Speech Emotion Recognition with CNN

In [29]:
```
# Import libraries
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.pyplot import specgram
import pandas as pd
import glob
from sklearn.metrics import confusion_matrix
import IPython.display as ipd
import os
import sys
import warnings
# ignore warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [30]:
```
#for dirname, _, filenames in os.walk('/input'):
#    for filename in filenames:
#        print(os.path.join(dirname, filename))

TESS = "/input/toronto-emotional-speech-set-tess/tess toronto emotional speech set dat
RAV = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
SAVEE = "/input/surrey-audiovisual-expressed-emotion-savee/ALL/"
CREMA = "/input/cremad/AudioWAV/"

# Run one example
dir_list = os.listdir(SAVEE)
dir_list[0:5]
```

Out[30]:
```
['JE_h09.wav', 'KL_f12.wav', 'DC_h03.wav', 'DC_d04.wav', 'KL_a14.wav']
```

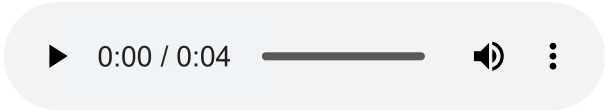
In [31]:
```
# Get the data location for SAVEE
dir_list = os.listdir(SAVEE)

# parse the filename to get the emotions
emotion=[]
path = []
for i in dir_list:
    if i[-8:-6]=='_a':
        emotion.append('male_angry')
    elif i[-8:-6]=='_d':
        emotion.append('male_disgust')
    elif i[-8:-6]=='_f':
        emotion.append('male_fear')
    elif i[-8:-6]=='_h':
        emotion.append('male_happy')
    elif i[-8:-6]=='_n':
        emotion.append('male_neutral')
    elif i[-8:-6]=='sa':
        emotion.append('male_sad')
```

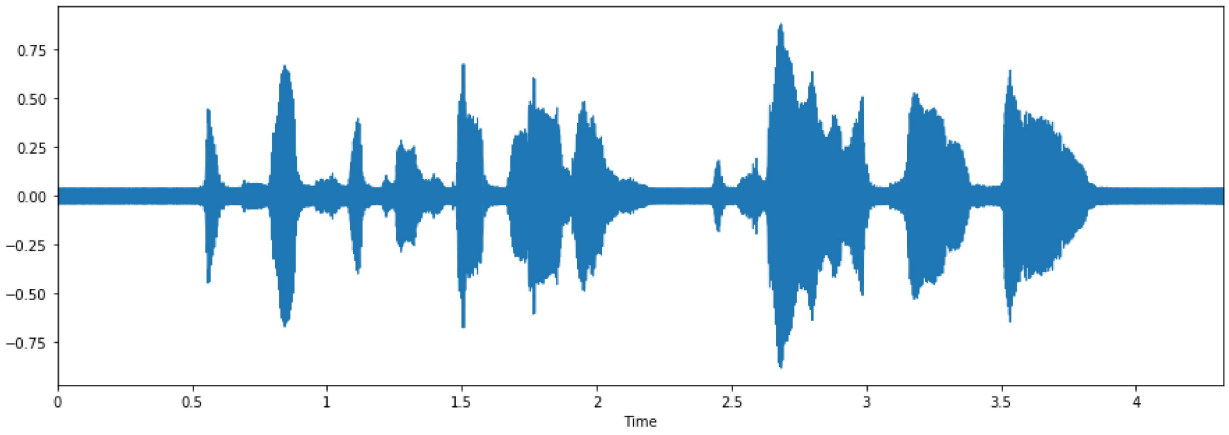
```
    elif i[-8:-6]=='su':
        emotion.append('male_surprise')
    else:
        emotion.append('male_error')
    path.append(SAVEE + i)

# Now check out the label count distribution
SAVEE_df = pd.DataFrame(emotion, columns = ['labels'])
SAVEE_df['source'] = 'SAVEE'
SAVEE_df = pd.concat([SAVEE_df, pd.DataFrame(path, columns = ['path'])], axis = 1)
SAVEE_df.labels.value_counts()
```

Out[31]:
```
male_neutral     120
male_angry        60
male_happy        60
male_sad          60
male_surprise     60
male_fear         60
male_disgust      60
Name: labels, dtype: int64
```

In [32]:
```
# use Librosa library for this task
fname = SAVEE + 'DC_f11.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

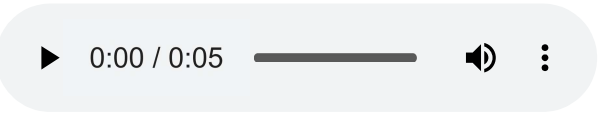
Out[32]:

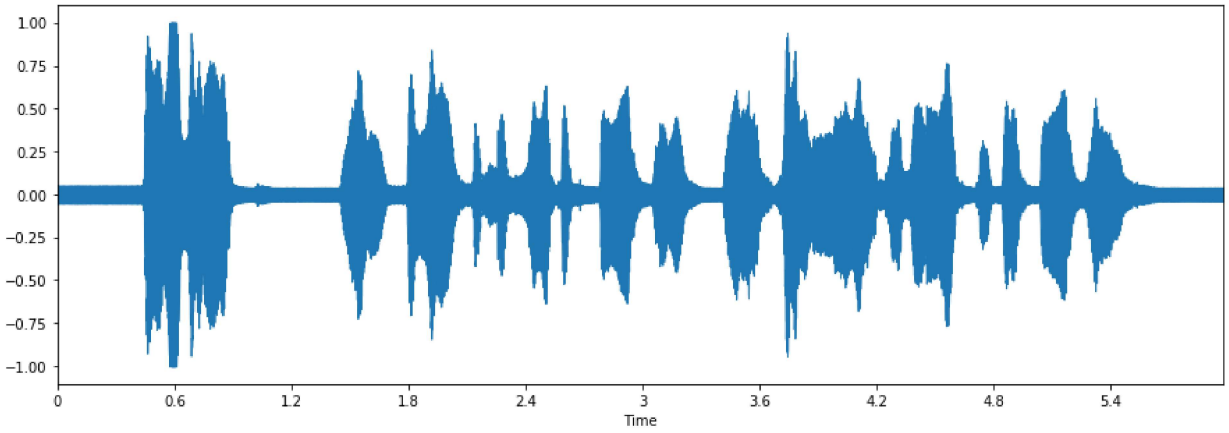




In [33]:
```
# Lets play a happy track
fname = SAVEE + 'DC_h11.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

Out[33]:





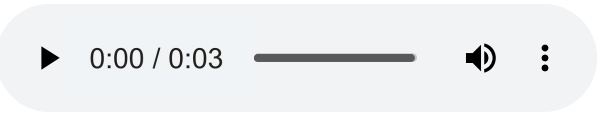
In [34]:

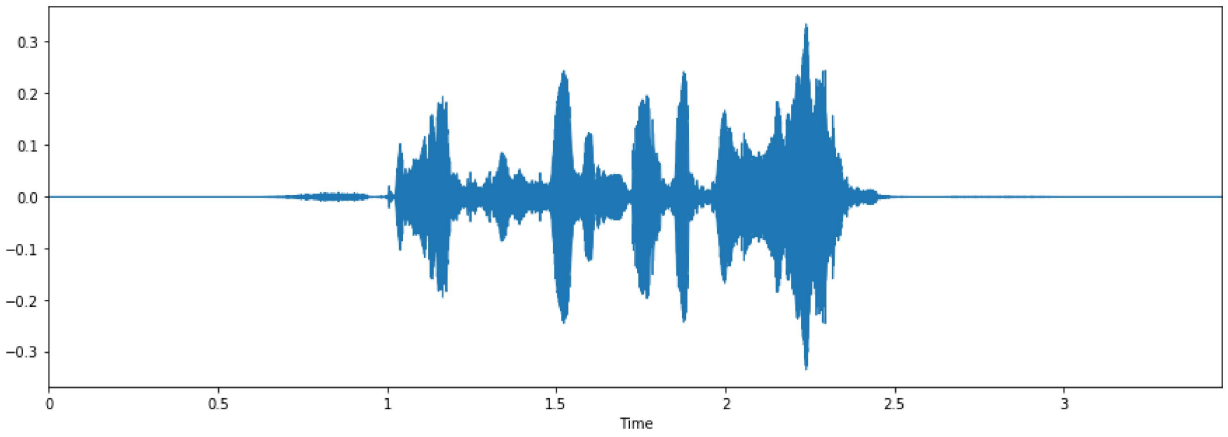
```
dir_list = os.listdir(RAV)
dir_list.sort()

emotion = []
gender = []
path = []
for i in dir_list:
    fname = os.listdir(RAV + i)
    for f in fname:
        part = f.split('.')[0].split('-')
        emotion.append(int(part[2]))
        temp = int(part[6])
        if temp%2 == 0:
            temp = "female"
        else:
            temp = "male"
        gender.append(temp)
        path.append(RAV + i + '/' + f)


RAV_df = pd.DataFrame(emotion)
RAV_df = RAV_df.replace({1:'neutral', 2:'neutral', 3:'happy', 4:'sad', 5:'angry', 6:'f
RAV_df = pd.concat([pd.DataFrame(gender),RAV_df],axis=1)
RAV_df.columns = ['gender','emotion']
RAV_df['labels'] =RAV_df.gender + '_' + RAV_df.emotion
RAV_df['source'] = 'RAVDESS'
RAV_df = pd.concat([RAV_df,pd.DataFrame(path, columns = ['path'])],axis=1)
RAV_df = RAV_df.drop(['gender', 'emotion'], axis=1)
RAV_df.labels.value_counts()
```

Out[34]:
```
male_neutral       144
female_neutral     144
female_sad          96
female_fear         96
male_happy          96
male_angry          96
female_disgust      96
male_surprise       96
male_fear           96
female_happy        96
female_surprise     96
male_sad            96
female_angry        96
male_disgust        96
Name: labels, dtype: int64
```

In [35]:
```
# Pick a fearful track
fname = RAV + 'Actor_14/03-01-06-02-02-02-14.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

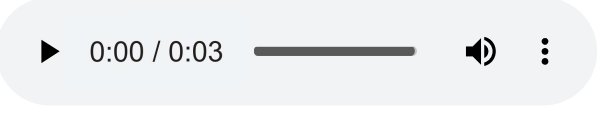
Out[35]:

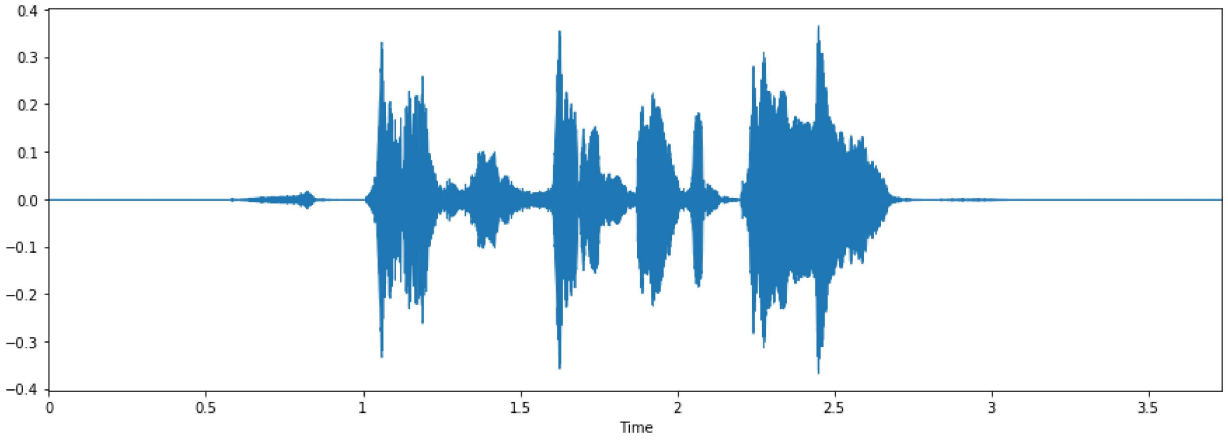




In [36]:
```
# Pick a happy track
fname = RAV + 'Actor_14/03-01-03-02-02-02-14.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

Out[36]:





In [37]:
```
dir_list = os.listdir(TESS)
dir_list.sort()
dir_list
```

Out[37]:
```
['OAF_Fear',
 'OAF_Pleasant_surprise',
 'OAF_Sad',
 'OAF_angry',
 'OAF_disgust',
 'OAF_happy',
 'OAF_neutral',
 'YAF_angry',
 'YAF_disgust',
 'YAF_fear',
 'YAF_happy',
 'YAF_neutral',
 'YAF_pleasant_surprised',
 'YAF_sad']
```

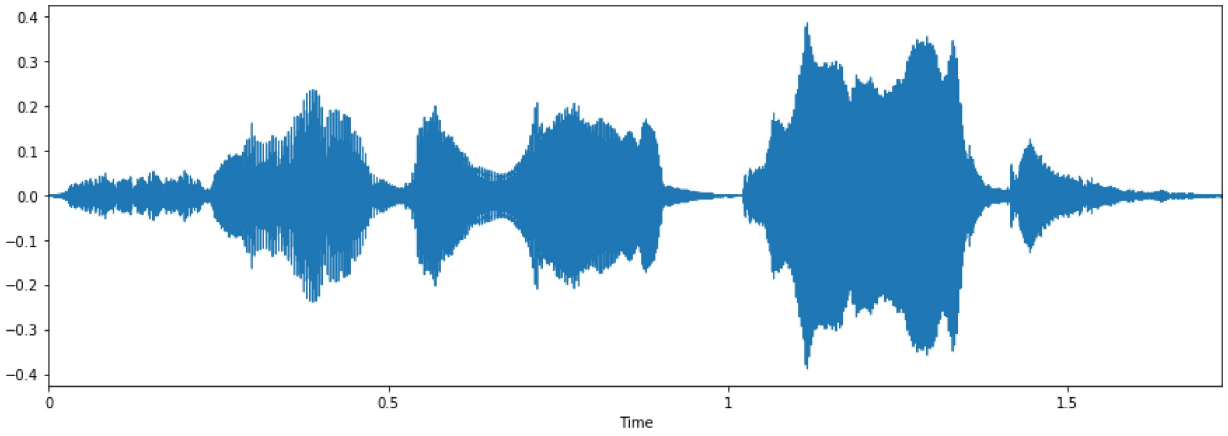
In [38]:
```
path = []
emotion = []

for i in dir_list:
    fname = os.listdir(TESS + i)
    for f in fname:
        if i == 'OAF_angry' or i == 'YAF_angry':
            emotion.append('female_angry')
        elif i == 'OAF_disgust' or i == 'YAF_disgust':
            emotion.append('female_disgust')
        elif i == 'OAF_Fear' or i == 'YAF_fear':
            emotion.append('female_fear')
        elif i == 'OAF_happy' or i == 'YAF_happy':
            emotion.append('female_happy')
        elif i == 'OAF_neutral' or i == 'YAF_neutral':
            emotion.append('female_neutral')
        elif i == 'OAF_Pleasant_surprise' or i == 'YAF_pleasant_surprised':
            emotion.append('female_surprise')
        elif i == 'OAF_Sad' or i == 'YAF_sad':
            emotion.append('female_sad')
        else:
            emotion.append('Unknown')
        path.append(TESS + i + "/" + f)

TESS_df = pd.DataFrame(emotion, columns = ['labels'])
TESS_df['source'] = 'TESS'
```

```
TESS_df = pd.concat([TESS_df,pd.DataFrame(path, columns = ['path'])],axis=1)
TESS_df.labels.value_counts()
```

Out[38]:
```
female_sad         400
female_fear        400
female_disgust     400
female_happy       400
female_angry       400
female_neutral     400
female_surprise    400
Name: labels, dtype: int64
```

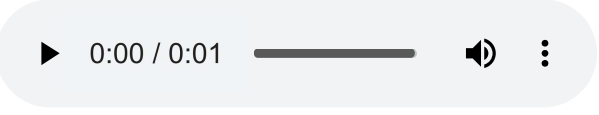
In [39]:
```
# lets play a fearful track
fname = TESS + 'YAF_fear/YAF_dog_fear.wav'

data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

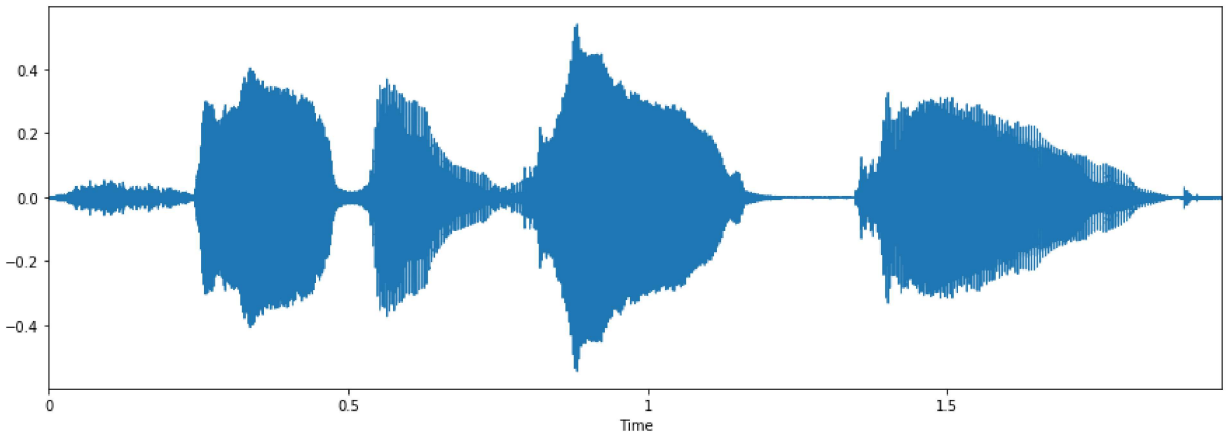
Out[39]:



In [40]:
```
# lets play a happy track
fname =  TESS + 'YAF_happy/YAF_dog_happy.wav'

data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

Out[40]:





In [41]:
```
dir_list = os.listdir(CREMA)
dir_list.sort()
print(dir_list[0:10])
```

```
['1001_DFA_ANG_XX.wav', '1001_DFA_DIS_XX.wav', '1001_DFA_FEA_XX.wav', '1001_DFA_HAP_X
X.wav', '1001_DFA_NEU_XX.wav', '1001_DFA_SAD_XX.wav', '1001_IEO_ANG_HI.wav', '1001_IE
O_ANG_LO.wav', '1001_IEO_ANG_MD.wav', '1001_IEO_DIS_HI.wav']
```

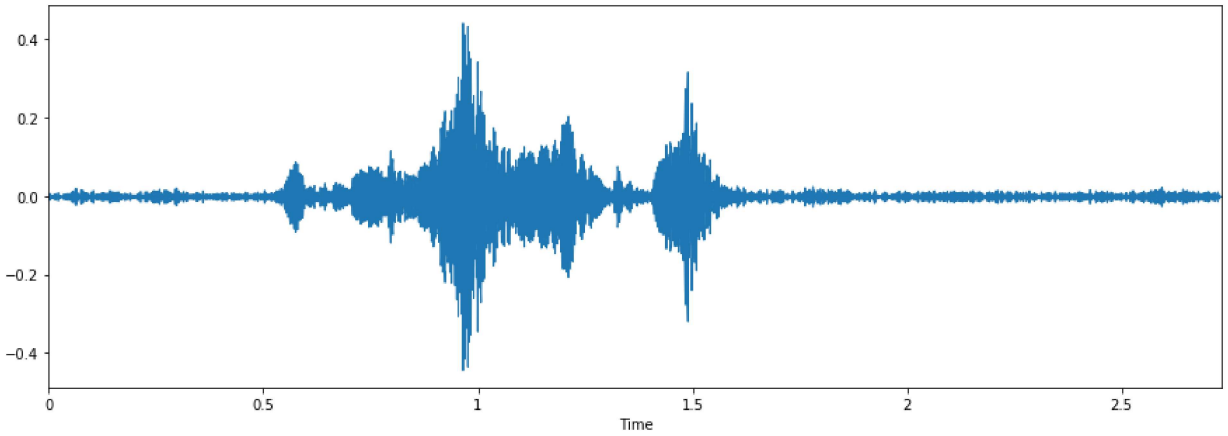
In [42]:
```
gender = []
emotion = []
path = []
female = [1002,1003,1004,1006,1007,1008,1009,1010,1012,1013,1018,1020,1021,1024,1025,1
          1052,1053,1054,1055,1056,1058,1060,1061,1063,1072,1073,1074,1075,1076,1078,1

for i in dir_list:
    part = i.split('_')
    if int(part[0]) in female:
        temp = 'female'
    else:
        temp = 'male'
    gender.append(temp)
    if part[2] == 'SAD' and temp == 'male':
        emotion.append('male_sad')
    elif part[2] == 'ANG' and temp == 'male':
        emotion.append('male_angry')
    elif part[2] == 'DIS' and temp == 'male':
        emotion.append('male_disgust')
    elif part[2] == 'FEA' and temp == 'male':
        emotion.append('male_fear')
    elif part[2] == 'HAP' and temp == 'male':
        emotion.append('male_happy')
    elif part[2] == 'NEU' and temp == 'male':
        emotion.append('male_neutral')
    elif part[2] == 'SAD' and temp == 'female':
        emotion.append('female_sad')
    elif part[2] == 'ANG' and temp == 'female':
        emotion.append('female_angry')
    elif part[2] == 'DIS' and temp == 'female':
        emotion.append('female_disgust')
    elif part[2] == 'FEA' and temp == 'female':
        emotion.append('female_fear')
    elif part[2] == 'HAP' and temp == 'female':
        emotion.append('female_happy')
    elif part[2] == 'NEU' and temp == 'female':
        emotion.append('female_neutral')
```

```
    else:
        emotion.append('Unknown')
    path.append(CREMA + i)

CREMA_df = pd.DataFrame(emotion, columns = ['labels'])
CREMA_df['source'] = 'CREMA'
CREMA_df = pd.concat([CREMA_df,pd.DataFrame(path, columns = ['path'])],axis=1)
CREMA_df.labels.value_counts()
```

Out[42]:
```
male_angry        671
male_disgust      671
male_fear         671
male_sad          671
male_happy        671
female_sad        600
female_fear       600
female_happy      600
female_disgust    600
female_angry      600
male_neutral      575
female_neutral    512
Name: labels, dtype: int64
```

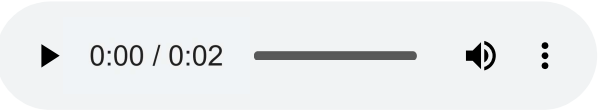
In [43]:
```
# use Librosa library for this task
fname = CREMA + '1012_IEO_HAP_HI.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

