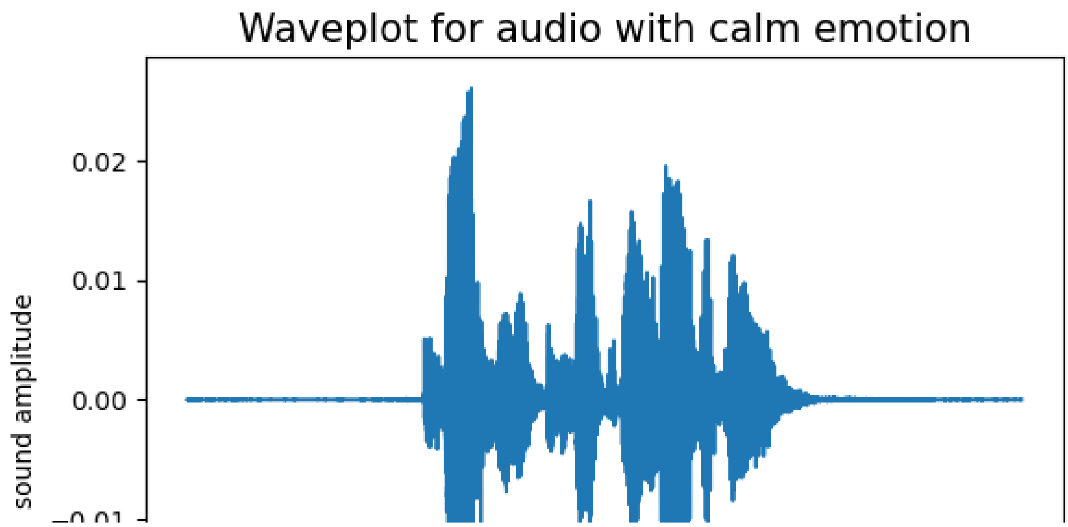
Amplitude vs Time Graph

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

In [11]:

```
# Plotting amplitude vs time graph for each emotion
import librosa as lr
import numpy as np
for i in Ravdess_df['Emotions'].unique():
    path = np.array(Ravdess_df.Path[Ravdess_df.Emotions==i])[0]
    audio,sfreq=lr.load(path)
    time=np.arange(0,len(audio))/sfreq
    fig,ax=plt.subplots()
    ax.plot(time,audio)
    ax.set(xlabel="time",ylabel="sound amplitude")
    plt.title('Waveplot for audio with {} emotion'.format(i), size=15)
    plt.show()
```





Waveplots & Spectrogram

In [13]:

```
#ploating waveplot and spectogram for each emotion
def get_audio(path,sr,e):
    print("This audio is for Emotion",e)
    ipd.display(ipd.Audio(path, rate=sr))

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

def create_spectrogram(data, sr, e):
    # stft function converts the data into short term fourier transform
    X = lr.stft(data)
    Xdb = lr.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')
    plt.colorbar()
    plt.show()
    print("-------------------------------------------------------------")

import IPython
import IPython.display as ipd
import librosa.display
y=[]
for i in Ravdess_df['Emotions']:
    emotion = i
    y.append(emotion)
    path = np.array(Ravdess_df.Path[Ravdess_df.Emotions==emotion])[2]
    data, sampling_rate = lr.load(path)
    extract_feature(data, sampling_rate)
    features=pd.DataFrame(x)
    emotions=pd.DataFrame(y)

    get_audio(path,sampling_rate,emotion)
    create_waveplot(data, sampling_rate, emotion)
    create_spectrogram(data, sampling_rate, emotion)
```

```
 1.01201199e-02 7.20746303e-03 3.15602086e-02 5.07584698e-02
 2.98722014e-02 8.87565035e-03 1.61066465e-02 5.60955182e-02
 4.76533882e-02 5.86342067e-03 5.97153232e-03 6.07381053e-02
 5.00289053e-02 3.08345221e-02 5.38572064e-03 8.97947990e-04
 2.12157075e-03 1.13319478e-03 2.16878322e-03 1.46298110e-03
 4.23635036e-04 8.01405637e-04 5.60195709e-04 5.13196574e-04
 4.69516352e-04 8.81678541e-04 1.76419097e-03 1.12101843e-03
 2.45607551e-03 2.91236932e-03 1.71205737e-02 1.54236481e-02
 3.62386298e-03 1.78652618e-03 2.37545744e-03 1.04755443e-03
 5.23059163e-04 1.21909799e-03 1.22118613e-03 7.21882679e-04
 1.25872414e-03 3.58277815e-04 9.06472211e-04 7.91530532e-04
 4.28281201e-04 2.93345569e-04 1.21435958e-04 1.28018117e-04
 1.49023181e-04 1.25625695e-04 2.16733883e-04 1.80183910e-04
 1.34798654e-04 1.43540135e-04 1.04141713e-04 1.13456728e-04
 1.45042126e-04 3.21479485e-04 3.88421351e-04 5.17632929e-04
 6.01599633e-04 2.28883917e-04 1.11606525e-04 6.57247292e-05
 6.29530405e-05 4.86985336e-05 5.21162547e-05 9.37330988e-05
 1.01586498e-04 1.73837136e-04 3.73912888e-04 4.65384714e-04
 5.75100770e-04 4.38977208e-04 2.04050855e-04 1.14512070e-04
 1.14800408e-04 6.18392005e-05 6.03446351e-05 1.08956134e-04
```

