In [1]:
```
#IMPORT THE LIBRARIES
import pandas as pd
import numpy as np

import os
import sys

# librosa is a Python library for analyzing audio and music. It can be used to extr
import librosa
import librosa.display
import seaborn as sns
import matplotlib.pyplot as plt

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

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

# for cnn model
import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM,BatchNormalization
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint

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

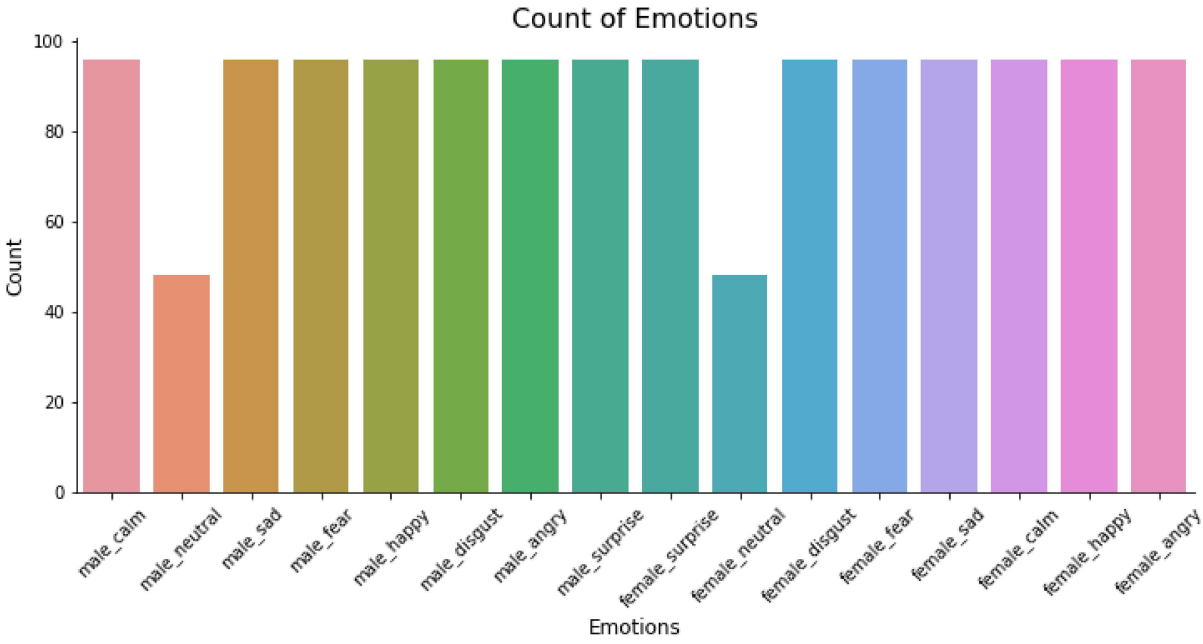
In [2]:
```
# path to the directory of the dataset
RAV = "/kaggle/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
```

In [3]:
```
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:'calm', 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[3]:
```
male_fear          96
male_angry         96
male_happy         96
male_calm          96
female_disgust     96
male_sad           96
male_disgust       96
female_angry       96
female_happy       96
female_surprise    96
female_sad         96
female_fear        96
female_calm        96
male_surprise      96
male_neutral       48
female_neutral     48
Name: labels, dtype: int64
```