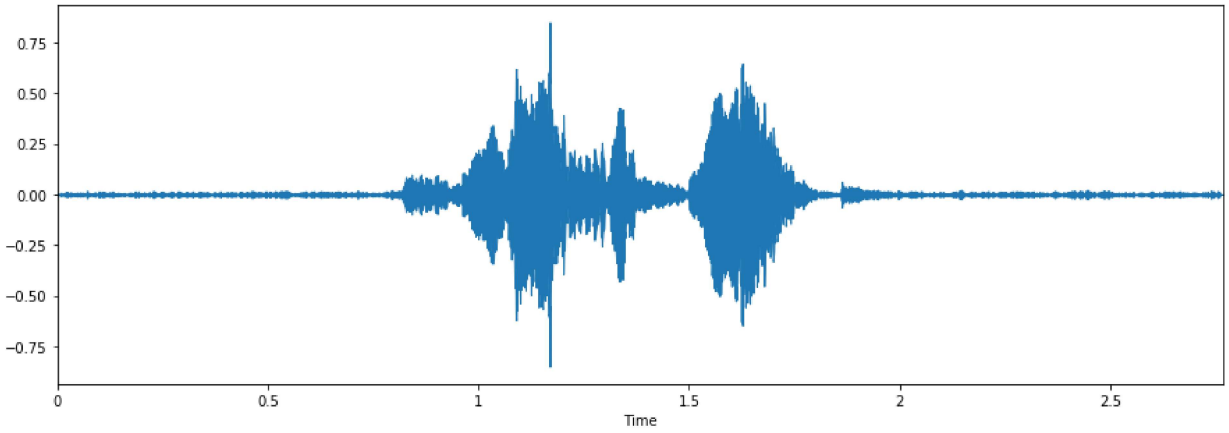
Out[43]:



In [44]:
```
# A fearful track
fname = CREMA + '1012_IEO_FEA_HI.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Lets play the audio
ipd.Audio(fname)
```

Out[44]:





In [45]:
```
df = pd.concat([SAVEE_df, RAV_df, TESS_df, CREMA_df], axis = 0)
print(df.labels.value_counts())
df.head()
df.to_csv("Data_path.csv",index=False)
```

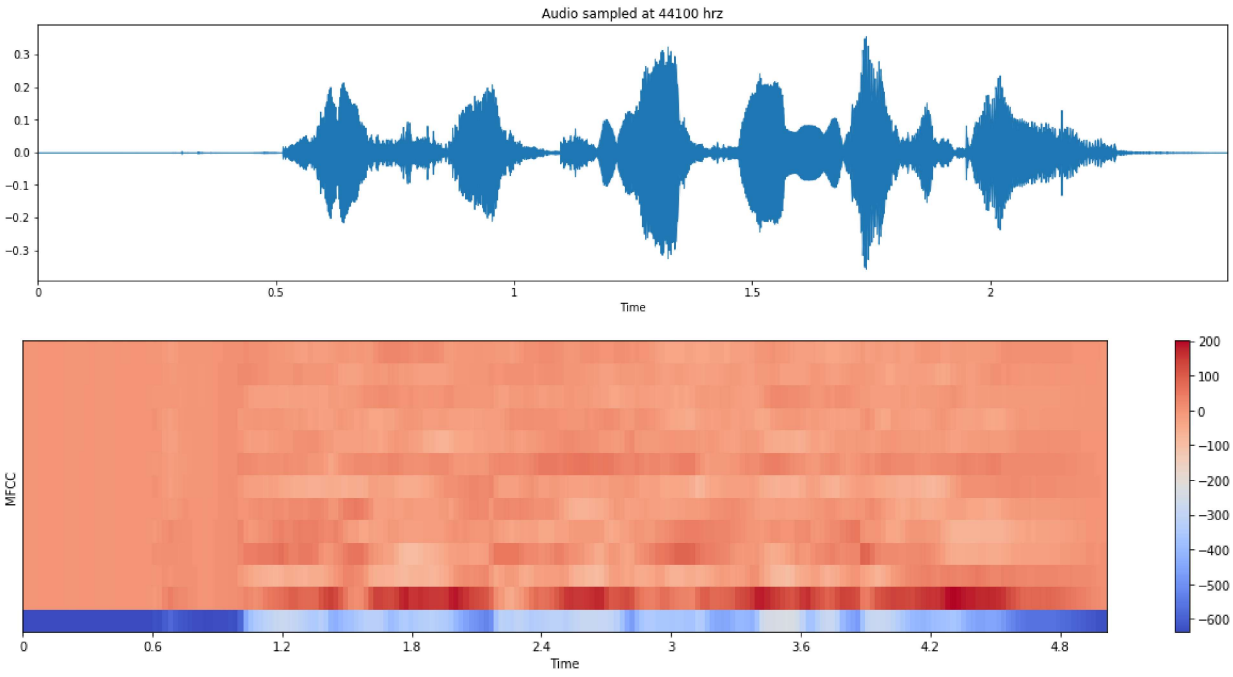
```
female_happy       1096
female_sad         1096
female_fear        1096
female_disgust     1096
female_angry       1096
female_neutral     1056
male_neutral        839
male_fear           827
male_angry          827
male_disgust        827
male_sad            827
male_happy          827
female_surprise     496
male_surprise       156
Name: labels, dtype: int64
```

In [46]:
```
# Import libraries
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import specgram
import pandas as pd
import os
import IPython.display as ipd  # To play sound in the notebook
```

In [47]:
```
# Source - RAVDESS; Gender - Female; Emotion - Angry
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_08/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
mfcc = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.waveplot(X, sr=sample_rate)
plt.title('Audio sampled at 44100 hrz')
```

```
# MFCC
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()

ipd.Audio(path)
```

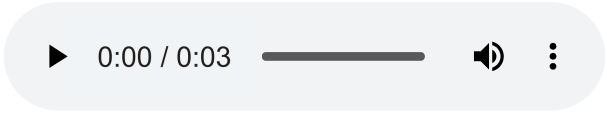
Out[47]:

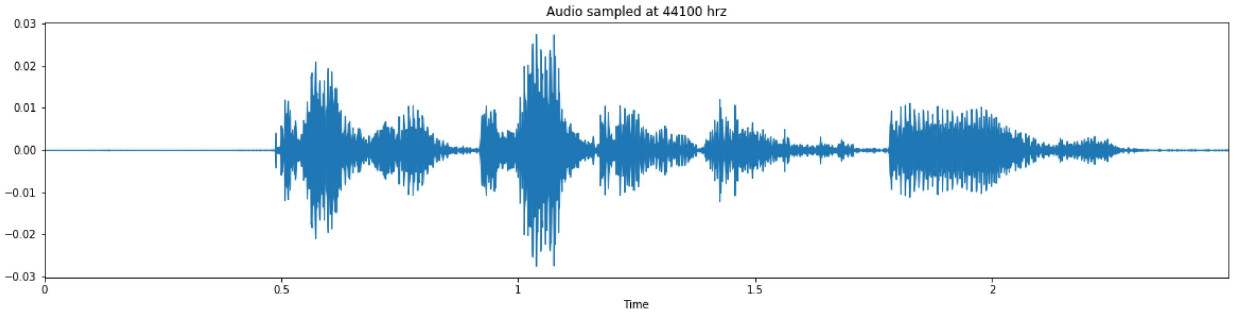


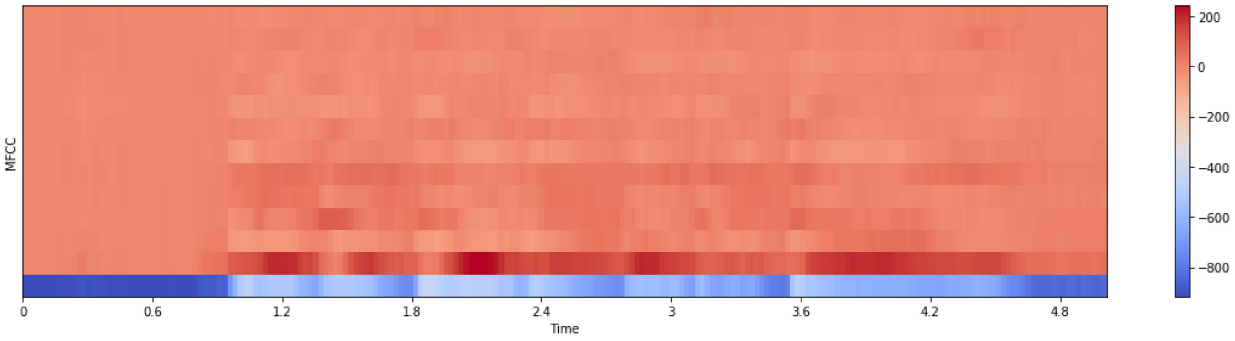
In [48]:

```
# Source - RAVDESS; Gender - Male; Emotion - Angry
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_09/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
mfcc = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.waveplot(X, sr=sample_rate)
plt.title('Audio sampled at 44100 hrz')

# MFCC
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()

ipd.Audio(path)
```

Out[48]:

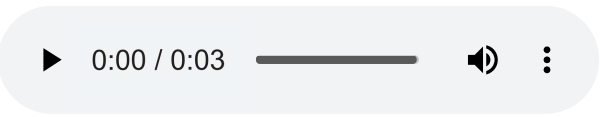


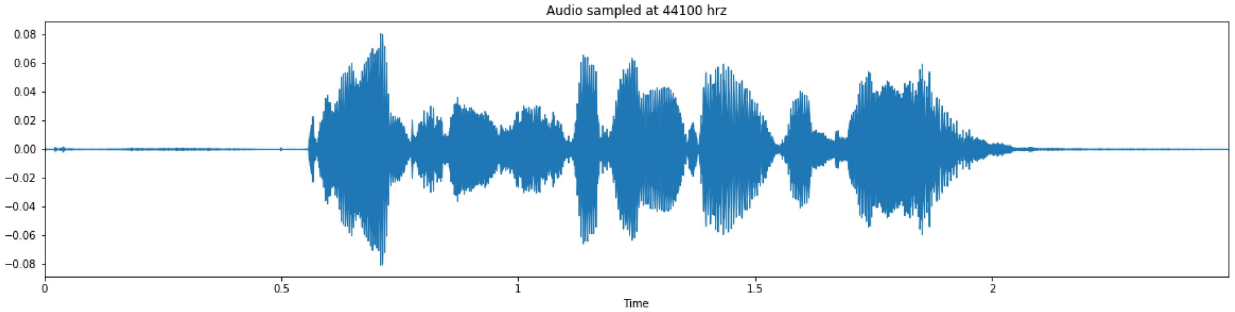


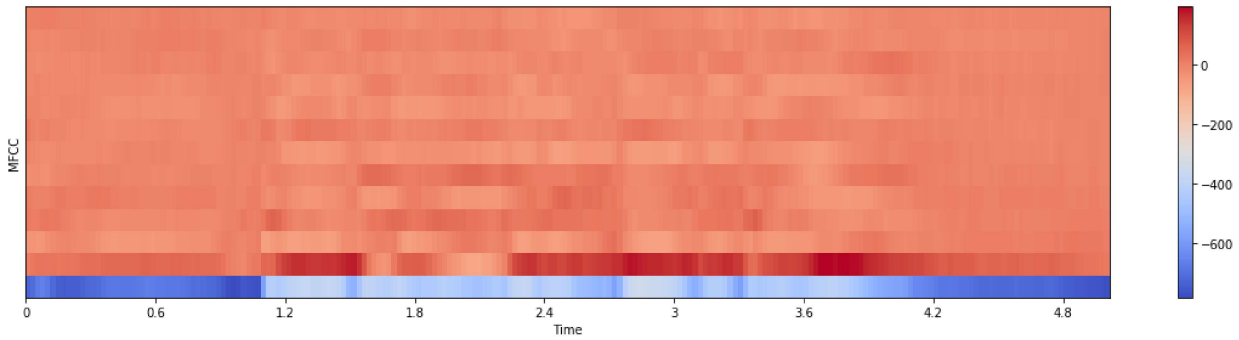


In [49]:
```
# Source - RAVDESS; Gender - Female; Emotion - Happy
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_12/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
mfcc = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.waveplot(X, sr=sample_rate)
plt.title('Audio sampled at 44100 hrz')

# MFCC
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()

ipd.Audio(path)
```