4.Feature Extraction

In [12]:

```python
#Feature Extraction of Audio Files Function
#Extract features (mfcc, chroma, mel) from a sound file
import pandas as pd
x=[]
def extract_feature(data,sampling_rate):
    result=np.array([])
    #Path=i
    #result=np.hstack((result, Path))

    stft = np.abs(librosa.stft(data))
    chromagram = np.mean(librosa.feature.chroma_stft(S=stft, sr=sampling_rate).T, axis=0
    result=np.hstack((result, chromagram))


    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sampling_rate, n_mfcc=40).T, axis=0)
    result=np.hstack((result, mfcc))


    mel=np.mean(librosa.feature.melspectrogram(y=data, sr=sampling_rate,n_mels=128).T,ax
    result=np.hstack((result, mel))

    x.append(result)


    print(result)


    return result
```

In [230]:

```
# Convering list of all extracted feature(x)and emotions(y) into dataframe (features & e
features=pd.DataFrame(x)
emotions=pd.DataFrame(y)
```

In [14]:

```
#Creating dataframe of all extracted features and our target variable
final_dataframe = pd.concat([emotions,features], axis=1)
#final_dataframe=features
```

In [15]:

```
# Storing final_dataframe into final_dataframe.csv for further computation
final_dataframe.to_csv('feature_extract.csv', index=False)
```

5.EDA (Exploratory Data Analysis)

Variable identification & Data Types

In [16]:

```
final_dataframe.dtypes
```

Out[16]:

```
0       object
0      float64
1      float64
2      float64
3      float64
        ...
175    float64
176    float64
177    float64
178    float64
179    float64
Length: 181, dtype: object
```

In [17]:

```
final_dataframe, final_dataframe.nunique()
```

```
 2    neutral  0.765898  0.793222  0.788278  0.798253  0.769865  0.680
034
 3    neutral  0.765898  0.793222  0.788278  0.798253  0.769865  0.680
034
 4       calm  0.688526  0.738672  0.771075  0.790328  0.739264  0.667
541
 ..       ...       ...       ...       ...       ...       ...
...
 129     calm  0.688526  0.738672  0.771075  0.790328  0.739264  0.667
541
 130     calm  0.688526  0.738672  0.771075  0.790328  0.739264  0.667
541
 131     calm  0.688526  0.738672  0.771075  0.790328  0.739264  0.667
541
 132    happy  0.693848  0.723479  0.759768  0.732952  0.684231  0.652
018
 133    happy  0.693848  0.723479  0.759768  0.732952  0.684231  0.652
018

          6         7         8    ...        170       171       172
```

Analyzing the basic metrics

In [18]:

```
final_dataframe.shape
```

Out[18]:

```
(134, 181)
```

In [19]:

```
final_dataframe.describe()
```

Out[19]:

	0	1	2	3	4	5	6
count	134.000000	134.000000	134.000000	134.000000	134.000000	134.000000	134.000000
mean	0.716728	0.749322	0.763863	0.765740	0.713684	0.658474	0.665867
std	0.035432	0.021792	0.031271	0.029436	0.033782	0.022042	0.018746
min	0.670340	0.723479	0.708731	0.714755	0.671781	0.612217	0.637457
25%	0.688526	0.731805	0.759768	0.735256	0.684231	0.652018	0.656162
50%	0.705003	0.744671	0.771075	0.788807	0.730402	0.667541	0.659589
75%	0.756434	0.755143	0.788278	0.790328	0.739264	0.675921	0.673258
max	0.771060	0.793222	0.797124	0.798253	0.769865	0.682063	0.698304

8 rows × 180 columns

In [20]:

```
final_dataframe.groupby([1]).mean()
```

Out[20]:

	0	2	3	4	5	6	7	8	
1									
0.723479	0.693848	0.759768	0.732952	0.684231	0.652018	0.655730	0.710225	0.730609	0.
0.729516	0.670340	0.777541	0.768362	0.676864	0.612217	0.637457	0.697412	0.705373	0.
0.738672	0.688526	0.771075	0.790328	0.739264	0.667541	0.657456	0.680098	0.703753	0.
0.744671	0.705003	0.795186	0.790227	0.730402	0.682063	0.659589	0.693474	0.695054	0.
0.746541	0.711967	0.716214	0.742167	0.749842	0.675921	0.659724	0.717568	0.748187	0.
0.755143	0.756434	0.708731	0.714755	0.692161	0.636235	0.698304	0.776760	0.778452	0.
0.782866	0.771060	0.797124	0.788807	0.671781	0.663426	0.692734	0.725988	0.764391	0.
0.793222	0.765898	0.788278	0.798253	0.769865	0.680034	0.673258	0.718917	0.763575	0.