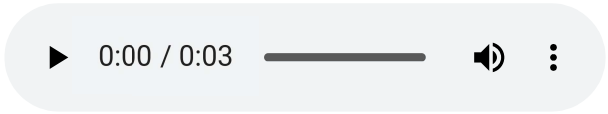
In [4]:
```
plt.figure(figsize=(12, 5))
plt.title('Count of Emotions', size=16)
sns.countplot(RAV_df.labels)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
plt.xticks(rotation=45)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```



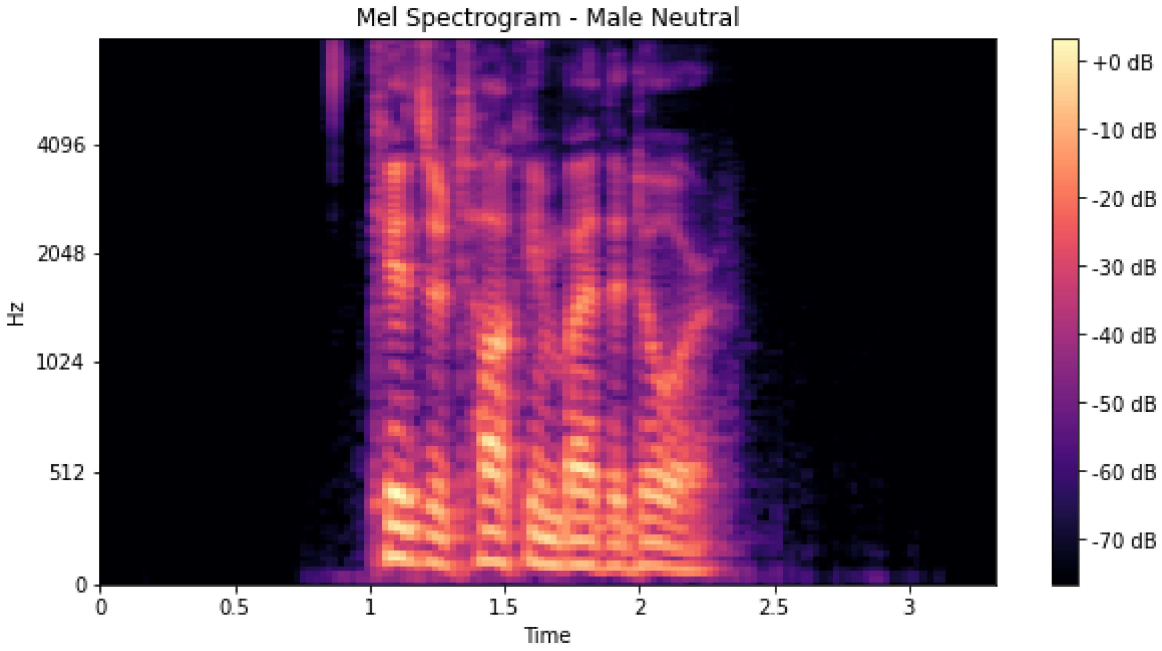
Now, we will compare mel spectograms of male and female neutral audio clips.

In [5]:
```
# MALE NEUTRAL
fname1=RAV+'Actor_01/03-01-01-01-01-01-01.wav'
data, sr = librosa.load(fname1)
ipd.Audio(fname1)
```

Out[5]:



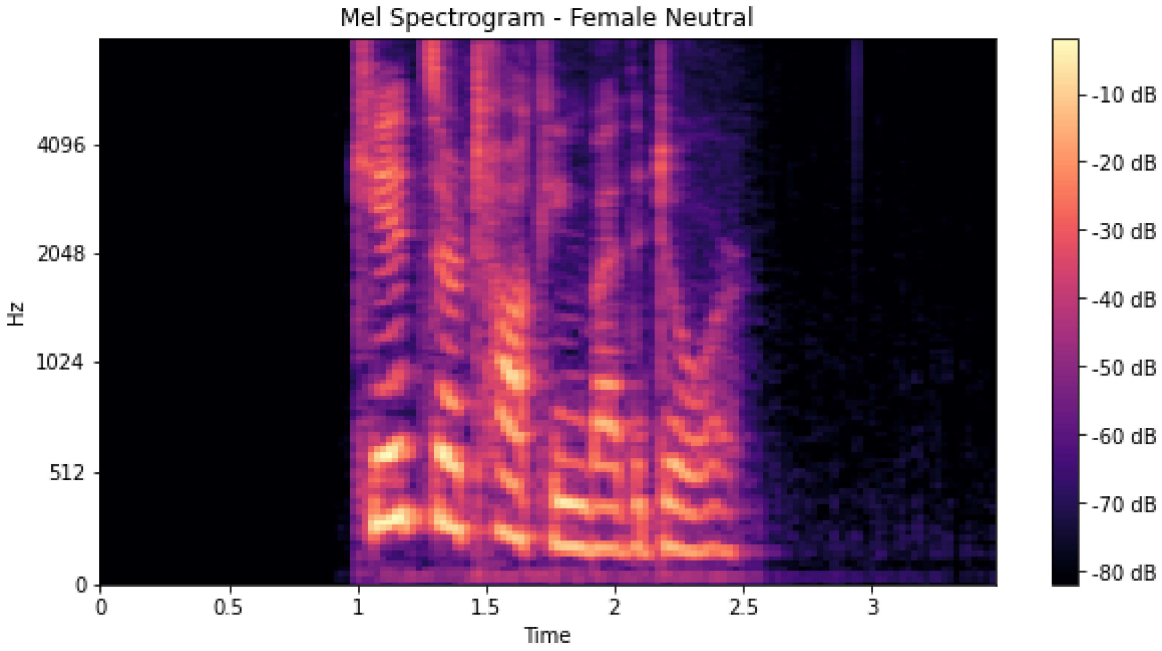
In [6]:
```
# CREATE LOG MEL SPECTROGRAM
plt.figure(figsize=(10, 5))
spectrogram = librosa.feature.melspectrogram(y=data, sr=sr, n_mels=128,fmax=8000)
spectrogram = librosa.power_to_db(spectrogram)
librosa.display.specshow(spectrogram, y_axis='mel', fmax=8000, x_axis='time');
plt.title('Mel Spectrogram - Male Neutral')
plt.colorbar(format='%+2.0f dB');
```



In [7]:
```
# FEMALE NEUTRAL
fname2=RAV+'Actor_14/03-01-01-01-01-01-14.wav'
data, sr = librosa.load(fname2)
ipd.Audio(fname2)
```

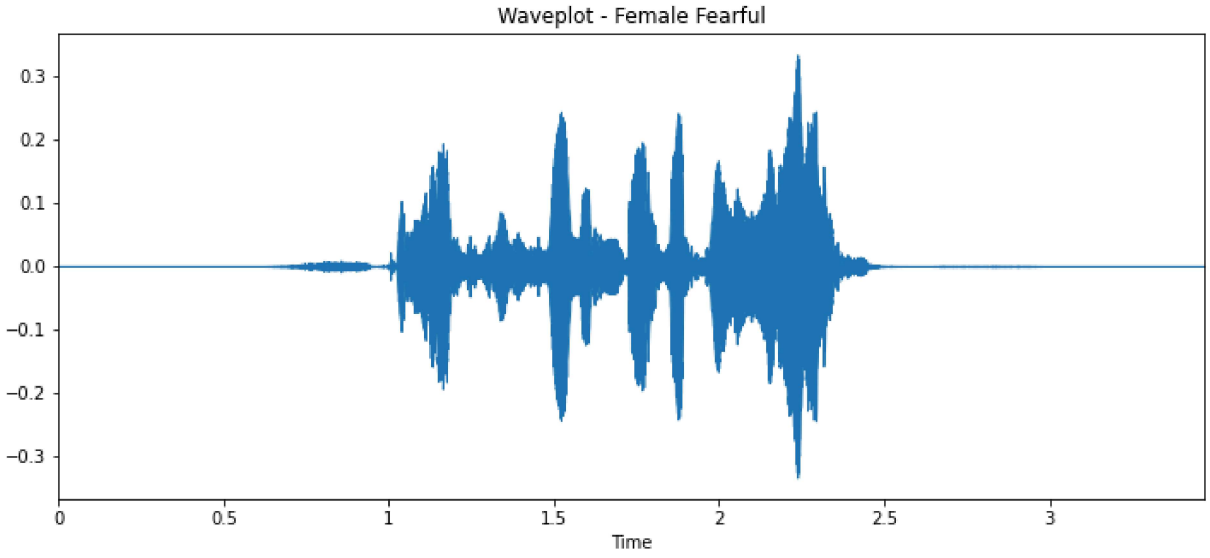
Out[7]:

In [8]:
```
# CREATE LOG MEL SPECTROGRAM
plt.figure(figsize=(10, 5))
spectrogram = librosa.feature.melspectrogram(y=data, sr=sr, n_mels=128,fmax=8000)
spectrogram = librosa.power_to_db(spectrogram)
librosa.display.specshow(spectrogram, y_axis='mel', fmax=8000, x_axis='time');
plt.title('Mel Spectrogram - Female Neutral')
plt.colorbar(format='%+2.0f dB');
```



In [9]:
```
# Pick a fearful track
fname3 = RAV + 'Actor_14/03-01-06-02-02-02-14.wav'
data, sr = librosa.load(fname3)
plt.figure(figsize=(12, 5))
librosa.display.waveplot(data, sr=sr)
plt.title('Waveplot - Female Fearful')
# Lets play the audio
ipd.Audio(fname3)
```

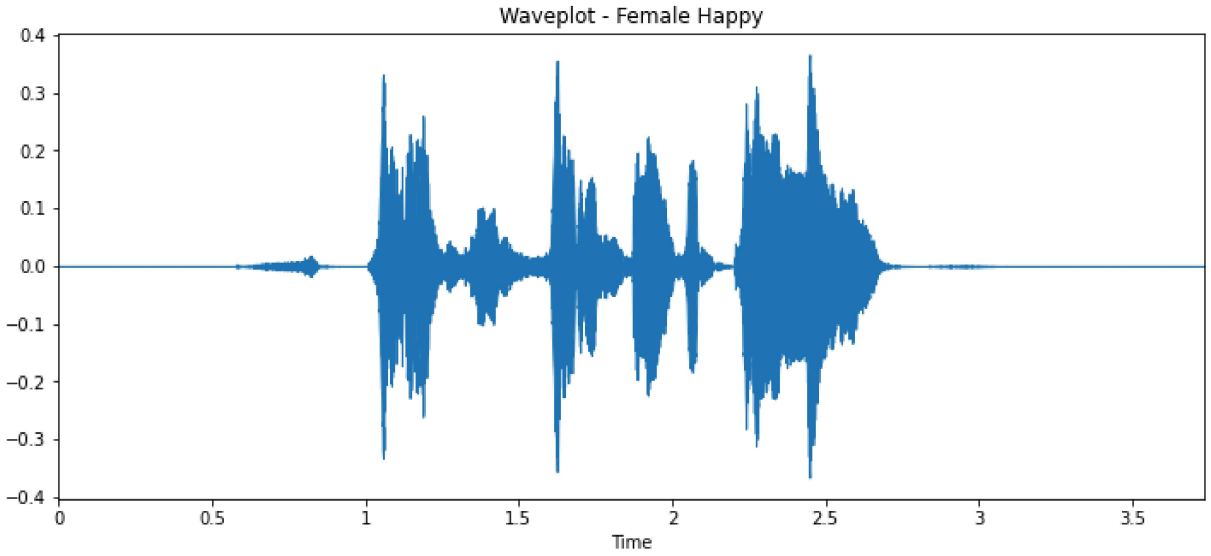
Out[9]:



In [10]:
```
# Pick a happy track
fname4 = RAV + 'Actor_14/03-01-03-02-02-02-14.wav'
data, sr = librosa.load(fname4)
plt.figure(figsize=(12, 5))
librosa.display.waveplot(data, sr=sr)
```

```
plt.title('Waveplot - Female Happy')

# Lets play the audio
ipd.Audio(fname4)
```

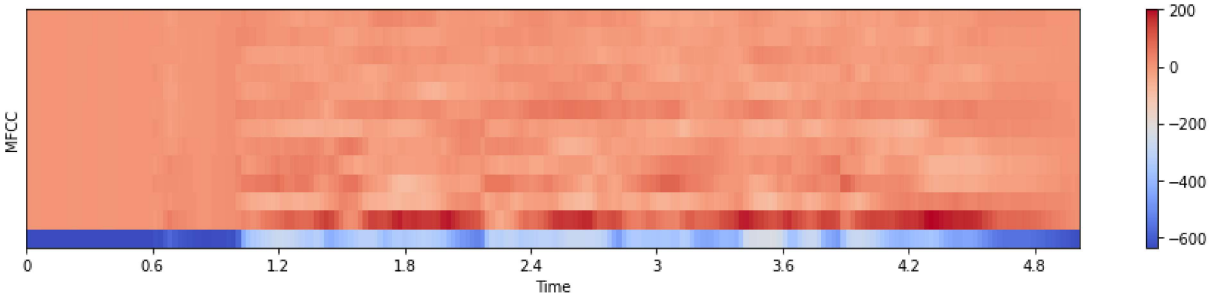
Out[10]:



In [11]:

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


# MFCC
plt.figure(figsize=(16, 10))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()
```