Out[49]:

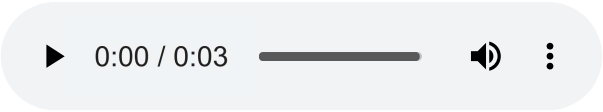


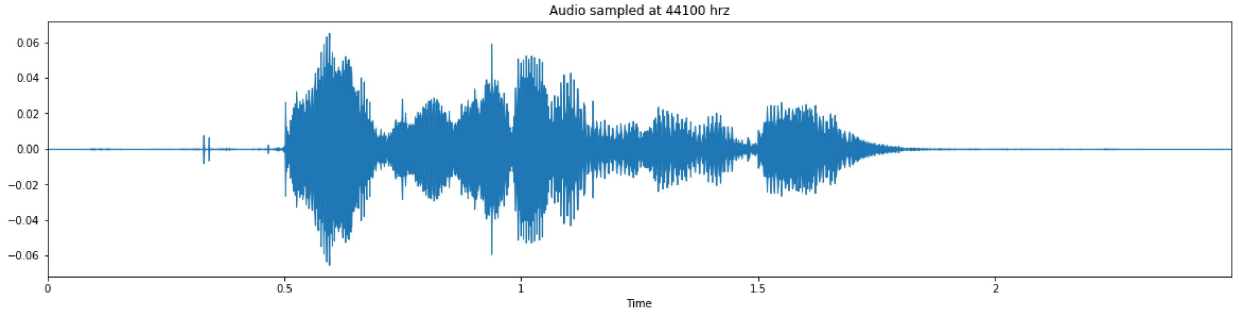


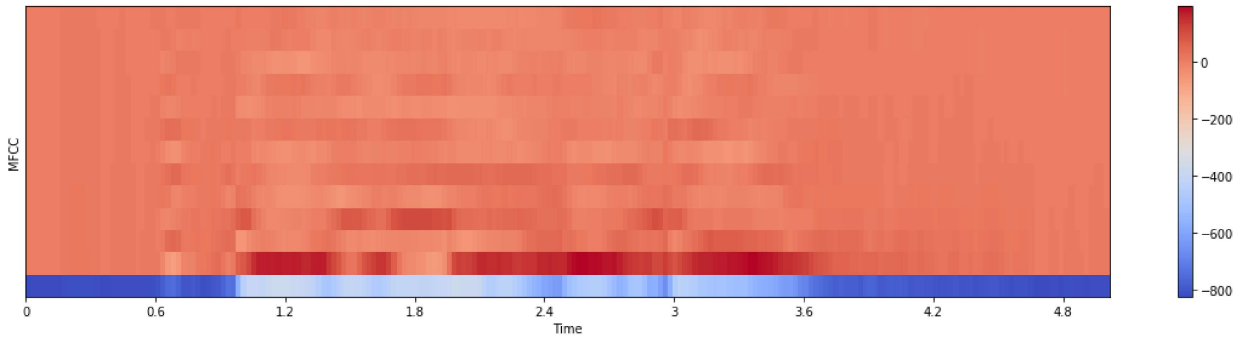


In [50]:
```
# Source - RAVDESS; Gender - Male; Emotion - Happy
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_11/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
mfcc = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.waveplot(X, sr=sample_rate)
plt.title('Audio sampled at 44100 hrz')

# MFCC
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()

ipd.Audio(path)
```

Out[50]:





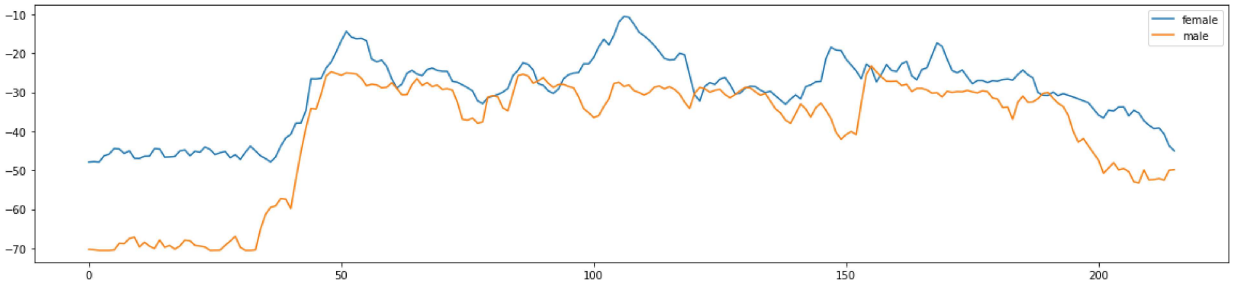


In [51]:
```
# Source - RAVDESS; Gender - Female; Emotion - Angry
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_08/03-01
```

```
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
female = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
female = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
print(len(female))

# Source - RAVDESS; Gender - Male; Emotion - Angry
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_09/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
male = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
male = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
print(len(male))

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
plt.plot(female, label='female')
plt.plot(male, label='male')
plt.legend()
```

```
216
216
```

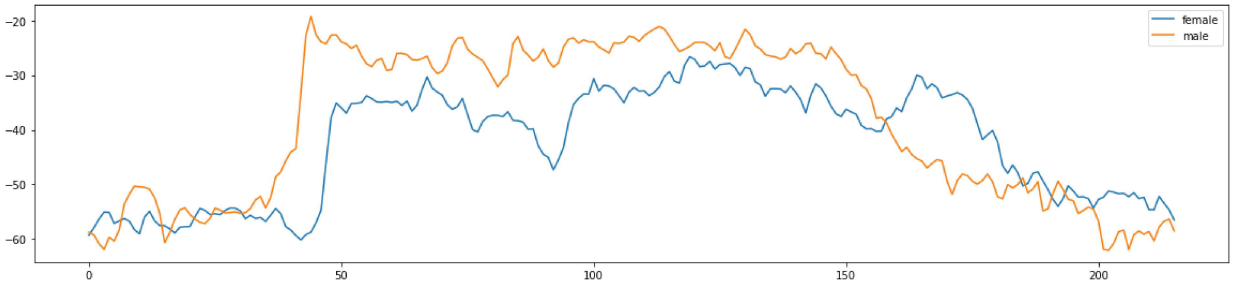
Out[51]:
```
<matplotlib.legend.Legend at 0x7ff7167c2828>
```



In [52]:
```
# Source - RAVDESS; Gender - Female; Emotion - happy
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_12/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
female = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
female = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
print(len(female))

# Source - RAVDESS; Gender - Male; Emotion - happy
path = "/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Actor_11/03-01
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,off
male = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
male = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
print(len(male))

# Plot the two audio waves together
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
plt.plot(female, label='female')
plt.plot(male, label='male')
plt.legend()
```

```
216
216
```

Out[52]:
```
<matplotlib.legend.Legend at 0x7ff7166d2978>
```



In [53]:
```
# Import libraries
# Keras
import keras
from keras import regularizers
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model, model_from_json
from keras.layers import Dense, Embedding, LSTM
from keras.layers import Input, Flatten, Dropout, Activation, BatchNormalization
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint

# sklearn
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Other
import librosa
import librosa.display
import json
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.pyplot import specgram
import pandas as pd
import seaborn as sns
import glob
import os
import pickle
import IPython.display as ipd  # To play sound in the notebook
```

In [ ]:
```
# lets pick up the meta-data that we got from our first part of the Kernel
ref = pd.read_csv("/input/speech-emotion-recognition-with-cnn/Data_path.csv")
```

In [55]:
```
# Note this takes a couple of minutes (~10 mins) as we're iterating over 4 datasets
df = pd.DataFrame(columns=['feature'])

# loop feature extraction over the entire dataset
counter=0
for index,path in enumerate(ref.path):
    X, sample_rate = librosa.load(path
                                  , res_type='kaiser_fast'
                                  ,duration=2.5
                                  ,sr=44100
                                  ,offset=0.5
                                 )
```

```
    sample_rate = np.array(sample_rate)

    # mean as the feature. Could do min and max etc as well.
    mfccs = np.mean(librosa.feature.mfcc(y=X,
                                         sr=sample_rate,
                                         n_mfcc=13),
                    axis=0)
    df.loc[counter] = [mfccs]
    counter=counter+1

# Check a few records to make sure its processed successfully
print(len(df))
df.head()
```

```
12162
```

Out[55]:

	feature
0	[-37.653667, -33.313198, -30.6239, -28.531654,...
1	[-22.694735, -24.921019, -26.885933, -24.67741...
2	[-23.571938, -23.896921, -23.477041, -22.94047...
3	[-42.00334, -41.782307, -42.463604, -32.360104...
4	[-24.375729, -25.236479, -25.401834, -25.77177...

In [56]:
```
# Now extract the mean bands to its own feature columns
df = pd.concat([ref,pd.DataFrame(df['feature'].values.tolist())],axis=1)
df[:5]
```

Out[56]:

	labels	source	path	0	1	2	3	
0	male_disgust	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-37.653667	-33.313198	-30.623899	-28.531654	-27.61735
1	male_neutral	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-22.694735	-24.921019	-26.885933	-24.677412	-23.99654
2	male_sad	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-23.571938	-23.896921	-23.477041	-22.940472	-23.12606
3	male_neutral	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-42.003342	-41.782307	-42.463604	-32.360104	-27.22429
4	male_surprise	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-24.375729	-25.236479	-25.401834	-25.771776	-24.43267

5 rows × 219 columns

In [57]:
```
# replace NA with 0
df=df.fillna(0)
print(df.shape)
df[:5]
```

(12162, 219)

Out[57]:

	labels	source	path	0	1	2	3	
0	male_disgust	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-37.653667	-33.313198	-30.623899	-28.531654	-27.61735
1	male_neutral	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-22.694735	-24.921019	-26.885933	-24.677412	-23.99654
2	male_sad	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-23.571938	-23.896921	-23.477041	-22.940472	-23.12606
3	male_neutral	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-42.003342	-41.782307	-42.463604	-32.360104	-27.22429
4	male_surprise	SAVEE	/kaggle/input/surrey-audiovisual-expressed-emo...	-24.375729	-25.236479	-25.401834	-25.771776	-24.43267

5 rows × 219 columns

In [58]:

```
# Split between train and test
X_train, X_test, y_train, y_test = train_test_split(df.drop(['path','labels','source']
                                                    , df.labels
                                                    , test_size=0.25
                                                    , shuffle=True
                                                    , random_state=42
                                                    )

# Lets see how the data present itself before normalisation
X_train[150:160]
```

Out[58]:

	0	1	2	3	4	5	6	7
4950	-18.611179	-17.616539	-18.411484	-18.987419	-17.404621	-16.747272	-17.733747	-18.055025
3860	-10.208663	-14.527320	-25.505571	-25.095884	-26.218971	-23.725719	-22.898743	-26.671621
9761	-1.533947	-4.030602	-9.614023	-12.045173	-9.992992	-11.926250	-14.008465	-13.561555
7620	-4.531077	-3.933792	-4.567834	-5.871509	-5.282475	-6.490459	-8.156466	-9.188803
11586	-20.621702	-21.587507	-20.563646	-20.703459	-21.205715	-18.608534	-18.446669	-16.211845
7914	-17.514988	-18.551867	-17.043016	-16.977907	-19.369633	-19.562126	-22.008749	-20.178385
9513	-18.740368	-18.824930	-16.149488	-16.963457	-18.229979	-18.183954	-19.274342	-18.395123
5835	-19.066849	-18.328381	-17.710285	-18.043194	-18.252480	-18.710625	-16.626354	-17.831005
5389	-20.760590	-20.047138	-18.961346	-19.468687	-19.316292	-18.162563	-18.102333	-19.914133
11222	-18.252924	-17.727373	-19.222475	-18.469971	-17.572325	-17.850542	-17.932026	-20.588900