8 rows × 179 columns

Non-Graphical Univariate Analysis

In [21]:

```
final_dataframe.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[21]:

```
0      0
0      0
1      0
2      0
3      0
      ..
175    0
176    0
177    0
178    0
179    0
Length: 181, dtype: int64
```

In [22]:

```
final_dataframe.isnull().values.any()
```

Out[22]:

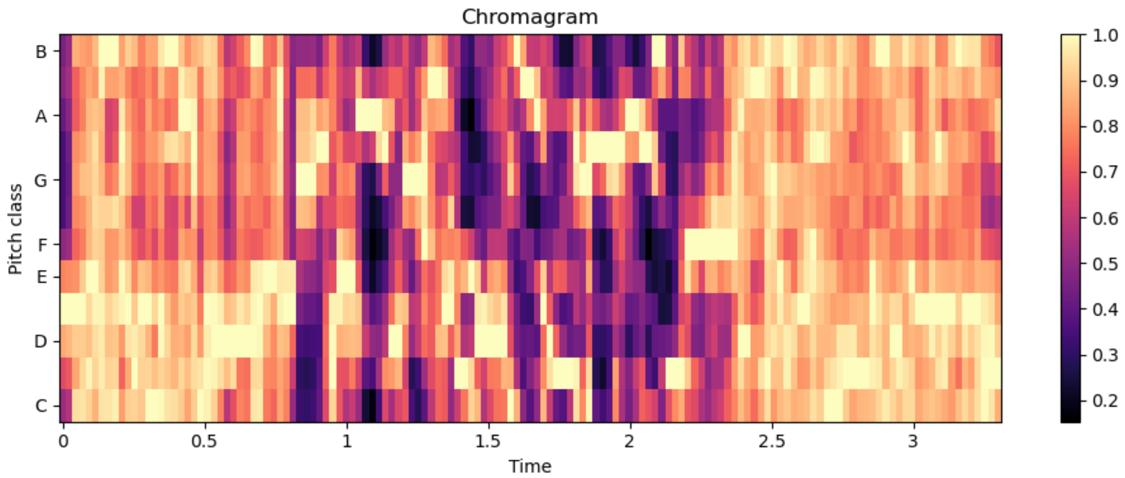
```
False
```

Graphical Representation Of features

1. Chroma

In [23]:

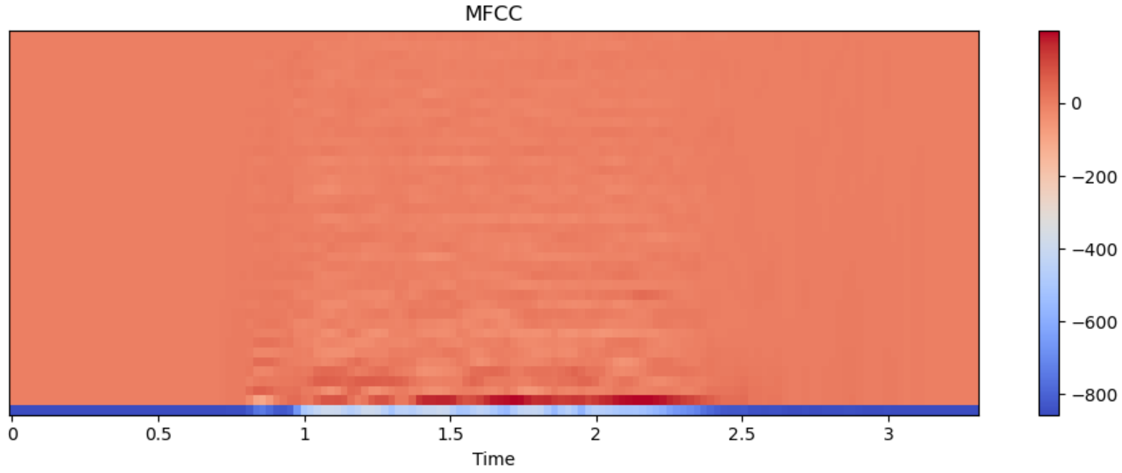
```
x,sr=librosa.load(Ravdess_df.Path[0])
S = np.abs(librosa.stft(x))
chroma = librosa.feature.chroma_stft(S=S, sr=sr)
plt.figure(figsize=(10, 4))
librosa.display.specshow(chroma, y_axis='chroma', x_axis='time')
plt.colorbar()
plt.title('Chromagram')
plt.tight_layout()
```



2. MFCC

In [24]:

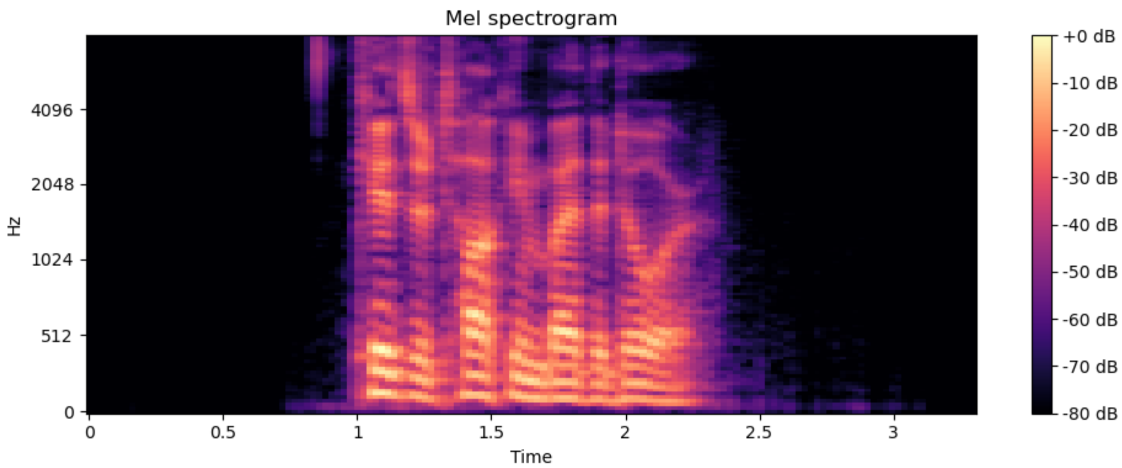
```
mfccs = librosa.feature.mfcc(y=x, sr=sr, n_mfcc=40)
plt.figure(figsize=(10, 4))
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('MFCC')
plt.tight_layout()
```



3. Mel Spectogram

In [25]:

```
S = librosa.feature.melspectrogram(y=x, sr=sr, n_mels=128,fmax=8000)
plt.figure(figsize=(10, 4))
librosa.display.specshow(librosa.power_to_db(S,ref=np.max),y_axis='mel', fmax=8000,x_axi
plt.colorbar(format='%+2.0f dB')
plt.title('Mel spectrogram')
plt.tight_layout()
```



6.Data Pre-processing

In [26]:

```
Ravdess_df=pd.read_csv("feature_extract.csv")
```

In [27]:

```
Ravdess_df.head()
```

Out[27]:

	0	0.1	1	2	3	4	5	6	7
0	neutral	0.765898	0.793222	0.788278	0.798253	0.769865	0.680034	0.673258	0.718917
1	neutral	0.765898	0.793222	0.788278	0.798253	0.769865	0.680034	0.673258	0.718917
2	neutral	0.765898	0.793222	0.788278	0.798253	0.769865	0.680034	0.673258	0.718917
3	neutral	0.765898	0.793222	0.788278	0.798253	0.769865	0.680034	0.673258	0.718917
4	calm	0.688526	0.738672	0.771075	0.790328	0.739264	0.667541	0.657456	0.680098

5 rows × 181 columns

Seperating target variable and feature variable

In [28]:

```
X = Ravdess_df.iloc[: ,1:].values
y=Ravdess_df["0"].values
```

Splitting X and y into train and test set

In [29]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=42,test_size=0.2(
```

In [30]:

```
print(f'Features extracted: {x_train.shape[1]}')
```

```
Features extracted: 180
```

In [31]:

```
y
```

Out[31]:

```
array(['neutral', 'neutral', 'neutral', 'neutral', 'calm', 'calm', 'cal
m',
       'calm', 'calm', 'calm', 'calm', 'calm', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'sad', 'sad', 'sad',
       'sad', 'sad', 'sad', 'sad', 'sad', 'angry', 'angry', 'angry',
       'angry', 'angry', 'angry', 'angry', 'angry', 'fear', 'fear',
       'fear', 'fear', 'fear', 'fear', 'fear', 'fear', 'disgust',
       'disgust', 'disgust', 'disgust', 'disgust', 'disgust', 'disgust',
       'disgust', 'surprise', 'surprise', 'surprise', 'surprise',
       'surprise', 'surprise', 'surprise', 'surprise', 'neutral',
       'neutral', 'neutral', 'neutral', 'calm', 'calm', 'calm', 'calm',
       'calm', 'calm', 'calm', 'calm', 'happy', 'happy', 'happy', 'happ
y',
       'happy', 'happy', 'happy', 'happy', 'sad', 'sad', 'sad', 'sad',
       'sad', 'sad', 'sad', 'sad', 'angry', 'angry', 'angry', 'angry',
       'angry', 'angry', 'angry', 'angry', 'fear', 'fear', 'fear', 'fea
r',
       'fear', 'fear', 'fear', 'fear', 'disgust', 'disgust', 'disgust',
       'disgust', 'disgust', 'disgust', 'disgust', 'disgust', 'surprise',
       'surprise', 'surprise', 'surprise', 'surprise', 'surprise',
       'surprise', 'surprise', 'neutral', 'neutral', 'neutral', 'neutra
l',
       'calm', 'calm', 'calm', 'calm', 'calm', 'calm', 'calm', 'calm',
       'happy', 'happy'], dtype=object)
```