Out[11]: <matplotlib.colorbar.Colorbar at 0x7f610cc6d650>

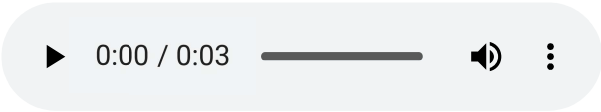


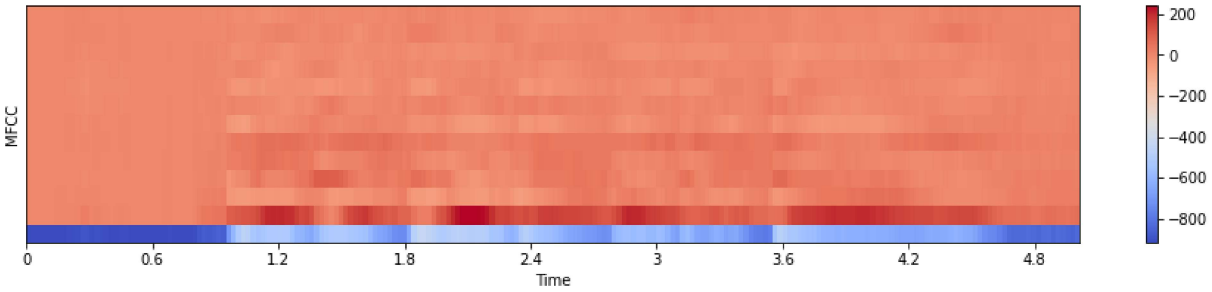
In [12]:

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

```
# MFCC
plt.figure(figsize=(16, 10))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()

ipd.Audio(path)
```

Out[12]:



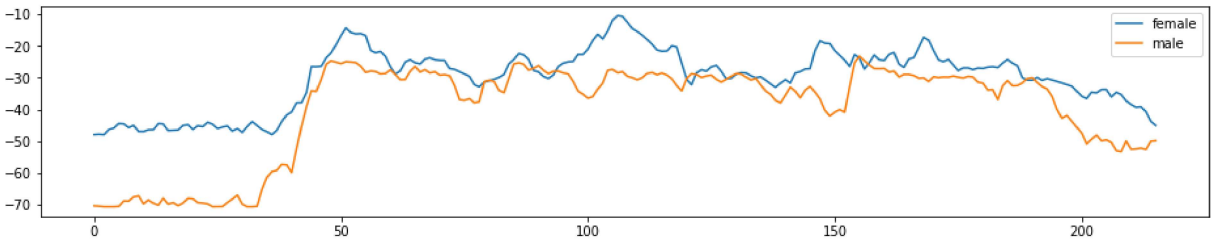


In [13]:

```
# Gender - Female; Emotion - angry
path = "/kaggle/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Acto
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,
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))

# Gender - Male; Emotion - angry
path = "/kaggle/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/Acto
X, sample_rate = librosa.load(path, res_type='kaiser_fast',duration=2.5,sr=22050*2,
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=(16,10))
plt.subplot(3,1,1)
plt.plot(female, label='female')
plt.plot(male, label='male')
plt.legend()
```

```
216
216
```

Out[13]: <matplotlib.legend.Legend at 0x7f610cb5a7d0>

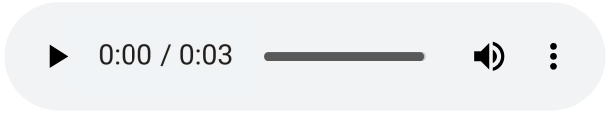


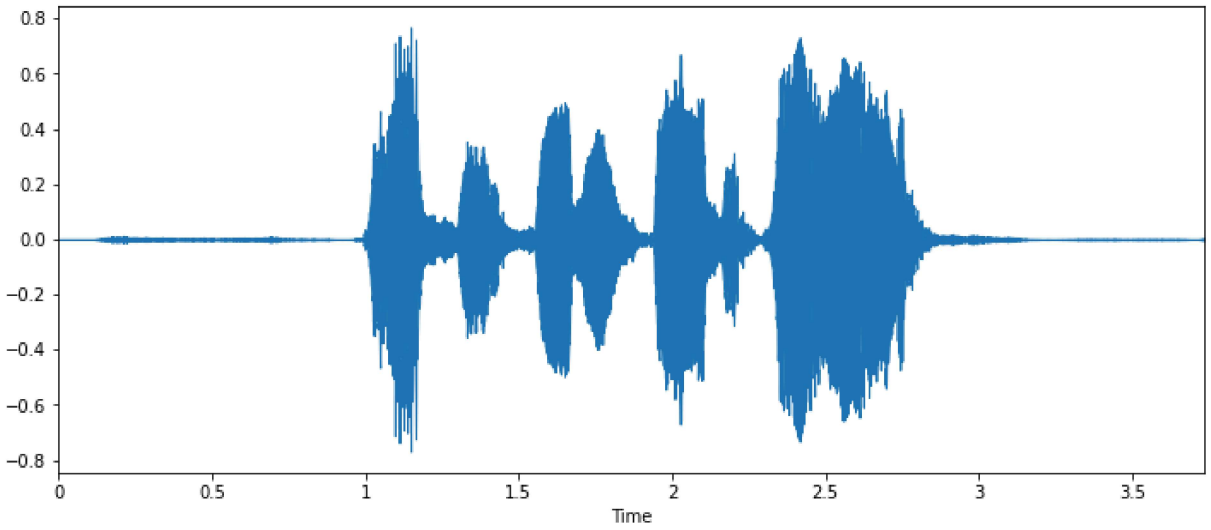
In [14]:
```
# NOISE
def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data
# STRETCH
def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate)
# SHIFT
def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)
# PITCH
def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sampling_rate, pitch_factor)
```

In [15]:
```
# Trying different functions above
path = np.array(RAV_df['path'])[303]
data, sample_rate = librosa.load(path)
```

In [16]:
```
# NORMAL AUDIO
plt.figure(figsize=(12, 5))
librosa.display.waveplot(y=data, sr=sample_rate)
Audio(path)
```

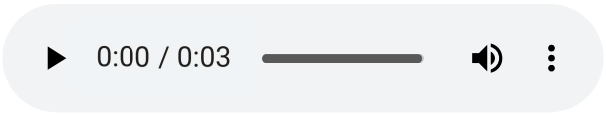
Out[16]:

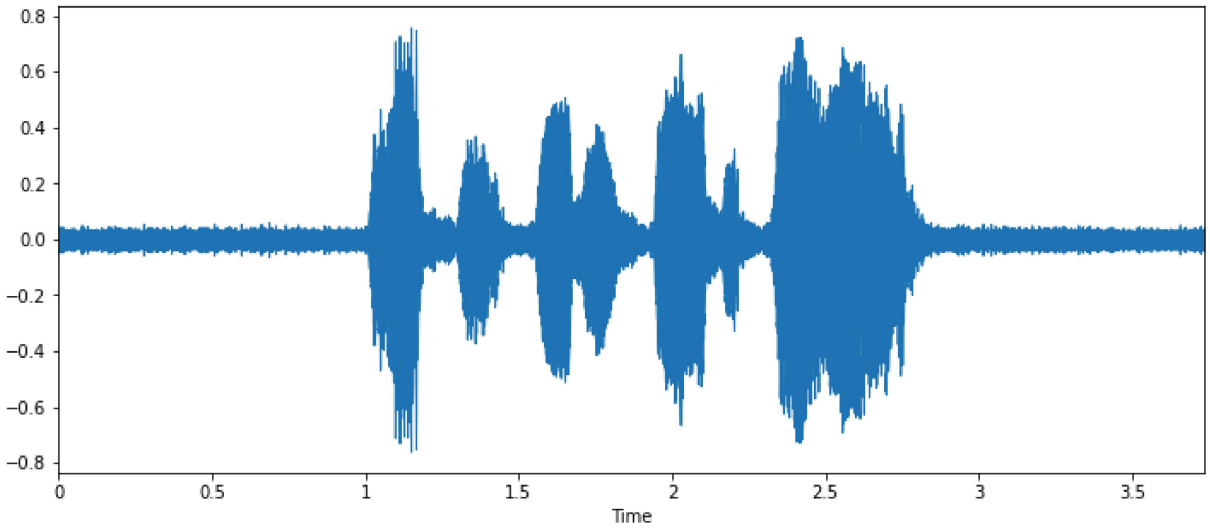




In [17]:
```
# AUDIO WITH NOISE
x = noise(data)
plt.figure(figsize=(12,5))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[17]:

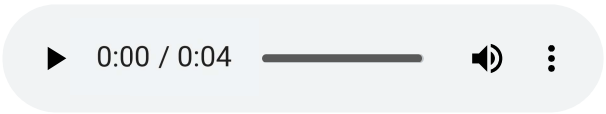


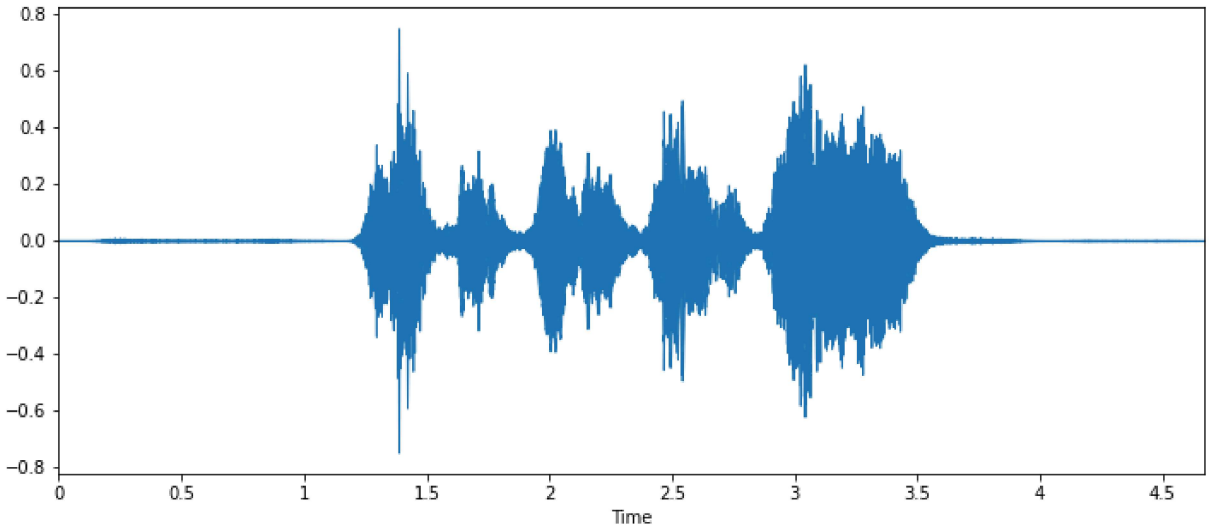


In [18]:

```
# STRETCHED AUDIO
x = stretch(data)
plt.figure(figsize=(12, 5))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[18]:

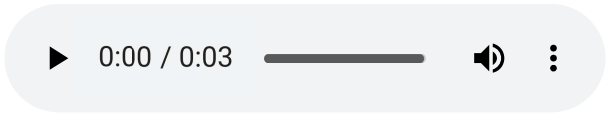


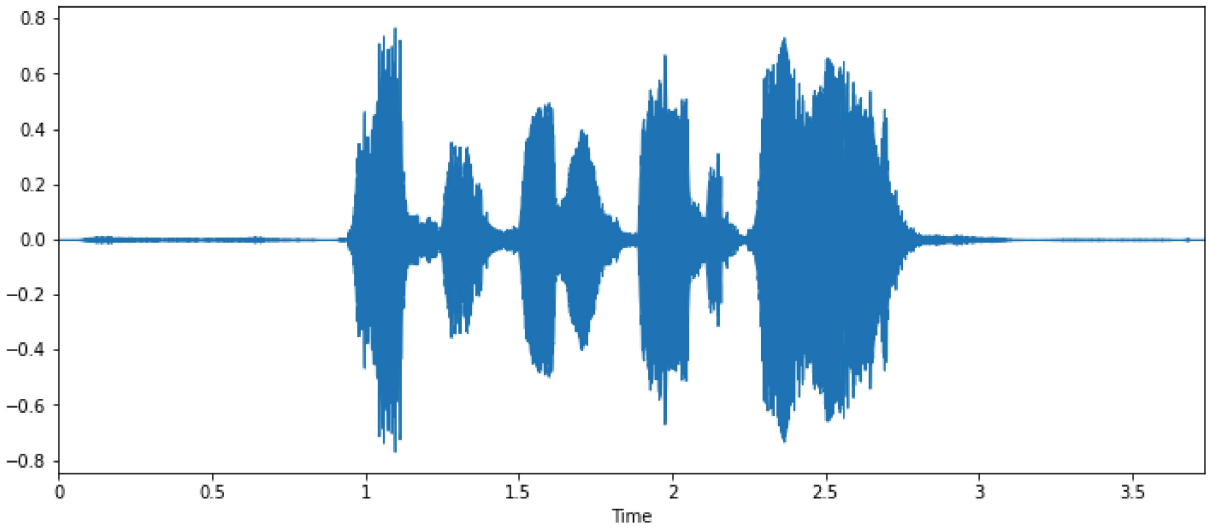


In [19]:

```
# SHIFTED AUDIO
x = shift(data)
plt.figure(figsize=(12,5))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[19]:

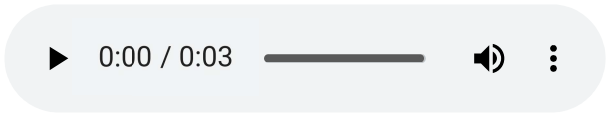


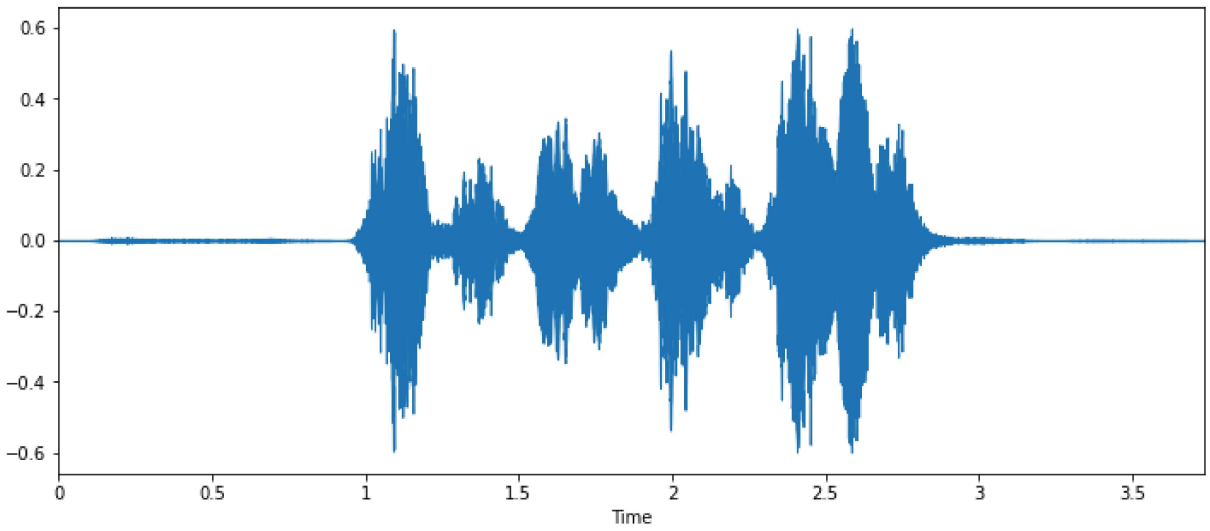


In [20]:

```
# AUDIO WITH PITCH
x = pitch(data, sample_rate)
plt.figure(figsize=(12, 5))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[20]:





FEATURE EXTRACTION

In [21]:

```
# def feat_ext(data):
#     mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
#     return mfcc
```

```
# def get_feat(path):
#     data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)
#     # normal data
#     res1 = feat_ext(data)
#     result = np.array(res1)
#     #data with noise
#     noise_data = noise(data)
#     res2 = feat_ext(noise_data)
#     result = np.vstack((result, res2))
#     #data with stretch and pitch
#     new_data = stretch(data)
#     data_stretch_pitch = pitch(new_data, sample_rate)
#     res3 = feat_ext(data_stretch_pitch)
#     result = np.vstack((result, res3))
#     return result
```

In [22]:
```
#RAV_df.head()
```

In [23]:
```
# X, Y = [], []
# for path, emotion in zip(RAV_df['path'], RAV_df['labels']):
#     feature = get_feat(path)
#     for ele in feature:
#         X.append(ele)
#         Y.append(emotion)
```

In [24]:
```
# Features = pd.DataFrame(X)
# Features['labels'] = Y
# Features.to_csv('features.csv', index=False)
# Features.head()
```

In [25]:
```
# this step can used directly from saved feature .csv file
Features = pd.read_csv('../input/features/features.csv')
Features.head()
```

Out[25]:

	0	1	2	3	4	5	6	
0	-637.701233	104.299019	4.894947	20.494011	12.552954	2.851410	-6.633390	-4.091
1	-596.908460	86.871936	9.470162	17.109819	11.198966	1.541056	-6.677264	-5.755
2	-698.086548	99.795929	1.892679	19.915264	7.532868	1.265761	-9.188656	-5.798
3	-279.141052	41.092949	-21.319229	7.802911	-13.140503	-9.407660	-15.580647	-6.097
4	-160.074686	17.576058	-2.147436	3.133417	-4.745002	-6.510771	-5.911591	-4.481

5 rows × 21 columns

DATA PREPROCESSING

In [26]:
```
X = Features.iloc[: ,:-1].values
```

```
Y = Features['labels'].values
```

In [27]:
```
# As this is a multiclass classification problem onehotencoding our Y.
encoder = OneHotEncoder()
Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()
```