10 rows × 216 columns

In [59]:
```
# Lts do data normalization
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std

# Check the dataset now
X_train[150:160]
```

Out[59]:

	0	1	2	3	4	5	6	7	8
4950	0.183461	0.300479	0.435056	0.385458	0.495202	0.538995	0.458129	0.429506	0.432916
3860	0.771710	0.525487	-0.092859	-0.067769	-0.157619	0.022494	0.076773	-0.205939	-0.224307
9761	1.379016	1.290030	1.089730	0.900549	1.044132	0.895817	0.733184	0.760884	0.720953
7620	1.169191	1.297081	1.465248	1.358613	1.393009	1.298140	1.165265	1.083359	1.114880
11586	0.042707	0.011248	0.274900	0.258134	0.213680	0.401236	0.405491	0.565434	0.703767
7914	0.260204	0.232353	0.536892	0.534557	0.349667	0.330657	0.142485	0.272915	0.428611
9513	0.174417	0.212464	0.603385	0.535629	0.434073	0.432661	0.344380	0.404425	0.505166
5835	0.151560	0.248631	0.487236	0.455516	0.432407	0.393680	0.539893	0.446026	0.425754
5389	0.032984	0.123443	0.394137	0.349750	0.353617	0.434244	0.430915	0.292403	0.211713
11222	0.208542	0.292406	0.374705	0.423851	0.482781	0.457338	0.443489	0.242641	0.382708

10 rows × 216 columns

In [60]:
```
# Lets few preparation steps to get it into the correct format for Keras
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

# one hot encode the target
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

print(X_train.shape)
print(lb.classes_)
#print(y_train[0:10])
#print(y_test[0:10])

# Pick the lb object for future use
filename = 'labels'
outfile = open(filename,'wb')
pickle.dump(lb,outfile)
outfile.close()
```

```
(9121, 216)
['female_angry' 'female_disgust' 'female_fear' 'female_happy'
 'female_neutral' 'female_sad' 'female_surprise' 'male_angry'
 'male_disgust' 'male_fear' 'male_happy' 'male_neutral' 'male_sad'
 'male_surprise']
```

In [61]:
```
X_train = np.expand_dims(X_train, axis=2)
X_test = np.expand_dims(X_test, axis=2)
X_train.shape
```

Out[61]:
```
(9121, 216, 1)
```

In [62]:
```
# New model
model = Sequential()
model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1)))  # X_train.
model.add(Activation('relu'))
model.add(Conv1D(256, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(14)) # Target class number
model.add(Activation('softmax'))
# opt = keras.optimizers.SGD(lr=0.0001, momentum=0.0, decay=0.0, nesterov=False)
# opt = keras.optimizers.Adam(lr=0.0001)
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 216, 256)	2304
activation_1 (Activation)	(None, 216, 256)	0
conv1d_2 (Conv1D)	(None, 216, 256)	524544
batch_normalization_1 (Batch	(None, 216, 256)	1024
activation_2 (Activation)	(None, 216, 256)	0
dropout_1 (Dropout)	(None, 216, 256)	0
max_pooling1d_1 (MaxPooling1	(None, 27, 256)	0
conv1d_3 (Conv1D)	(None, 27, 128)	262272
activation_3 (Activation)	(None, 27, 128)	0
conv1d_4 (Conv1D)	(None, 27, 128)	131200
activation_4 (Activation)	(None, 27, 128)	0
conv1d_5 (Conv1D)	(None, 27, 128)	131200
activation_5 (Activation)	(None, 27, 128)	0
conv1d_6 (Conv1D)	(None, 27, 128)	131200
batch_normalization_2 (Batch	(None, 27, 128)	512
activation_6 (Activation)	(None, 27, 128)	0
dropout_2 (Dropout)	(None, 27, 128)	0
max_pooling1d_2 (MaxPooling1	(None, 3, 128)	0
conv1d_7 (Conv1D)	(None, 3, 64)	65600
activation_7 (Activation)	(None, 3, 64)	0
conv1d_8 (Conv1D)	(None, 3, 64)	32832
activation_8 (Activation)	(None, 3, 64)	0
flatten_1 (Flatten)	(None, 192)	0
dense_1 (Dense)	(None, 14)	2702
activation_9 (Activation)	(None, 14)	0

Total params: 1,285,390
Trainable params: 1,284,622
Non-trainable params: 768

In [63]:
```
model.compile(loss='categorical_crossentropy', optimizer=opt,metrics=['accuracy'])
model_history=model.fit(X_train, y_train, batch_size=16, epochs=100, validation_data=(
```

```
Train on 9121 samples, validate on 3041 samples
Epoch 1/100
9121/9121 [==============================] - 10s 1ms/step - loss: 2.4389 - accuracy:
0.1929 - val_loss: 2.4424 - val_accuracy: 0.2243
Epoch 2/100
9121/9121 [==============================] - 6s 656us/step - loss: 2.2120 - accuracy:
0.2472 - val_loss: 2.2493 - val_accuracy: 0.2779
Epoch 3/100
9121/9121 [==============================] - 6s 661us/step - loss: 2.1085 - accuracy:
0.2777 - val_loss: 2.1622 - val_accuracy: 0.2844
Epoch 4/100
9121/9121 [==============================] - 6s 655us/step - loss: 2.0297 - accuracy:
0.3053 - val_loss: 2.1090 - val_accuracy: 0.3012
Epoch 5/100
9121/9121 [==============================] - 6s 660us/step - loss: 1.9741 - accuracy:
0.3230 - val_loss: 2.0538 - val_accuracy: 0.3259
Epoch 6/100
9121/9121 [==============================] - 6s 692us/step - loss: 1.9235 - accuracy:
0.3436 - val_loss: 2.0239 - val_accuracy: 0.3479
Epoch 7/100
9121/9121 [==============================] - 6s 654us/step - loss: 1.8794 - accuracy:
0.3566 - val_loss: 1.9690 - val_accuracy: 0.3624
Epoch 8/100
9121/9121 [==============================] - 6s 666us/step - loss: 1.8419 - accuracy:
0.3710 - val_loss: 1.9435 - val_accuracy: 0.3801
Epoch 9/100
9121/9121 [==============================] - 6s 667us/step - loss: 1.8076 - accuracy:
0.3804 - val_loss: 1.9066 - val_accuracy: 0.3936
Epoch 10/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.7762 - accuracy:
0.3859 - val_loss: 1.9008 - val_accuracy: 0.3897
Epoch 11/100
9121/9121 [==============================] - 6s 666us/step - loss: 1.7546 - accuracy:
0.3952 - val_loss: 1.8662 - val_accuracy: 0.3969
Epoch 12/100
9121/9121 [==============================] - 6s 667us/step - loss: 1.7315 - accuracy:
0.4042 - val_loss: 1.8526 - val_accuracy: 0.3979
Epoch 13/100
9121/9121 [==============================] - 6s 657us/step - loss: 1.7028 - accuracy:
0.4159 - val_loss: 1.8180 - val_accuracy: 0.4127
Epoch 14/100
9121/9121 [==============================] - 6s 652us/step - loss: 1.6856 - accuracy:
0.4176 - val_loss: 1.8100 - val_accuracy: 0.4166
Epoch 15/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.6643 - accuracy:
0.4207 - val_loss: 1.8007 - val_accuracy: 0.4239
Epoch 16/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.6494 - accuracy:
0.4333 - val_loss: 1.7915 - val_accuracy: 0.4071
Epoch 17/100
9121/9121 [==============================] - 6s 674us/step - loss: 1.6315 - accuracy:
0.4426 - val_loss: 1.7658 - val_accuracy: 0.4235
Epoch 18/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.6192 - accuracy:
0.4368 - val_loss: 1.7581 - val_accuracy: 0.4268
Epoch 19/100
9121/9121 [==============================] - 6s 654us/step - loss: 1.6030 - accuracy:
0.4462 - val_loss: 1.7483 - val_accuracy: 0.4318
Epoch 20/100
9121/9121 [==============================] - 6s 657us/step - loss: 1.5949 - accuracy:
```