7. Model Building

Multi Layer Perceptron Classifier (MLP)

In [32]:

```
# Initializing the Multi Layer Perceptron Classifier
from sklearn.neural_network import MLPClassifier
model=MLPClassifier(alpha=0.01, batch_size=16, epsilon=1e-08, hidden_layer_sizes=(1000),
```

In [ ]:

In [33]:

```
# Train the model
model.fit(x_train,y_train)
```

Out[33]:

```
▾                       MLPClassifier
MLPClassifier(alpha=0.01, batch_size=16, hidden_layer_sizes=1000,
              learning_rate='adaptive', max_iter=500)
```

In [38]:

```
# Predict for the test set
y_pred=model.predict(x_test)
```

In [39]:

```
# Calculate the accuracy of our model
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_true=y_test, y_pred=y_pred)

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

```
Accuracy: 100.00%
```

In [40]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       angry       1.00      1.00      1.00         1
        calm       1.00      1.00      1.00         5
     disgust       1.00      1.00      1.00         6
        fear       1.00      1.00      1.00         5
       happy       1.00      1.00      1.00         5
     neutral       1.00      1.00      1.00         1
         sad       1.00      1.00      1.00         4

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27
```

In [41]:

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test,y_pred)
print (matrix)
```

```
[[1 0 0 0 0 0 0]
 [0 5 0 0 0 0 0]
 [0 0 6 0 0 0 0]
 [0 0 0 5 0 0 0]
 [0 0 0 0 5 0 0]
 [0 0 0 0 0 1 0]
 [0 0 0 0 0 0 4]]
```

In [42]:

```
result = pd.DataFrame({'predicted_values': y_pred})
result["Actual_Values"]=pd.DataFrame(y_test)
result[:10]
```

Out[42]:

	predicted_values	Actual_Values
0	calm	calm
1	calm	calm
2	disgust	disgust
3	happy	happy
4	fear	fear
5	neutral	neutral
6	happy	happy
7	disgust	disgust
8	fear	fear
9	fear	fear

In [43]:

```
# SAVING THE MODEL
# Saving the Model to file in the current working directory

import pickle
Pkl_Filename = "new_mlp_74.pkl"
with open(Pkl_Filename, 'wb') as file:
    pickle.dump(model, file)
```

On Microphone data

In [44]:

```
import librosa
import speech_recognition as sr

# obtain audio from the microphone
r = sr.Recognizer()
with sr.Microphone() as source:
    print("Hiii SARK's Say something!")
    audio = r.listen(source,phrase_time_limit=4)

# write audio to a WAV file
with open("output1.wav", "wb") as f:
    f.write(audio.get_wav_data())
```

```
Hiii SARK's Say something!
```

In [45]:

```
#Audio to text

txt=sr.AudioFile("output1.wav")

with txt as source:
    audio = r.record(source)
try:
    s = r.recognize_google(audio)
    print("You Said  : "+s)
except Exception as e:
    print("Exception: "+str(e))

ipd.display(ipd.Audio('output1.wav'))
```

```
Exception: FLAC conversion utility not available - consider installing th
e FLAC command line application by running `apt-get install flac` or your
operating system's equivalent
```

0:00 / 0:01

In [46]:

```
x=[]

#for i in range (0,20,1):
data, sampling_rate = librosa.load('output1.wav')
feature=extract_feature(data,sampling_rate)
x.append(feature)
print(data,"  ",sampling_rate)
```