In [28]:
```
# splitting data
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0, shuffle=T
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[28]:
```
((3240, 20), (3240, 16), (1080, 20), (1080, 16))
```

In [29]:
```
# scaling our data with sklearn's Standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[29]:
```
((3240, 20), (3240, 16), (1080, 20), (1080, 16))
```

IMPLOYING CNN MODEL

In [30]:
```
x_traincnn =np.expand_dims(x_train, axis=2)
x_testcnn= np.expand_dims(x_test, axis=2)
x_traincnn.shape, y_train.shape, x_testcnn.shape, y_test.shape
```

Out[30]:
```
((3240, 20, 1), (3240, 16), (1080, 20, 1), (1080, 16))
```

In [31]:
```
model = Sequential()
model.add(Conv1D(2048, kernel_size=5, strides=1, padding='same', activation='relu',
model.add(MaxPooling1D(pool_size=2, strides = 2, padding = 'same'))
model.add(BatchNormalization())

model.add(Conv1D(1024, kernel_size=5, strides=1, padding='same', activation='relu')
model.add(MaxPooling1D(pool_size=2, strides = 2, padding = 'same'))
model.add(BatchNormalization())

model.add(Conv1D(512, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides = 2, padding = 'same'))
model.add(BatchNormalization())

model.add(LSTM(256, return_sequences=True))

model.add(LSTM(128))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'))
```

```
model.add(Dropout(0.2))

model.add(Dense(16, activation='softmax'))

optimiser = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimiser,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 20, 2048)          12288
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 10, 2048)          0
_________________________________________________________________
batch_normalization (BatchNo (None, 10, 2048)          8192
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 10, 1024)          10486784
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 5, 1024)           0
_________________________________________________________________
batch_normalization_1 (Batch (None, 5, 1024)           4096
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 5, 512)            2621952
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 3, 512)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 512)            2048
_________________________________________________________________
lstm (LSTM)                  (None, 3, 256)            787456
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               197120
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
=================================================================
Total params: 14,147,312
Trainable params: 14,140,144
Non-trainable params: 7,168
_________________________________________________________________
```