```
0.4489 - val_loss: 1.7508 - val_accuracy: 0.4199
Epoch 21/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.5705 - accuracy:
0.4565 - val_loss: 1.7232 - val_accuracy: 0.4288
Epoch 22/100
9121/9121 [==============================] - 6s 668us/step - loss: 1.5670 - accuracy:
0.4608 - val_loss: 1.7365 - val_accuracy: 0.4232
Epoch 23/100
9121/9121 [==============================] - 6s 668us/step - loss: 1.5560 - accuracy:
0.4635 - val_loss: 1.7317 - val_accuracy: 0.4193
Epoch 24/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.5458 - accuracy:
0.4682 - val_loss: 1.7117 - val_accuracy: 0.4324
Epoch 25/100
9121/9121 [==============================] - 6s 655us/step - loss: 1.5323 - accuracy:
0.4708 - val_loss: 1.6877 - val_accuracy: 0.4449
Epoch 26/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.5246 - accuracy:
0.4739 - val_loss: 1.6912 - val_accuracy: 0.4305
Epoch 27/100
9121/9121 [==============================] - 6s 674us/step - loss: 1.5119 - accuracy:
0.4762 - val_loss: 1.6951 - val_accuracy: 0.4357
Epoch 28/100
9121/9121 [==============================] - 6s 657us/step - loss: 1.4981 - accuracy:
0.4801 - val_loss: 1.6692 - val_accuracy: 0.4456
Epoch 29/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.4837 - accuracy:
0.4866 - val_loss: 1.6739 - val_accuracy: 0.4377
Epoch 30/100
9121/9121 [==============================] - 6s 653us/step - loss: 1.4819 - accuracy:
0.4896 - val_loss: 1.6510 - val_accuracy: 0.4541
Epoch 31/100
9121/9121 [==============================] - 6s 655us/step - loss: 1.4687 - accuracy:
0.4991 - val_loss: 1.6730 - val_accuracy: 0.4377
Epoch 32/100
9121/9121 [==============================] - 6s 668us/step - loss: 1.4596 - accuracy:
0.4953 - val_loss: 1.6618 - val_accuracy: 0.4403
Epoch 33/100
9121/9121 [==============================] - 6s 661us/step - loss: 1.4485 - accuracy:
0.5020 - val_loss: 1.6481 - val_accuracy: 0.4328
Epoch 34/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.4434 - accuracy:
0.5069 - val_loss: 1.6427 - val_accuracy: 0.4571
Epoch 35/100
9121/9121 [==============================] - 6s 653us/step - loss: 1.4303 - accuracy:
0.5043 - val_loss: 1.6586 - val_accuracy: 0.4459
Epoch 36/100
9121/9121 [==============================] - 6s 655us/step - loss: 1.4192 - accuracy:
0.5117 - val_loss: 1.6369 - val_accuracy: 0.4561
Epoch 37/100
9121/9121 [==============================] - 6s 665us/step - loss: 1.4127 - accuracy:
0.5178 - val_loss: 1.6333 - val_accuracy: 0.4515
Epoch 38/100
9121/9121 [==============================] - 6s 664us/step - loss: 1.4063 - accuracy:
0.5151 - val_loss: 1.6542 - val_accuracy: 0.4410
Epoch 39/100
9121/9121 [==============================] - 6s 655us/step - loss: 1.4061 - accuracy:
0.5164 - val_loss: 1.6640 - val_accuracy: 0.4413
Epoch 40/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.3911 - accuracy:
```

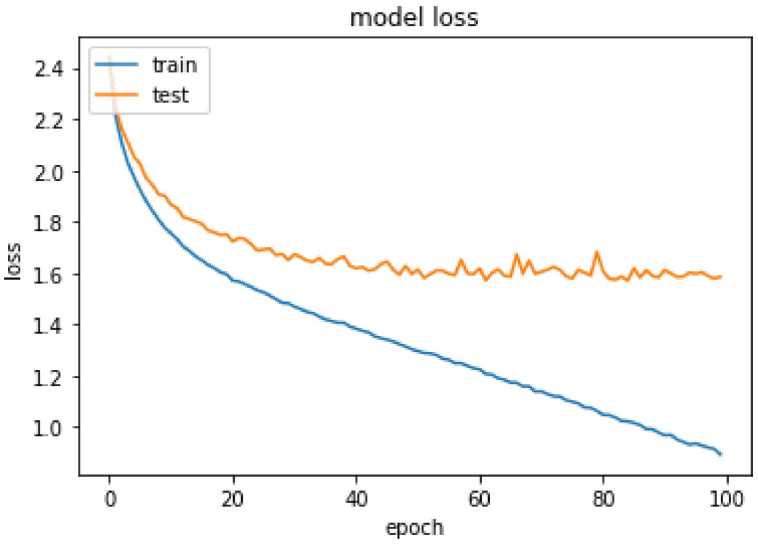
```
0.5168 - val_loss: 1.6271 - val_accuracy: 0.4492
Epoch 41/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.3829 - accuracy:
0.5234 - val_loss: 1.6180 - val_accuracy: 0.4577
Epoch 42/100
9121/9121 [==============================] - 6s 652us/step - loss: 1.3745 - accuracy:
0.5291 - val_loss: 1.6233 - val_accuracy: 0.4469
Epoch 43/100
9121/9121 [==============================] - 6s 675us/step - loss: 1.3684 - accuracy:
0.5292 - val_loss: 1.6084 - val_accuracy: 0.4551
Epoch 44/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.3533 - accuracy:
0.5327 - val_loss: 1.6137 - val_accuracy: 0.4568
Epoch 45/100
9121/9121 [==============================] - 6s 652us/step - loss: 1.3456 - accuracy:
0.5373 - val_loss: 1.6342 - val_accuracy: 0.4449
Epoch 46/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.3399 - accuracy:
0.5372 - val_loss: 1.6448 - val_accuracy: 0.4311
Epoch 47/100
9121/9121 [==============================] - 6s 653us/step - loss: 1.3329 - accuracy:
0.5384 - val_loss: 1.6121 - val_accuracy: 0.4420
Epoch 48/100
9121/9121 [==============================] - 6s 677us/step - loss: 1.3226 - accuracy:
0.5493 - val_loss: 1.5934 - val_accuracy: 0.4541
Epoch 49/100
9121/9121 [==============================] - 6s 654us/step - loss: 1.3136 - accuracy:
0.5493 - val_loss: 1.6276 - val_accuracy: 0.4397
Epoch 50/100
9121/9121 [==============================] - 6s 651us/step - loss: 1.3025 - accuracy:
0.5581 - val_loss: 1.5958 - val_accuracy: 0.4640
Epoch 51/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.2951 - accuracy:
0.5563 - val_loss: 1.6129 - val_accuracy: 0.4564
Epoch 52/100
9121/9121 [==============================] - 6s 653us/step - loss: 1.2882 - accuracy:
0.5610 - val_loss: 1.5813 - val_accuracy: 0.4630
Epoch 53/100
9121/9121 [==============================] - 6s 669us/step - loss: 1.2864 - accuracy:
0.5643 - val_loss: 1.5949 - val_accuracy: 0.4528
Epoch 54/100
9121/9121 [==============================] - 6s 662us/step - loss: 1.2802 - accuracy:
0.5558 - val_loss: 1.6095 - val_accuracy: 0.4397
Epoch 55/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.2656 - accuracy:
0.5656 - val_loss: 1.6084 - val_accuracy: 0.4479
Epoch 56/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.2615 - accuracy:
0.5628 - val_loss: 1.5971 - val_accuracy: 0.4538
Epoch 57/100
9121/9121 [==============================] - 6s 654us/step - loss: 1.2484 - accuracy:
0.5744 - val_loss: 1.5915 - val_accuracy: 0.4623
Epoch 58/100
9121/9121 [==============================] - 6s 670us/step - loss: 1.2484 - accuracy:
0.5727 - val_loss: 1.6510 - val_accuracy: 0.4262
Epoch 59/100
9121/9121 [==============================] - 6s 661us/step - loss: 1.2383 - accuracy:
0.5812 - val_loss: 1.5985 - val_accuracy: 0.4535
Epoch 60/100
9121/9121 [==============================] - 6s 663us/step - loss: 1.2291 - accuracy:
```

```
0.5825 - val_loss: 1.5955 - val_accuracy: 0.4541
Epoch 61/100
9121/9121 [==============================] - 6s 662us/step - loss: 1.2246 - accuracy:
0.5838 - val_loss: 1.6175 - val_accuracy: 0.4403
Epoch 62/100
9121/9121 [==============================] - 6s 652us/step - loss: 1.2063 - accuracy:
0.5898 - val_loss: 1.5713 - val_accuracy: 0.4538
Epoch 63/100
9121/9121 [==============================] - 6s 667us/step - loss: 1.2040 - accuracy:
0.5926 - val_loss: 1.6001 - val_accuracy: 0.4462
Epoch 64/100
9121/9121 [==============================] - 6s 666us/step - loss: 1.1895 - accuracy:
0.5927 - val_loss: 1.6139 - val_accuracy: 0.4400
Epoch 65/100
9121/9121 [==============================] - 6s 653us/step - loss: 1.1838 - accuracy:
0.6019 - val_loss: 1.5896 - val_accuracy: 0.4545
Epoch 66/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.1728 - accuracy:
0.6012 - val_loss: 1.5854 - val_accuracy: 0.4439
Epoch 67/100
9121/9121 [==============================] - 6s 654us/step - loss: 1.1721 - accuracy:
0.6004 - val_loss: 1.6731 - val_accuracy: 0.4058
Epoch 68/100
9121/9121 [==============================] - 6s 660us/step - loss: 1.1578 - accuracy:
0.6107 - val_loss: 1.5972 - val_accuracy: 0.4495
Epoch 69/100
9121/9121 [==============================] - 6s 670us/step - loss: 1.1580 - accuracy:
0.6103 - val_loss: 1.6480 - val_accuracy: 0.4301
Epoch 70/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.1378 - accuracy:
0.6152 - val_loss: 1.5966 - val_accuracy: 0.4413
Epoch 71/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.1388 - accuracy:
0.6173 - val_loss: 1.6049 - val_accuracy: 0.4318
Epoch 72/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.1276 - accuracy:
0.6222 - val_loss: 1.6126 - val_accuracy: 0.4308
Epoch 73/100
9121/9121 [==============================] - 6s 657us/step - loss: 1.1196 - accuracy:
0.6280 - val_loss: 1.6241 - val_accuracy: 0.4337
Epoch 74/100
9121/9121 [==============================] - 6s 678us/step - loss: 1.1181 - accuracy:
0.6196 - val_loss: 1.6120 - val_accuracy: 0.4337
Epoch 75/100
9121/9121 [==============================] - 6s 658us/step - loss: 1.1044 - accuracy:
0.6253 - val_loss: 1.5880 - val_accuracy: 0.4485
Epoch 76/100
9121/9121 [==============================] - 6s 673us/step - loss: 1.0979 - accuracy:
0.6267 - val_loss: 1.5780 - val_accuracy: 0.4420
Epoch 77/100
9121/9121 [==============================] - 6s 653us/step - loss: 1.0912 - accuracy:
0.6360 - val_loss: 1.6126 - val_accuracy: 0.4380
Epoch 78/100
9121/9121 [==============================] - 6s 655us/step - loss: 1.0761 - accuracy:
0.6402 - val_loss: 1.6005 - val_accuracy: 0.4387
Epoch 79/100
9121/9121 [==============================] - 6s 671us/step - loss: 1.0745 - accuracy:
0.6330 - val_loss: 1.5903 - val_accuracy: 0.4305
Epoch 80/100
9121/9121 [==============================] - 6s 660us/step - loss: 1.0620 - accuracy:
```

```
0.6459 - val_loss: 1.6823 - val_accuracy: 0.4186
Epoch 81/100
9121/9121 [==============================] - 6s 659us/step - loss: 1.0471 - accuracy:
0.6521 - val_loss: 1.6064 - val_accuracy: 0.4479
Epoch 82/100
9121/9121 [==============================] - 6s 655us/step - loss: 1.0460 - accuracy:
0.6492 - val_loss: 1.5778 - val_accuracy: 0.4446
Epoch 83/100
9121/9121 [==============================] - 6s 656us/step - loss: 1.0365 - accuracy:
0.6546 - val_loss: 1.5741 - val_accuracy: 0.4429
Epoch 84/100
9121/9121 [==============================] - 6s 674us/step - loss: 1.0232 - accuracy:
0.6591 - val_loss: 1.5864 - val_accuracy: 0.4397
Epoch 85/100
9121/9121 [==============================] - 6s 660us/step - loss: 1.0215 - accuracy:
0.6554 - val_loss: 1.5703 - val_accuracy: 0.4597
Epoch 86/100
9121/9121 [==============================] - 6s 660us/step - loss: 1.0155 - accuracy:
0.6603 - val_loss: 1.6187 - val_accuracy: 0.4397
Epoch 87/100
9121/9121 [==============================] - 6s 662us/step - loss: 1.0073 - accuracy:
0.6619 - val_loss: 1.5828 - val_accuracy: 0.4472
Epoch 88/100
9121/9121 [==============================] - 6s 658us/step - loss: 0.9917 - accuracy:
0.6715 - val_loss: 1.6110 - val_accuracy: 0.4370
Epoch 89/100
9121/9121 [==============================] - 6s 674us/step - loss: 0.9906 - accuracy:
0.6690 - val_loss: 1.5889 - val_accuracy: 0.4426
Epoch 90/100
9121/9121 [==============================] - 6s 651us/step - loss: 0.9778 - accuracy:
0.6764 - val_loss: 1.5827 - val_accuracy: 0.4439
Epoch 91/100
9121/9121 [==============================] - 6s 659us/step - loss: 0.9672 - accuracy:
0.6740 - val_loss: 1.6120 - val_accuracy: 0.4331
Epoch 92/100
9121/9121 [==============================] - 6s 655us/step - loss: 0.9683 - accuracy:
0.6733 - val_loss: 1.5976 - val_accuracy: 0.4452
Epoch 93/100
9121/9121 [==============================] - 6s 662us/step - loss: 0.9505 - accuracy:
0.6891 - val_loss: 1.5848 - val_accuracy: 0.4462
Epoch 94/100
9121/9121 [==============================] - 6s 667us/step - loss: 0.9413 - accuracy:
0.6915 - val_loss: 1.5860 - val_accuracy: 0.4370
Epoch 95/100
9121/9121 [==============================] - 6s 657us/step - loss: 0.9310 - accuracy:
0.6933 - val_loss: 1.6010 - val_accuracy: 0.4400
Epoch 96/100
9121/9121 [==============================] - 6s 658us/step - loss: 0.9352 - accuracy:
0.6855 - val_loss: 1.5966 - val_accuracy: 0.4459
Epoch 97/100
9121/9121 [==============================] - 6s 658us/step - loss: 0.9271 - accuracy:
0.6912 - val_loss: 1.6030 - val_accuracy: 0.4426
Epoch 98/100
9121/9121 [==============================] - 6s 657us/step - loss: 0.9196 - accuracy:
0.6970 - val_loss: 1.5883 - val_accuracy: 0.4446
Epoch 99/100
9121/9121 [==============================] - 6s 657us/step - loss: 0.9139 - accuracy:
0.6949 - val_loss: 1.5792 - val_accuracy: 0.4426
Epoch 100/100
```