```
[ 7.19771445e-01  6.95009232e-01  6.67005479e-01  6.66425228e-01
  6.88024461e-01  7.93078482e-01  8.24678004e-01  7.44679630e-01
  6.95955276e-01  6.89126372e-01  6.81473672e-01  7.17974305e-01
 -7.38146240e+02  8.76847458e+01 -1.18114824e+01  3.15197811e+01
  1.02429838e+01 -5.07917166e+00 -1.16378202e+01 -2.18770576e+00
 -8.04884434e+00 -6.03404713e+00 -5.19719839e+00 -1.41543531e+01
 -8.77010345e+00 -9.61664581e+00 -7.03260040e+00 -7.79095936e+00
 -3.10951471e+00 -2.09853005e+00 -1.80778742e+00 -2.16891602e-01
 -5.61715174e+00 -4.63197708e+00 -4.83059216e+00 -4.96079397e+00
 -9.07571125e+00 -8.03846264e+00 -6.11125088e+00 -8.71125221e+00
 -8.21972179e+00 -7.25804901e+00 -6.27532864e+00 -6.27534294e+00
 -4.69210911e+00 -3.64799857e+00 -4.49970293e+00 -3.76720953e+00
 -2.17726469e+00 -3.08601046e+00 -2.50497198e+00 -5.24530530e-01
  2.40944473e-07  1.33845060e-06  9.42448096e-05  2.18946468e-02
  8.98935720e-02  1.16930343e-02  7.21866847e-04  9.62046208e-04
  3.08796228e-03  3.93517455e-03  1.20226655e-03  1.17383339e-03
  5.08103007e-03  2.20317896e-02  1.78152416e-02  1.82339188e-03
  1.24996819e-03  3.74277122e-04  2.70055811e-04  4.14397655e-04
  2.33789309e-04  1.35492723e-04  1.20035322e-04  1.46219740e-04
  2.70996592e-04  2.87615374e-04  1.05744446e-04  8.85680129e-05
  4.99870002e-05  5.36114057e-05  7.62365162e-05  2.81699467e-05
  2.41754769e-05  3.47635141e-05  4.58337781e-05  2.86408886e-05
  1.16036053e-05  1.38763262e-05  4.18436321e-05  3.25280125e-05
  6.24836684e-06  7.42279781e-06  1.23664677e-05  1.41850296e-05
  1.70924868e-05  1.78520149e-05  3.69403569e-05  3.26089503e-05
  2.91861907e-05  4.23184101e-05  2.62165759e-05  2.49855530e-05
  2.31375307e-05  1.75283767e-05  2.31943613e-05  4.89516497e-05
  7.87655299e-05  3.80369784e-05  4.43456265e-05  3.21569860e-05
  3.82258950e-05  3.22419146e-05  2.27112614e-05  1.51952190e-05
  4.91285646e-05  5.61347879e-05  5.03976626e-05  9.51050097e-05
  2.73305504e-05  2.22604176e-05  2.50886096e-05  1.99285278e-05
  1.38303403e-05  9.48834531e-06  5.99565328e-06  7.14695398e-06
  1.11112122e-05  1.46900866e-05  1.88056274e-05  9.16325644e-06
  6.51719984e-06  5.26327267e-06  4.75938577e-06  3.33199409e-06
  3.00016040e-06  2.04676235e-06  8.93836557e-07  6.61818831e-07
  9.74272666e-07  7.30665647e-07  2.28891764e-07  1.92900217e-07
  2.25514313e-07  1.84985851e-07  1.28956884e-07  1.08913568e-07
  1.42017910e-07  9.06681805e-08  9.70940164e-08  8.66810339e-08
  9.08937707e-08  6.44085816e-08  5.82763384e-08  5.36932383e-08
  6.03542532e-08  6.44975842e-08  6.44000124e-08  6.48734684e-08
  7.87161767e-08  7.45870281e-08  5.75781094e-08  4.52435849e-08
  2.90948545e-08  2.66290456e-08  1.38983109e-08  9.21481913e-09
  8.10894729e-09  8.05384204e-09  8.64991456e-09  8.50186144e-09
  7.76846765e-09  8.21615309e-09  8.46525428e-09  8.42797032e-09
  7.59531371e-09  7.83027687e-09  4.43444526e-09  3.58026803e-10]
[-1.2170676e-07 -1.5612781e-05  1.5712767e-06 ...  1.2616920e-05
 -1.3108790e-05 -7.3623114e-06]    22050
```

In [47]:

```
import pickle
Pkl_Filename = "new_mlp_79.pkl"
# Loading the Model back from file
with open(Pkl_Filename, 'rb') as file:
    model= pickle.load(file)
```

In [48]:

```
model.predict(x)
```

Out[48]:

```
array(['calm', 'calm'], dtype='<U8')
```

Sequential

Separating target and feature variables

In [49]:

```
X = Ravdess_df.iloc[: ,1:].values
Y = Ravdess_df['0'].values
```

In [50]:

```
X
```

Out[50]:

```
array([[7.65898407e-01, 7.93222010e-01, 7.88278162e-01, ...,
        2.42458536e-05, 1.34415195e-05, 1.11975589e-06],
       [7.65898407e-01, 7.93222010e-01, 7.88278162e-01, ...,
        2.42458536e-05, 1.34415195e-05, 1.11975589e-06],
       [7.65898407e-01, 7.93222010e-01, 7.88278162e-01, ...,
        2.42458536e-05, 1.34415195e-05, 1.11975589e-06],
       ...,
       [6.88525736e-01, 7.38671720e-01, 7.71074533e-01, ...,
        1.22736194e-04, 4.09652894e-05, 2.65603944e-06],
       [6.93848133e-01, 7.23478615e-01, 7.59767771e-01, ...,
        1.89758448e-05, 1.28757110e-05, 8.03474563e-07],
       [6.93848133e-01, 7.23478615e-01, 7.59767771e-01, ...,
        1.89758448e-05, 1.28757110e-05, 8.03474563e-07]])
```

One Hot Encoding our Target variable

In [51]:

```
# As this is a multiclass classification problem, we're onehotencoding our Y
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()
```

Splitting X and y into train and test set

In [52]:

```
# Splitting data in Train-Test sets

x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=9, test_size=0.1
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[52]:

```
((120, 180), (120, 8), (14, 180), (14, 8))
```

Scaling Data

In [53]:

```
# Scaling our data with sklearn's Standard scaler
from sklearn.preprocessing import StandardScaler
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
```

Out[53]:

```
((120, 180), (120, 8), (14, 180), (14, 8))
```

Making data compatible for Neural Networks

In [54]:

```
x_train = np.expand_dims(x_train, axis=2)
x_test = np.expand_dims(x_test, axis=2)
```

In [55]:

```
x_train.shape, x_test.shape
```

Out[55]:

```
((120, 180, 1), (14, 180, 1))
```

Importing necessary modules

In [56]:

```
import keras
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from keras.optimizers import RMSprop
```

Model Definition

In [57]:

```
model = Sequential()

model.add(Conv1D(256, 5,padding='same',input_shape=(180,1)))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(MaxPooling1D(pool_size=(4)))
model.add(Conv1D(128, 5,padding='same',))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(MaxPooling1D(pool_size=(4)))
model.add(Conv1D(64, 5,padding='same',))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(8))
model.add(Activation('softmax'))
opt = keras.optimizers.RMSprop(lr=0.00005, rho=0.9, epsilon=1e-07, decay=0.0)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d (Conv1D)             (None, 180, 256)          1536      
                                                                 
 activation (Activation)     (None, 180, 256)          0         
                                                                 
 dropout (Dropout)           (None, 180, 256)          0         
                                                                 
 max_pooling1d (MaxPooling1D  (None, 45, 256)          0         
 )                                                               
                                                                 
 conv1d_1 (Conv1D)           (None, 45, 128)           163968    
                                                                 
 activation_1 (Activation)   (None, 45, 128)           0         
                                                                 
 dropout_1 (Dropout)         (None, 45, 128)           0         
                                                                 
 max_pooling1d_1 (MaxPooling  (None, 11, 128)          0         
 1D)                                                             
                                                                 
 conv1d_2 (Conv1D)           (None, 11, 64)            41024     
                                                                 
 activation_2 (Activation)   (None, 11, 64)            0         
                                                                 
 dropout_2 (Dropout)         (None, 11, 64)            0         
                                                                 
 flatten (Flatten)           (None, 704)               0         
                                                                 
 dense (Dense)               (None, 8)                 5640      
                                                                 
 activation_3 (Activation)   (None, 8)                 0         
                                                                 
=================================================================
Total params: 212,168
Trainable params: 212,168
Non-trainable params: 0
_________________________________________________________________
```

Model compiling

In [58]:

```
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
```

In [59]:

```
y_train
```

```
       [1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
```

Model fitting

In [60]:

```
cnnhistory=model.fit(x_train, y_train, batch_size=16, epochs=200, validation_data=(x_tes
```

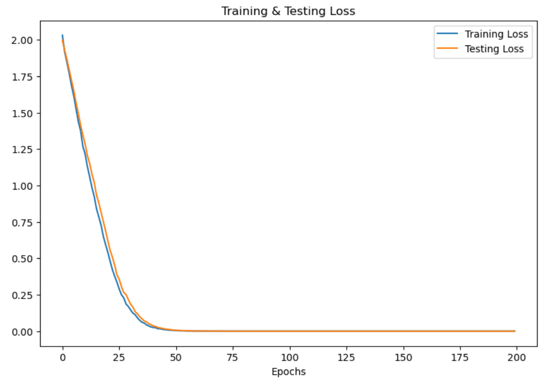
```
ccuracy: 0.1833 - val_loss: 1.9976 - val_accuracy: 0.2857
Epoch 2/200
8/8 [==============================] - 0s 51ms/step - loss: 1.9160 - a
ccuracy: 0.3917 - val_loss: 1.9273 - val_accuracy: 0.3571
Epoch 3/200
8/8 [==============================] - 0s 45ms/step - loss: 1.8443 - a
ccuracy: 0.5083 - val_loss: 1.8652 - val_accuracy: 0.3571
Epoch 4/200
8/8 [==============================] - 0s 42ms/step - loss: 1.7690 - a
ccuracy: 0.5583 - val_loss: 1.7924 - val_accuracy: 0.3571
Epoch 5/200
8/8 [==============================] - 0s 43ms/step - loss: 1.6856 - a
ccuracy: 0.6167 - val_loss: 1.7223 - val_accuracy: 0.7143
Epoch 6/200
8/8 [==============================] - 0s 43ms/step - loss: 1.6138 - a
ccuracy: 0.7417 - val_loss: 1.6521 - val_accuracy: 0.8571
Epoch 7/200
8/8 [==============================] - 0s 41ms/step - loss: 1.5254 - a
ccuracy: 0.8667 - val_loss: 1.5675 - val_accuracy: 0.8571
Epoch 8/200
```