In [32]:
```
history = model.fit(x_traincnn, y_train, batch_size=64, epochs=200, validation_data
```

```
Epoch 1/200
51/51 [==============================] - 12s 63ms/step - loss: 2.7709 - accuracy: 0.
0758 - val_loss: 2.7703 - val_accuracy: 0.0963
Epoch 2/200
51/51 [==============================] - 1s 28ms/step - loss: 2.7282 - accuracy: 0.1
090 - val_loss: 2.7667 - val_accuracy: 0.0954
Epoch 3/200
51/51 [==============================] - 1s 28ms/step - loss: 2.6970 - accuracy: 0.1
329 - val_loss: 2.7641 - val_accuracy: 0.0731
Epoch 4/200
51/51 [==============================] - 1s 28ms/step - loss: 2.6406 - accuracy: 0.1
492 - val_loss: 2.7552 - val_accuracy: 0.0769
Epoch 5/200
51/51 [==============================] - 2s 30ms/step - loss: 2.5877 - accuracy: 0.1
834 - val_loss: 2.7413 - val_accuracy: 0.1204
Epoch 6/200
51/51 [==============================] - 2s 29ms/step - loss: 2.5063 - accuracy: 0.1
984 - val_loss: 2.7000 - val_accuracy: 0.1380
Epoch 7/200
51/51 [==============================] - 2s 30ms/step - loss: 2.3804 - accuracy: 0.2
488 - val_loss: 2.7023 - val_accuracy: 0.1398
Epoch 8/200
51/51 [==============================] - 1s 28ms/step - loss: 2.2972 - accuracy: 0.2
705 - val_loss: 2.6241 - val_accuracy: 0.2083
Epoch 9/200
51/51 [==============================] - 1s 28ms/step - loss: 2.2343 - accuracy: 0.2
907 - val_loss: 2.5070 - val_accuracy: 0.2343
Epoch 10/200
51/51 [==============================] - 1s 28ms/step - loss: 2.0515 - accuracy: 0.3
203 - val_loss: 2.4090 - val_accuracy: 0.2963
Epoch 11/200
51/51 [==============================] - 1s 28ms/step - loss: 1.9378 - accuracy: 0.3
838 - val_loss: 2.1638 - val_accuracy: 0.3157
Epoch 12/200
51/51 [==============================] - 1s 27ms/step - loss: 1.8182 - accuracy: 0.4
064 - val_loss: 1.8520 - val_accuracy: 0.4296
Epoch 13/200
51/51 [==============================] - 1s 27ms/step - loss: 1.6792 - accuracy: 0.4
490 - val_loss: 1.6521 - val_accuracy: 0.4519
Epoch 14/200
51/51 [==============================] - 1s 28ms/step - loss: 1.6184 - accuracy: 0.4
450 - val_loss: 1.4728 - val_accuracy: 0.5148
Epoch 15/200
51/51 [==============================] - 2s 30ms/step - loss: 1.5275 - accuracy: 0.4
939 - val_loss: 1.3277 - val_accuracy: 0.6009
Epoch 16/200
51/51 [==============================] - 1s 27ms/step - loss: 1.4339 - accuracy: 0.5
164 - val_loss: 1.2441 - val_accuracy: 0.5889
Epoch 17/200
51/51 [==============================] - 1s 27ms/step - loss: 1.3341 - accuracy: 0.5
504 - val_loss: 1.2181 - val_accuracy: 0.5861
Epoch 18/200
51/51 [==============================] - 1s 27ms/step - loss: 1.2546 - accuracy: 0.5
737 - val_loss: 1.1610 - val_accuracy: 0.6130
Epoch 19/200
51/51 [==============================] - 1s 28ms/step - loss: 1.2327 - accuracy: 0.5
```

```
779 - val_loss: 1.1473 - val_accuracy: 0.6481
Epoch 20/200
51/51 [==============================] - 1s 28ms/step - loss: 1.1586 - accuracy: 0.5
884 - val_loss: 1.0858 - val_accuracy: 0.6361
Epoch 21/200
51/51 [==============================] - 1s 28ms/step - loss: 1.0725 - accuracy: 0.6
367 - val_loss: 1.0517 - val_accuracy: 0.6657
Epoch 22/200
51/51 [==============================] - 1s 28ms/step - loss: 1.0290 - accuracy: 0.6
587 - val_loss: 0.9894 - val_accuracy: 0.6861
Epoch 23/200
51/51 [==============================] - 1s 29ms/step - loss: 0.9869 - accuracy: 0.6
501 - val_loss: 1.0056 - val_accuracy: 0.6898
Epoch 24/200
51/51 [==============================] - 1s 27ms/step - loss: 0.9377 - accuracy: 0.6
928 - val_loss: 1.0095 - val_accuracy: 0.6963
Epoch 25/200
51/51 [==============================] - 1s 28ms/step - loss: 0.9200 - accuracy: 0.6
740 - val_loss: 0.9382 - val_accuracy: 0.7139
Epoch 26/200
51/51 [==============================] - 1s 28ms/step - loss: 0.8835 - accuracy: 0.6
924 - val_loss: 0.9373 - val_accuracy: 0.7139
Epoch 27/200
51/51 [==============================] - 1s 28ms/step - loss: 0.7620 - accuracy: 0.7
281 - val_loss: 0.8988 - val_accuracy: 0.7287
Epoch 28/200
51/51 [==============================] - 2s 30ms/step - loss: 0.8019 - accuracy: 0.7
183 - val_loss: 0.9450 - val_accuracy: 0.7102
Epoch 29/200
51/51 [==============================] - 2s 30ms/step - loss: 0.7783 - accuracy: 0.7
301 - val_loss: 0.9389 - val_accuracy: 0.7343
Epoch 30/200
51/51 [==============================] - 2s 30ms/step - loss: 0.7078 - accuracy: 0.7
588 - val_loss: 0.9021 - val_accuracy: 0.7333
Epoch 31/200
51/51 [==============================] - 1s 28ms/step - loss: 0.6743 - accuracy: 0.7
569 - val_loss: 0.8275 - val_accuracy: 0.7611
Epoch 32/200
51/51 [==============================] - 1s 28ms/step - loss: 0.6341 - accuracy: 0.7
753 - val_loss: 0.8824 - val_accuracy: 0.7407
Epoch 33/200
51/51 [==============================] - 1s 28ms/step - loss: 0.6187 - accuracy: 0.7
828 - val_loss: 0.8039 - val_accuracy: 0.7620
Epoch 34/200
51/51 [==============================] - 1s 28ms/step - loss: 0.5954 - accuracy: 0.7
952 - val_loss: 0.8233 - val_accuracy: 0.7667
Epoch 35/200
51/51 [==============================] - 1s 28ms/step - loss: 0.5780 - accuracy: 0.7
941 - val_loss: 0.8849 - val_accuracy: 0.7694
Epoch 36/200
51/51 [==============================] - 1s 28ms/step - loss: 0.5426 - accuracy: 0.8
037 - val_loss: 0.9950 - val_accuracy: 0.7444
Epoch 37/200
51/51 [==============================] - 1s 28ms/step - loss: 0.6057 - accuracy: 0.7
879 - val_loss: 0.9979 - val_accuracy: 0.7407
Epoch 38/200
```

```
51/51 [==============================] - 2s 30ms/step - loss: 0.5701 - accuracy: 0.8
076 - val_loss: 0.9828 - val_accuracy: 0.7565
Epoch 39/200
51/51 [==============================] - 1s 27ms/step - loss: 0.5307 - accuracy: 0.8
246 - val_loss: 1.0547 - val_accuracy: 0.7454
Epoch 40/200
51/51 [==============================] - 1s 28ms/step - loss: 0.5233 - accuracy: 0.8
276 - val_loss: 0.8882 - val_accuracy: 0.7815
Epoch 41/200
51/51 [==============================] - 1s 28ms/step - loss: 0.4998 - accuracy: 0.8
259 - val_loss: 0.9240 - val_accuracy: 0.7741
Epoch 42/200
51/51 [==============================] - 1s 28ms/step - loss: 0.4691 - accuracy: 0.8
299 - val_loss: 1.0453 - val_accuracy: 0.7583
Epoch 43/200
51/51 [==============================] - 1s 28ms/step - loss: 0.4579 - accuracy: 0.8
451 - val_loss: 1.0130 - val_accuracy: 0.7676
Epoch 44/200
51/51 [==============================] - 1s 27ms/step - loss: 0.4787 - accuracy: 0.8
496 - val_loss: 0.9477 - val_accuracy: 0.7593
Epoch 45/200
51/51 [==============================] - 1s 28ms/step - loss: 0.4996 - accuracy: 0.8
398 - val_loss: 1.1656 - val_accuracy: 0.7491
Epoch 46/200
51/51 [==============================] - 1s 29ms/step - loss: 0.4413 - accuracy: 0.8
543 - val_loss: 1.0257 - val_accuracy: 0.7593
Epoch 47/200
51/51 [==============================] - 1s 28ms/step - loss: 0.3930 - accuracy: 0.8
602 - val_loss: 1.0082 - val_accuracy: 0.7713
Epoch 48/200
51/51 [==============================] - 1s 28ms/step - loss: 0.3806 - accuracy: 0.8
735 - val_loss: 0.9779 - val_accuracy: 0.7880
Epoch 49/200
51/51 [==============================] - 1s 28ms/step - loss: 0.3831 - accuracy: 0.8
704 - val_loss: 0.9863 - val_accuracy: 0.7972
Epoch 50/200
51/51 [==============================] - 1s 28ms/step - loss: 0.3238 - accuracy: 0.8
964 - val_loss: 0.9486 - val_accuracy: 0.7963
Epoch 51/200
51/51 [==============================] - 1s 28ms/step - loss: 0.3243 - accuracy: 0.8
884 - val_loss: 0.9274 - val_accuracy: 0.8037
Epoch 52/200
51/51 [==============================] - 2s 31ms/step - loss: 0.3686 - accuracy: 0.8
711 - val_loss: 1.0759 - val_accuracy: 0.7741
Epoch 53/200
51/51 [==============================] - 2s 30ms/step - loss: 0.3925 - accuracy: 0.8
790 - val_loss: 1.0275 - val_accuracy: 0.7843
Epoch 54/200
51/51 [==============================] - 1s 27ms/step - loss: 0.4007 - accuracy: 0.8
721 - val_loss: 0.8979 - val_accuracy: 0.7917
Epoch 55/200
51/51 [==============================] - 1s 27ms/step - loss: 0.3374 - accuracy: 0.8
888 - val_loss: 0.9418 - val_accuracy: 0.7870
Epoch 56/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2985 - accuracy: 0.8
996 - val_loss: 0.9722 - val_accuracy: 0.8046
```

```
Epoch 57/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2738 - accuracy: 0.9
134 - val_loss: 1.0692 - val_accuracy: 0.7935
Epoch 58/200
51/51 [==============================] - 1s 27ms/step - loss: 0.2945 - accuracy: 0.8
980 - val_loss: 1.0453 - val_accuracy: 0.7963
Epoch 59/200
51/51 [==============================] - 1s 27ms/step - loss: 0.2980 - accuracy: 0.9
049 - val_loss: 1.0056 - val_accuracy: 0.8083
Epoch 60/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2864 - accuracy: 0.9
134 - val_loss: 0.9478 - val_accuracy: 0.8093
Epoch 61/200
51/51 [==============================] - 2s 30ms/step - loss: 0.2830 - accuracy: 0.9
059 - val_loss: 1.0274 - val_accuracy: 0.8037
Epoch 62/200
51/51 [==============================] - 1s 27ms/step - loss: 0.3249 - accuracy: 0.8
910 - val_loss: 1.1934 - val_accuracy: 0.7667
Epoch 63/200
51/51 [==============================] - 1s 27ms/step - loss: 0.3376 - accuracy: 0.8
834 - val_loss: 1.1193 - val_accuracy: 0.7759
Epoch 64/200
51/51 [==============================] - 1s 27ms/step - loss: 0.3362 - accuracy: 0.8
950 - val_loss: 1.1183 - val_accuracy: 0.7843
Epoch 65/200
51/51 [==============================] - 1s 27ms/step - loss: 0.3135 - accuracy: 0.9
079 - val_loss: 1.1180 - val_accuracy: 0.7704
Epoch 66/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2827 - accuracy: 0.9
150 - val_loss: 1.0912 - val_accuracy: 0.7861
Epoch 67/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2525 - accuracy: 0.9
153 - val_loss: 1.0247 - val_accuracy: 0.7963
Epoch 68/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2642 - accuracy: 0.9
159 - val_loss: 1.0906 - val_accuracy: 0.7963
Epoch 69/200
51/51 [==============================] - 2s 30ms/step - loss: 0.2312 - accuracy: 0.9
280 - val_loss: 1.3001 - val_accuracy: 0.7926
Epoch 70/200
51/51 [==============================] - 1s 27ms/step - loss: 0.3001 - accuracy: 0.9
112 - val_loss: 1.0851 - val_accuracy: 0.7898
Epoch 71/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2555 - accuracy: 0.9
201 - val_loss: 1.1930 - val_accuracy: 0.7843
Epoch 72/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2596 - accuracy: 0.9
088 - val_loss: 1.1226 - val_accuracy: 0.8009
Epoch 73/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2085 - accuracy: 0.9
326 - val_loss: 1.0416 - val_accuracy: 0.8046
Epoch 74/200
51/51 [==============================] - 2s 30ms/step - loss: 0.2003 - accuracy: 0.9
372 - val_loss: 1.1461 - val_accuracy: 0.8120
Epoch 75/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1873 - accuracy: 0.9
```

```
399 - val_loss: 1.1281 - val_accuracy: 0.8167
Epoch 76/200
51/51 [==============================] - 2s 31ms/step - loss: 0.2010 - accuracy: 0.9
321 - val_loss: 1.1329 - val_accuracy: 0.8102
Epoch 77/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1936 - accuracy: 0.9
407 - val_loss: 1.1264 - val_accuracy: 0.8093
Epoch 78/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1939 - accuracy: 0.9
341 - val_loss: 1.1370 - val_accuracy: 0.8074
Epoch 79/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1780 - accuracy: 0.9
385 - val_loss: 1.2362 - val_accuracy: 0.8093
Epoch 80/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1509 - accuracy: 0.9
522 - val_loss: 1.2081 - val_accuracy: 0.8111
Epoch 81/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1632 - accuracy: 0.9
379 - val_loss: 1.2110 - val_accuracy: 0.8185
Epoch 82/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1624 - accuracy: 0.9
420 - val_loss: 1.2286 - val_accuracy: 0.8083
Epoch 83/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1804 - accuracy: 0.9
393 - val_loss: 1.1758 - val_accuracy: 0.8250
Epoch 84/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1534 - accuracy: 0.9
490 - val_loss: 1.1701 - val_accuracy: 0.8222
Epoch 85/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1599 - accuracy: 0.9
463 - val_loss: 1.1322 - val_accuracy: 0.8269
Epoch 86/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1521 - accuracy: 0.9
533 - val_loss: 1.2956 - val_accuracy: 0.8093
Epoch 87/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2028 - accuracy: 0.9
407 - val_loss: 1.2097 - val_accuracy: 0.8204
Epoch 88/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1860 - accuracy: 0.9
393 - val_loss: 1.4493 - val_accuracy: 0.7935
Epoch 89/200
51/51 [==============================] - 1s 27ms/step - loss: 0.2388 - accuracy: 0.9
274 - val_loss: 1.3950 - val_accuracy: 0.7926
Epoch 90/200
51/51 [==============================] - 1s 27ms/step - loss: 0.2759 - accuracy: 0.9
254 - val_loss: 1.2683 - val_accuracy: 0.7861
Epoch 91/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2545 - accuracy: 0.9