```
9121/9121 [==============================] - 6s 670us/step - loss: 0.8930 - accuracy:
0.7032 - val_loss: 1.5847 - val_accuracy: 0.4462
```

In [64]:
```
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [ ]:
```
# Save model and weights
model_name = 'Emotion_Model.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')

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('Save model and weights at %s ' % model_path)

# Save the model to disk
model_json = model.to_json()
with open("model_json.json", "w") as json_file:
    json_file.write(model_json)
```

In [66]:
```
# loading json and model architecture
json_file = open('model_json.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/Emotion_Model.h5")
print("Loaded model from disk")

# Keras optimiser
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accurac
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: 44.62%
```

In [67]:
```
preds = loaded_model.predict(X_test,
                         batch_size=16,
                         verbose=1)

preds=preds.argmax(axis=1)
preds
```

```
3041/3041 [==============================] - 1s 237us/step
```

Out[67]:
```
array([2, 3, 0, ..., 8, 5, 0])
```

In [68]:
```
# predictions
preds = preds.astype(int).flatten()
preds = (lb.inverse_transform((preds)))
preds = pd.DataFrame({'predictedvalues': preds})

# Actual labels
actual=y_test.argmax(axis=1)
actual = actual.astype(int).flatten()
actual = (lb.inverse_transform((actual)))
actual = pd.DataFrame({'actualvalues': actual})

# Lets combined both of them into a single dataframe
finaldf = actual.join(preds)
finaldf[170:180]
```

Out[68]:

	actualvalues	predictedvalues
170	male_sad	female_disgust
171	female_angry	female_fear
172	male_angry	female_angry
173	female_disgust	female_disgust
174	male_angry	male_angry
175	female_fear	female_happy
176	male_happy	male_surprise
177	female_fear	female_happy
178	female_happy	female_happy
179	female_neutral	female_neutral

In [69]:
```
# Write out the predictions to disk
finaldf.to_csv('Predictions.csv', index=False)
finaldf.groupby('predictedvalues').count()
```

Out[69]:

predictedvalues	actualvalues
female_angry	313
female_disgust	372
female_fear	228
female_happy	378
female_neutral	188
female_sad	441
female_surprise	100
male_angry	176
male_disgust	144
male_fear	254
male_happy	89
male_neutral	201
male_sad	141
male_surprise	16

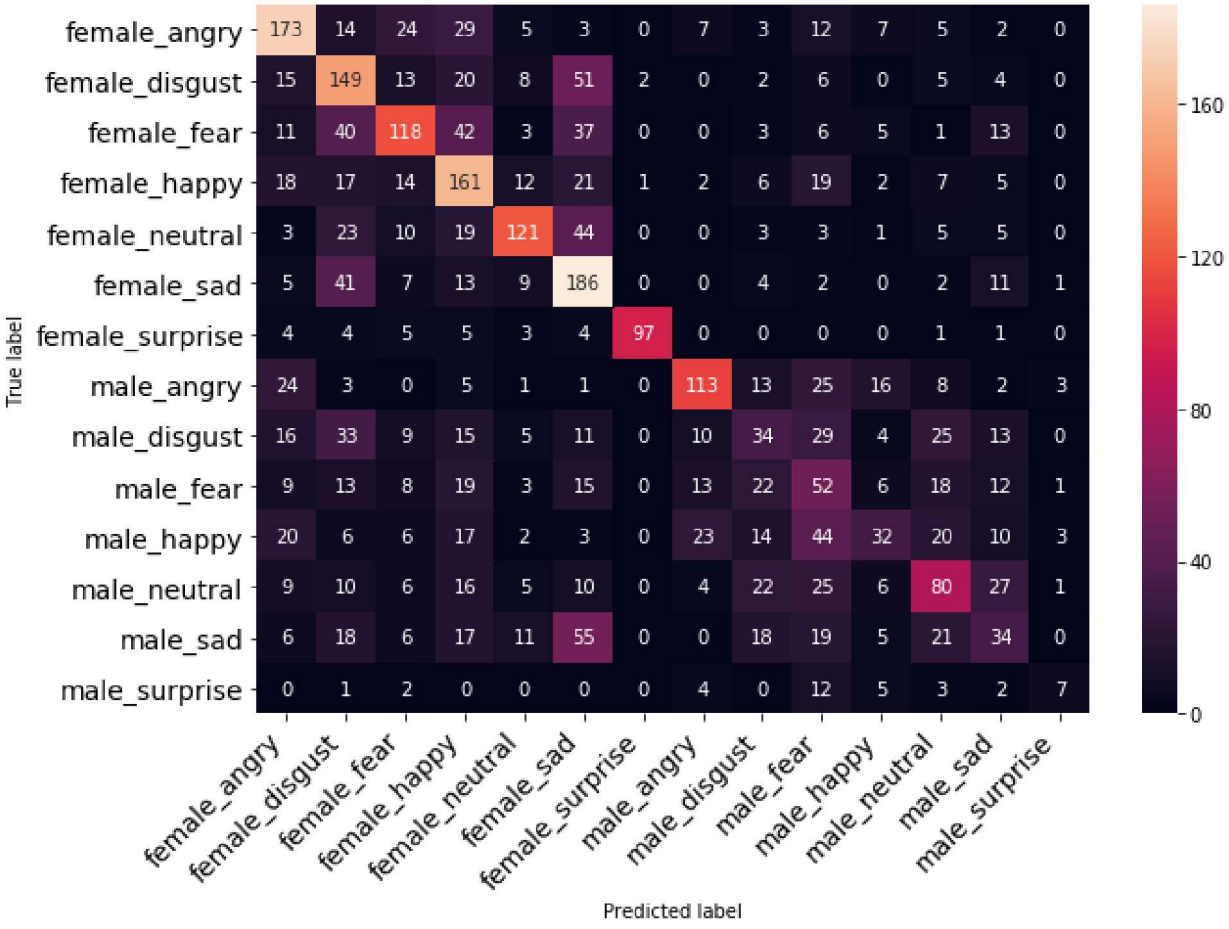
In [70]:

```
# the confusion matrix heat map plot
def print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=1
    """Prints a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as

    Arguments
    ---------
    confusion_matrix: numpy.ndarray
        The numpy.ndarray object returned from a call to sklearn.metrics.confusion_mat
        Similarly constructed ndarrays can also be used.
    class_names: list
        An ordered list of class names, in the order they index the given confusion ma
    figsize: tuple
        A 2-long tuple, the first value determining the horizontal size of the ouputte
        the second determining the vertical size. Defaults to (10,7).
    fontsize: int
        Font size for axes labels. Defaults to 14.

    Returns
    -------
    matplotlib.figure.Figure
        The resulting confusion matrix figure
    """
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
```

```
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='righ
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Gender recode function
def gender(row):
    if row == 'female_disgust' or 'female_fear' or 'female_happy' or 'female_sad' or '
        return 'female'
    elif row == 'male_angry' or 'male_fear' or 'male_happy' or 'male_sad' or 'male_sur
        return 'male'
```

In [71]:
```
# Get the predictions file
finaldf = pd.read_csv("Predictions.csv")
classes = finaldf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(finaldf.actualvalues, finaldf.predictedvalues)
print(accuracy_score(finaldf.actualvalues, finaldf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
```

```
0.4462347911871095
```



In [72]:
```
# Classification report
classes = finaldf.actualvalues.unique()
classes.sort()
print(classification_report(finaldf.actualvalues, finaldf.predictedvalues, target_name
```

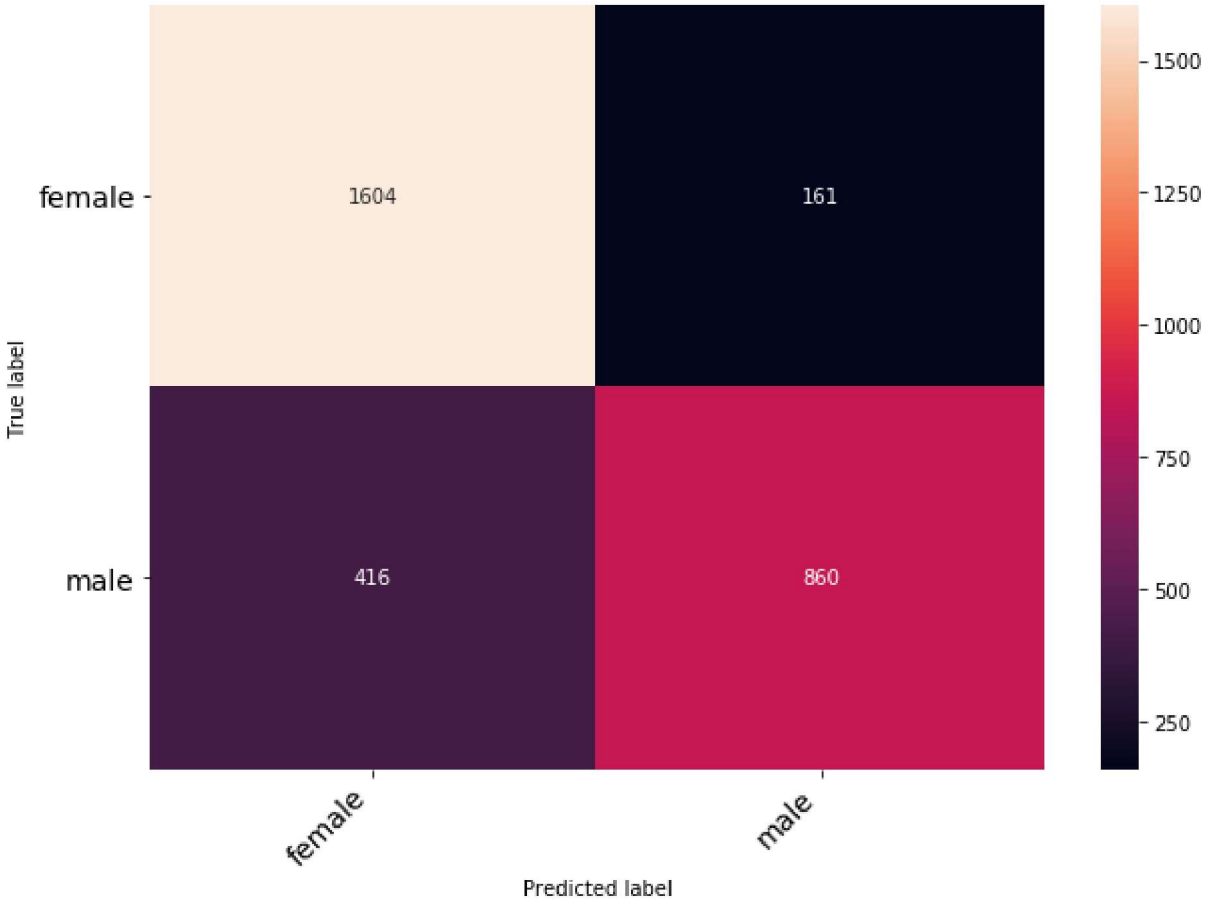
```
                precision    recall  f1-score   support

  female_angry       0.55      0.61      0.58       284
 female_disgust       0.40      0.54      0.46       275
   female_fear       0.52      0.42      0.47       279
  female_happy       0.43      0.56      0.49       285
female_neutral       0.64      0.51      0.57       237
    female_sad       0.42      0.66      0.52       281
female_surprise       0.97      0.78      0.87       124
    male_angry       0.64      0.53      0.58       214
  male_disgust       0.24      0.17      0.20       204
     male_fear       0.20      0.27      0.23       191
    male_happy       0.36      0.16      0.22       200
  male_neutral       0.40      0.36      0.38       221
      male_sad       0.24      0.16      0.19       210
 male_surprise       0.44      0.19      0.27        36

       accuracy                           0.45      3041
      macro avg       0.46      0.42      0.43      3041
   weighted avg       0.45      0.45      0.44      3041
```