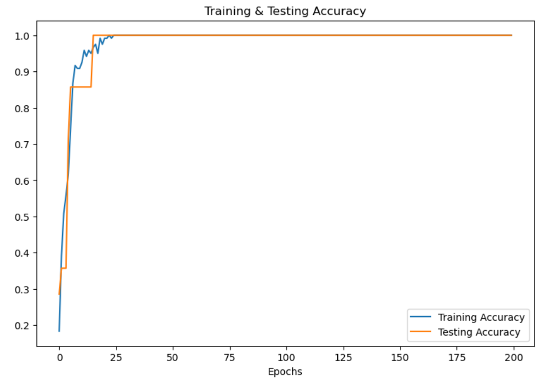
Model analysis

In [61]:

```
print("Accuracy of our model on test data : " , model.evaluate(x_test,y_test)[1]*100 , '

epochs = [i for i in range(200)]
fig , ax = plt.subplots(1,2)
train_acc = cnnhistory.history['accuracy']
train_loss = cnnhistory.history['loss']
test_acc = cnnhistory.history['val_accuracy']
test_loss = cnnhistory.history['val_loss']

fig.set_size_inches(20,6)
ax[0].plot(epochs , train_loss , label = 'Training Loss')
ax[0].plot(epochs , test_loss , label = 'Testing Loss')
ax[0].set_title('Training & Testing Loss')
ax[0].legend()
ax[0].set_xlabel("Epochs")

ax[1].plot(epochs , train_acc , label = 'Training Accuracy')
ax[1].plot(epochs , test_acc , label = 'Testing Accuracy')
ax[1].set_title('Training & Testing Accuracy')
ax[1].legend()
ax[1].set_xlabel("Epochs")
plt.show()
```

```
1/1 [==============================] - 0s 28ms/step - loss: 4.2575e-08 -
accuracy: 1.0000
Accuracy of our model on test data :  100.0 %
```





Saving the Model

In [63]:

```
model_name = 'By_Sequential_12-4.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

```
Saved trained model at C:\Users\ASUS\Desktop\Project Final\saved_models\B
y_Sequential_12-4.h5
```

In [64]:

```
import json
model_json = model.to_json()
with open("By_Sequential_12-4.json", "w") as json_file:
    json_file.write(model_json)
```

Loading the model

In [66]:

```
# Loading the model
# Loading json and creating model

from keras.models import model_from_json
json_file = open('By_Sequential_12-4.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load weights into new model
loaded_model.load_weights("saved_models/By_Sequential_12-4.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'
score = loaded_model.evaluate(x_test, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 100.00%
```

Applying model on test data

In [67]:

```
preds = loaded_model.predict(x_test, batch_size=10,verbose=1)
```

```
2/2 [==============================] - 0s 19ms/step
```

In [68]:

```
predict=preds.argmax(axis=1)
actual=y_test.argmax(axis=1)
```

In [69]:

```
predict = pd.DataFrame(predict, columns=['Predicted_y'])
actual = pd.DataFrame(actual, columns=['Actual'])
```

In [70]:

```
result=pd.concat([predict,actual],axis=1)
```

In [71]:

```
result['Predicted_y']=result['Predicted_y'].map({0:'neutral', 1:'calm', 2:'happy', 3:'sa
result['Actual']=result['Actual'].map({0:'neutral', 1:'calm', 2:'happy', 3:'sad', 4:'ang
```

In [72]:

```
result.head(6)
```

Out[72]:

	Predicted_y	Actual
0	calm	calm
1	fear	fear
2	calm	calm
3	calm	calm
4	surprise	surprise
5	fear	fear

In [73]:

```
#by microphone
op=np.expand_dims(x,-1)
pred=loaded_model.predict(op)
```

```
1/1 [==============================] - 0s 26ms/step
```

In [ ]:

```

```