367 - val_loss: 1.3499 - val_accuracy: 0.7861
Epoch 92/200
51/51 [==============================] - 2s 30ms/step - loss: 0.2307 - accuracy: 0.9
280 - val_loss: 1.2389 - val_accuracy: 0.7889
Epoch 93/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1711 - accuracy: 0.9
443 - val_loss: 1.1992 - val_accuracy: 0.7981
Epoch 94/200
```

```
51/51 [==============================] - 1s 28ms/step - loss: 0.1655 - accuracy: 0.9
484 - val_loss: 1.1829 - val_accuracy: 0.8157
Epoch 95/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1528 - accuracy: 0.9
461 - val_loss: 1.2841 - val_accuracy: 0.8074
Epoch 96/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1697 - accuracy: 0.9
479 - val_loss: 1.1715 - val_accuracy: 0.8222
Epoch 97/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1770 - accuracy: 0.9
431 - val_loss: 1.1885 - val_accuracy: 0.8157
Epoch 98/200
51/51 [==============================] - 2s 31ms/step - loss: 0.2072 - accuracy: 0.9
427 - val_loss: 1.3472 - val_accuracy: 0.7917
Epoch 99/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1555 - accuracy: 0.9
513 - val_loss: 1.2568 - val_accuracy: 0.8185
Epoch 100/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1693 - accuracy: 0.9
437 - val_loss: 1.2629 - val_accuracy: 0.8056
Epoch 101/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1301 - accuracy: 0.9
596 - val_loss: 1.2853 - val_accuracy: 0.8019
Epoch 102/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1807 - accuracy: 0.9
463 - val_loss: 1.2554 - val_accuracy: 0.8074
Epoch 103/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1683 - accuracy: 0.9
461 - val_loss: 1.2378 - val_accuracy: 0.8037
Epoch 104/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1417 - accuracy: 0.9
519 - val_loss: 1.3345 - val_accuracy: 0.8009
Epoch 105/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1798 - accuracy: 0.9
451 - val_loss: 1.3639 - val_accuracy: 0.7833
Epoch 106/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1525 - accuracy: 0.9
540 - val_loss: 1.5179 - val_accuracy: 0.7731
Epoch 107/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1696 - accuracy: 0.9
462 - val_loss: 1.5167 - val_accuracy: 0.7833
Epoch 108/200
51/51 [==============================] - 1s 27ms/step - loss: 0.2040 - accuracy: 0.9
509 - val_loss: 1.4357 - val_accuracy: 0.8019
Epoch 109/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1475 - accuracy: 0.9
489 - val_loss: 1.3065 - val_accuracy: 0.8065
Epoch 110/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1420 - accuracy: 0.9
568 - val_loss: 1.3213 - val_accuracy: 0.8139
Epoch 111/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1361 - accuracy: 0.9
596 - val_loss: 1.2951 - val_accuracy: 0.8111
Epoch 112/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1315 - accuracy: 0.9
538 - val_loss: 1.3367 - val_accuracy: 0.8139
```

```
Epoch 113/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1226 - accuracy: 0.9
563 - val_loss: 1.3667 - val_accuracy: 0.8111
Epoch 114/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1235 - accuracy: 0.9
625 - val_loss: 1.2639 - val_accuracy: 0.8306
Epoch 115/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1200 - accuracy: 0.9
548 - val_loss: 1.3077 - val_accuracy: 0.8250
Epoch 116/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1209 - accuracy: 0.9
611 - val_loss: 1.3715 - val_accuracy: 0.8083
Epoch 117/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1023 - accuracy: 0.9
665 - val_loss: 1.4314 - val_accuracy: 0.8083
Epoch 118/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1166 - accuracy: 0.9
601 - val_loss: 1.3478 - val_accuracy: 0.8204
Epoch 119/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1072 - accuracy: 0.9
634 - val_loss: 1.3861 - val_accuracy: 0.8139
Epoch 120/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1098 - accuracy: 0.9
622 - val_loss: 1.5313 - val_accuracy: 0.8120
Epoch 121/200
51/51 [==============================] - 2s 31ms/step - loss: 0.0938 - accuracy: 0.9
688 - val_loss: 1.4636 - val_accuracy: 0.8130
Epoch 122/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1416 - accuracy: 0.9
572 - val_loss: 1.3680 - val_accuracy: 0.8185
Epoch 123/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1245 - accuracy: 0.9
553 - val_loss: 1.5490 - val_accuracy: 0.8065
Epoch 124/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1348 - accuracy: 0.9
612 - val_loss: 1.4595 - val_accuracy: 0.8009
Epoch 125/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1368 - accuracy: 0.9
627 - val_loss: 1.3861 - val_accuracy: 0.8046
Epoch 126/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1149 - accuracy: 0.9
618 - val_loss: 1.4295 - val_accuracy: 0.8120
Epoch 127/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1019 - accuracy: 0.9
646 - val_loss: 1.5757 - val_accuracy: 0.7991
Epoch 128/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1372 - accuracy: 0.9
586 - val_loss: 1.5597 - val_accuracy: 0.7954
Epoch 129/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1822 - accuracy: 0.9
531 - val_loss: 1.6330 - val_accuracy: 0.7815
Epoch 130/200
51/51 [==============================] - 1s 29ms/step - loss: 0.1799 - accuracy: 0.9
553 - val_loss: 1.4129 - val_accuracy: 0.8120
Epoch 131/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1567 - accuracy: 0.9
```

```
659 - val_loss: 1.5269 - val_accuracy: 0.7907
Epoch 132/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1762 - accuracy: 0.9
604 - val_loss: 1.4758 - val_accuracy: 0.7824
Epoch 133/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1815 - accuracy: 0.9
469 - val_loss: 1.4210 - val_accuracy: 0.7907
Epoch 134/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1404 - accuracy: 0.9
628 - val_loss: 1.5349 - val_accuracy: 0.7907
Epoch 135/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1378 - accuracy: 0.9
607 - val_loss: 1.3614 - val_accuracy: 0.8074
Epoch 136/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1098 - accuracy: 0.9
625 - val_loss: 1.3462 - val_accuracy: 0.8046
Epoch 137/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1014 - accuracy: 0.9
661 - val_loss: 1.4646 - val_accuracy: 0.7981
Epoch 138/200
51/51 [==============================] - 2s 30ms/step - loss: 0.0945 - accuracy: 0.9
654 - val_loss: 1.4178 - val_accuracy: 0.8167
Epoch 139/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0786 - accuracy: 0.9
774 - val_loss: 1.4451 - val_accuracy: 0.8167
Epoch 140/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0916 - accuracy: 0.9
678 - val_loss: 1.5284 - val_accuracy: 0.8157
Epoch 141/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0861 - accuracy: 0.9
672 - val_loss: 1.5091 - val_accuracy: 0.8213
Epoch 142/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0869 - accuracy: 0.9
669 - val_loss: 1.5188 - val_accuracy: 0.8213
Epoch 143/200
51/51 [==============================] - 1s 27ms/step - loss: 0.0946 - accuracy: 0.9
673 - val_loss: 1.5226 - val_accuracy: 0.8204
Epoch 144/200
51/51 [==============================] - 2s 31ms/step - loss: 0.0972 - accuracy: 0.9
629 - val_loss: 1.6258 - val_accuracy: 0.8120
Epoch 145/200
51/51 [==============================] - 1s 30ms/step - loss: 0.0964 - accuracy: 0.9
685 - val_loss: 1.5690 - val_accuracy: 0.8065
Epoch 146/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1393 - accuracy: 0.9
616 - val_loss: 1.4962 - val_accuracy: 0.7981
Epoch 147/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1127 - accuracy: 0.9
663 - val_loss: 1.5520 - val_accuracy: 0.8167
Epoch 148/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0872 - accuracy: 0.9
698 - val_loss: 1.5513 - val_accuracy: 0.8130
Epoch 149/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1091 - accuracy: 0.9
635 - val_loss: 1.5166 - val_accuracy: 0.8157
Epoch 150/200
```

```
51/51 [==============================] - 1s 27ms/step - loss: 0.1072 - accuracy: 0.9
669 - val_loss: 1.4524 - val_accuracy: 0.8185
Epoch 151/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1035 - accuracy: 0.9
643 - val_loss: 1.3744 - val_accuracy: 0.8176
Epoch 152/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0720 - accuracy: 0.9
731 - val_loss: 1.4774 - val_accuracy: 0.8130
Epoch 153/200
51/51 [==============================] - 1s 29ms/step - loss: 0.0824 - accuracy: 0.9
728 - val_loss: 1.5179 - val_accuracy: 0.8102
Epoch 154/200
51/51 [==============================] - 1s 28ms/step - loss: 0.2493 - accuracy: 0.9
483 - val_loss: 1.4644 - val_accuracy: 0.8019
Epoch 155/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1679 - accuracy: 0.9
573 - val_loss: 1.5371 - val_accuracy: 0.7898
Epoch 156/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1153 - accuracy: 0.9
681 - val_loss: 1.4343 - val_accuracy: 0.7898
Epoch 157/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1218 - accuracy: 0.9
707 - val_loss: 1.5666 - val_accuracy: 0.7944
Epoch 158/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1220 - accuracy: 0.9
721 - val_loss: 1.5928 - val_accuracy: 0.8028
Epoch 159/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1309 - accuracy: 0.9
685 - val_loss: 1.4778 - val_accuracy: 0.8120
Epoch 160/200
51/51 [==============================] - 1s 27ms/step - loss: 0.0994 - accuracy: 0.9
707 - val_loss: 1.4915 - val_accuracy: 0.8139
Epoch 161/200
51/51 [==============================] - 2s 30ms/step - loss: 0.1042 - accuracy: 0.9
662 - val_loss: 1.6537 - val_accuracy: 0.8019
Epoch 162/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0857 - accuracy: 0.9
717 - val_loss: 1.5764 - val_accuracy: 0.8083
Epoch 163/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0736 - accuracy: 0.9
781 - val_loss: 1.5630 - val_accuracy: 0.8167
Epoch 164/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1678 - accuracy: 0.9
688 - val_loss: 1.4536 - val_accuracy: 0.8157
Epoch 165/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0973 - accuracy: 0.9
752 - val_loss: 1.4181 - val_accuracy: 0.8222
Epoch 166/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1062 - accuracy: 0.9
727 - val_loss: 1.4752 - val_accuracy: 0.8194
Epoch 167/200
51/51 [==============================] - 2s 32ms/step - loss: 0.0857 - accuracy: 0.9
768 - val_loss: 1.5689 - val_accuracy: 0.8130
Epoch 168/200
51/51 [==============================] - 2s 30ms/step - loss: 0.0859 - accuracy: 0.9
754 - val_loss: 1.7020 - val_accuracy: 0.8083
```

```
Epoch 169/200
51/51 [==============================] - 1s 29ms/step - loss: 0.1044 - accuracy: 0.9
716 - val_loss: 1.5292 - val_accuracy: 0.8139
Epoch 170/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0956 - accuracy: 0.9
721 - val_loss: 1.4296 - val_accuracy: 0.8231
Epoch 171/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0800 - accuracy: 0.9
760 - val_loss: 1.6652 - val_accuracy: 0.8056
Epoch 172/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0834 - accuracy: 0.9
747 - val_loss: 1.6405 - val_accuracy: 0.8120
Epoch 173/200
51/51 [==============================] - 1s 27ms/step - loss: 0.0735 - accuracy: 0.9
745 - val_loss: 1.5848 - val_accuracy: 0.8139
Epoch 174/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0891 - accuracy: 0.9
783 - val_loss: 1.4611 - val_accuracy: 0.8194
Epoch 175/200
51/51 [==============================] - 1s 27ms/step - loss: 0.0696 - accuracy: 0.9
749 - val_loss: 1.5219 - val_accuracy: 0.8213
Epoch 176/200
51/51 [==============================] - 2s 30ms/step - loss: 0.0851 - accuracy: 0.9
714 - val_loss: 1.5370 - val_accuracy: 0.8157
Epoch 177/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0797 - accuracy: 0.9
768 - val_loss: 1.6197 - val_accuracy: 0.8102
Epoch 178/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0969 - accuracy: 0.9
791 - val_loss: 1.5826 - val_accuracy: 0.8074
Epoch 179/200
51/51 [==============================] - 1s 27ms/step - loss: 0.0854 - accuracy: 0.9
734 - val_loss: 1.6410 - val_accuracy: 0.8009
Epoch 180/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0910 - accuracy: 0.9
699 - val_loss: 1.7419 - val_accuracy: 0.8037
Epoch 181/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0938 - accuracy: 0.9
701 - val_loss: 1.8294 - val_accuracy: 0.8000
Epoch 182/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1101 - accuracy: 0.9
736 - val_loss: 1.7601 - val_accuracy: 0.7963
Epoch 183/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1250 - accuracy: 0.9
641 - val_loss: 1.6687 - val_accuracy: 0.8167
Epoch 184/200
51/51 [==============================] - 1s 29ms/step - loss: 0.1072 - accuracy: 0.9
748 - val_loss: 1.7506 - val_accuracy: 0.8000
Epoch 185/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1518 - accuracy: 0.9
691 - val_loss: 1.6729 - val_accuracy: 0.7963
Epoch 186/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1107 - accuracy: 0.9
723 - val_loss: 1.7241 - val_accuracy: 0.7954
Epoch 187/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1041 - accuracy: 0.9
```