In [73]:

```
modidf = finaldf
modidf['actualvalues'] = finaldf.actualvalues.replace({'female_angry':'female'
                                       , 'female_disgust':'female'
                                       , 'female_fear':'female'
                                       , 'female_happy':'female'
                                       , 'female_sad':'female'
                                       , 'female_surprise':'female'
                                       , 'female_neutral':'female'
                                       , 'male_angry':'male'
                                       , 'male_fear':'male'
                                       , 'male_happy':'male'
                                       , 'male_sad':'male'
                                       , 'male_surprise':'male'
                                       , 'male_neutral':'male'
                                       , 'male_disgust':'male'
                                      })

modidf['predictedvalues'] = finaldf.predictedvalues.replace({'female_angry':'female'
                                       , 'female_disgust':'female'
                                       , 'female_fear':'female'
                                       , 'female_happy':'female'
                                       , 'female_sad':'female'
                                       , 'female_surprise':'female'
                                       , 'female_neutral':'female'
                                       , 'male_angry':'male'
                                       , 'male_fear':'male'
                                       , 'male_happy':'male'
                                       , 'male_sad':'male'
                                       , 'male_surprise':'male'
                                       , 'male_neutral':'male'
                                       , 'male_disgust':'male'
                                      })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
```

```
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
```

```
0.8102597829661295
```



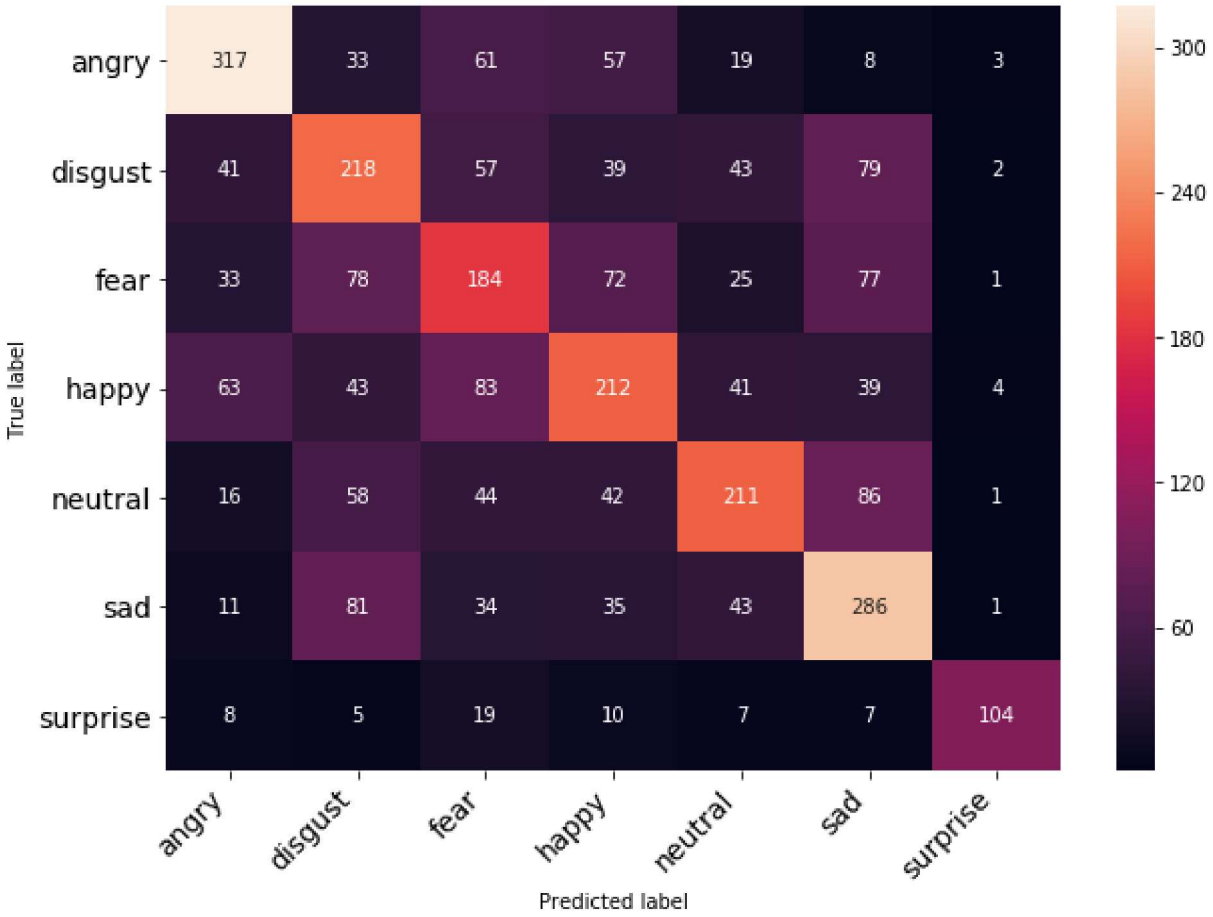
In [74]:
```
# Classification report
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues, target_names=
```

```
              precision    recall  f1-score   support

      female       0.79      0.91      0.85      1765
        male       0.84      0.67      0.75      1276

    accuracy                           0.81      3041
   macro avg       0.82      0.79      0.80      3041
weighted avg       0.81      0.81      0.81      3041
```

In [75]:
```
modidf = pd.read_csv("Predictions.csv")
modidf['actualvalues'] = modidf.actualvalues.replace({'female_angry':'angry'
                                       , 'female_disgust':'disgust'
                                       , 'female_fear':'fear'
                                       , 'female_happy':'happy'
                                       , 'female_sad':'sad'
                                       , 'female_surprise':'surprise'
                                       , 'female_neutral':'neutral'
                                       , 'male_angry':'angry'
                                       , 'male_fear':'fear'
                                       , 'male_happy':'happy'
                                       , 'male_sad':'sad'
```

```
                                       , 'male_surprise':'surprise'
                                       , 'male_neutral':'neutral'
                                       , 'male_disgust':'disgust'
                                      })

modidf['predictedvalues'] = modidf.predictedvalues.replace({'female_angry':'angry'
                                       , 'female_disgust':'disgust'
                                       , 'female_fear':'fear'
                                       , 'female_happy':'happy'
                                       , 'female_sad':'sad'
                                       , 'female_surprise':'surprise'
                                       , 'female_neutral':'neutral'
                                       , 'male_angry':'angry'
                                       , 'male_fear':'fear'
                                       , 'male_happy':'happy'
                                       , 'male_sad':'sad'
                                       , 'male_surprise':'surprise'
                                       , 'male_neutral':'neutral'
                                       , 'male_disgust':'disgust'
                                      })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
```

```
0.5037816507727721
```



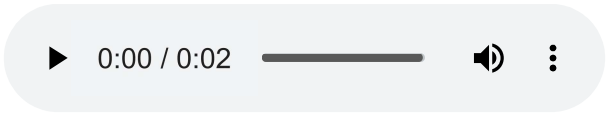
In [76]:
```
# Classification report
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues, target_names=
```

```
              precision    recall  f1-score   support

       angry       0.65      0.64      0.64       498
     disgust       0.42      0.46      0.44       479
        fear       0.38      0.39      0.39       470
       happy       0.45      0.44      0.45       485
     neutral       0.54      0.46      0.50       458
         sad       0.49      0.58      0.53       491
    surprise       0.90      0.65      0.75       160

    accuracy                           0.50      3041
   macro avg       0.55      0.52      0.53      3041
weighted avg       0.51      0.50      0.51      3041
```

In [77]:
```
from keras.models import Sequential, Model, model_from_json
import matplotlib.pyplot as plt
import keras
import pickle
import wave  # !pip install wave
import os
import pandas as pd
import numpy as np
import sys
import warnings
import librosa
import librosa.display
import IPython.display as ipd  # To play sound in the notebook

# ignore warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

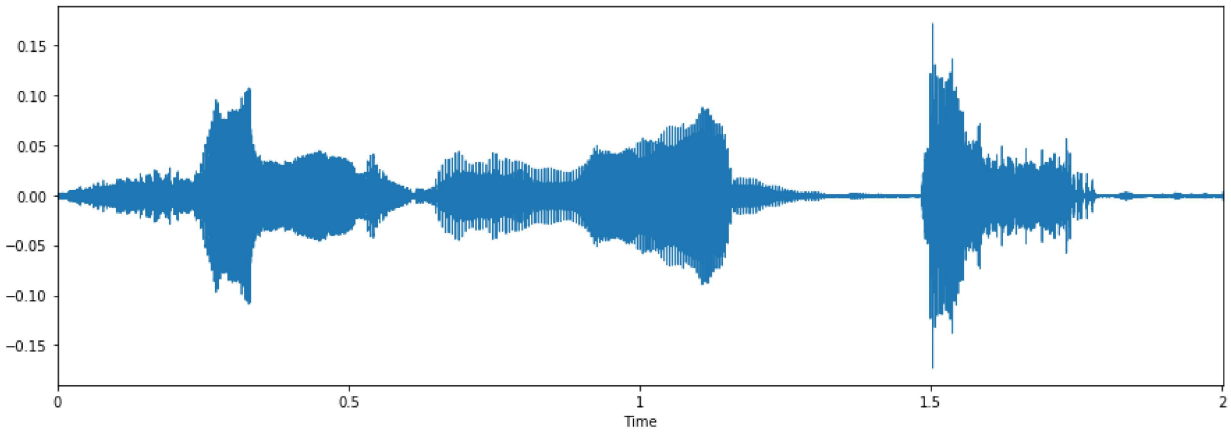
In [79]:
```
data, sampling_rate = librosa.load('/input/toronto-emotional-speech-set-tess/TESS Toro
ipd.Audio('/input/toronto-emotional-speech-set-tess/TESS Toronto emotional speech set
```

Out[79]:



In [80]:
```
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
```

Out[80]:
```
<matplotlib.collections.PolyCollection at 0x7ff6783fe128>
```



In [83]:
```
# loading json and model architecture
json_file = open('/input/speech-emotion-recognition-with-cnn/model_json.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("/input/speech-emotion-recognition-with-cnn/saved_models/Emo
print("Loaded model from disk")

# the optimiser
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accurac
```

```
Loaded model from disk
```

In [84]:
```
# loading json and model architecture
json_file = open('/input/speech-emotion-recognition-with-cnn/model_json.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("/input/speech-emotion-recognition-with-cnn/saved_models/Emo
print("Loaded model from disk")

# the optimiser
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accurac
```

```
Loaded model from disk
```