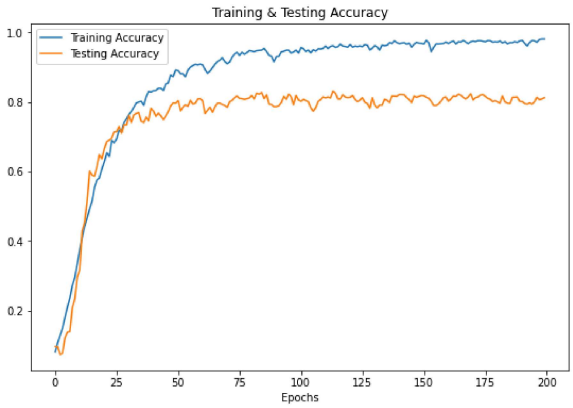
```
723 - val_loss: 1.6151 - val_accuracy: 0.8130
Epoch 188/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1281 - accuracy: 0.9
713 - val_loss: 1.5464 - val_accuracy: 0.8130
Epoch 189/200
51/51 [==============================] - 1s 27ms/step - loss: 0.0801 - accuracy: 0.9
728 - val_loss: 1.6292 - val_accuracy: 0.8139
Epoch 190/200
51/51 [==============================] - 2s 32ms/step - loss: 0.0805 - accuracy: 0.9
735 - val_loss: 1.6332 - val_accuracy: 0.8019
Epoch 191/200
51/51 [==============================] - 2s 30ms/step - loss: 0.0773 - accuracy: 0.9
815 - val_loss: 1.5902 - val_accuracy: 0.8009
Epoch 192/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1051 - accuracy: 0.9
705 - val_loss: 1.5614 - val_accuracy: 0.7954
Epoch 193/200
51/51 [==============================] - 1s 28ms/step - loss: 0.1664 - accuracy: 0.9
609 - val_loss: 1.4564 - val_accuracy: 0.7935
Epoch 194/200
51/51 [==============================] - 1s 27ms/step - loss: 0.1343 - accuracy: 0.9
648 - val_loss: 1.5770 - val_accuracy: 0.7972
Epoch 195/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0830 - accuracy: 0.9
714 - val_loss: 1.5972 - val_accuracy: 0.7935
Epoch 196/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0762 - accuracy: 0.9
767 - val_loss: 1.5137 - val_accuracy: 0.8000
Epoch 197/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0928 - accuracy: 0.9
676 - val_loss: 1.5008 - val_accuracy: 0.8130
Epoch 198/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0619 - accuracy: 0.9
821 - val_loss: 1.5398 - val_accuracy: 0.8065
Epoch 199/200
51/51 [==============================] - 2s 30ms/step - loss: 0.0654 - accuracy: 0.9
780 - val_loss: 1.5799 - val_accuracy: 0.8083
Epoch 200/200
51/51 [==============================] - 1s 28ms/step - loss: 0.0520 - accuracy: 0.9
865 - val_loss: 1.5817 - val_accuracy: 0.8120
```

In [33]:
```
# computing the accuracy
print("Accuracy of our model on test data : " , model.evaluate(x_testcnn,y_test)[1]
```

```
34/34 [==============================] - 0s 7ms/step - loss: 1.5817 - accuracy: 0.81
20
Accuracy of our model on test data :  81.20370507240295 %
```

In [34]:
```
# training and test accuracy vs epochs plot
epochs = [i for i in range(200)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
test_acc = history.history['val_accuracy']
test_loss = history.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()
```



PREDICTION

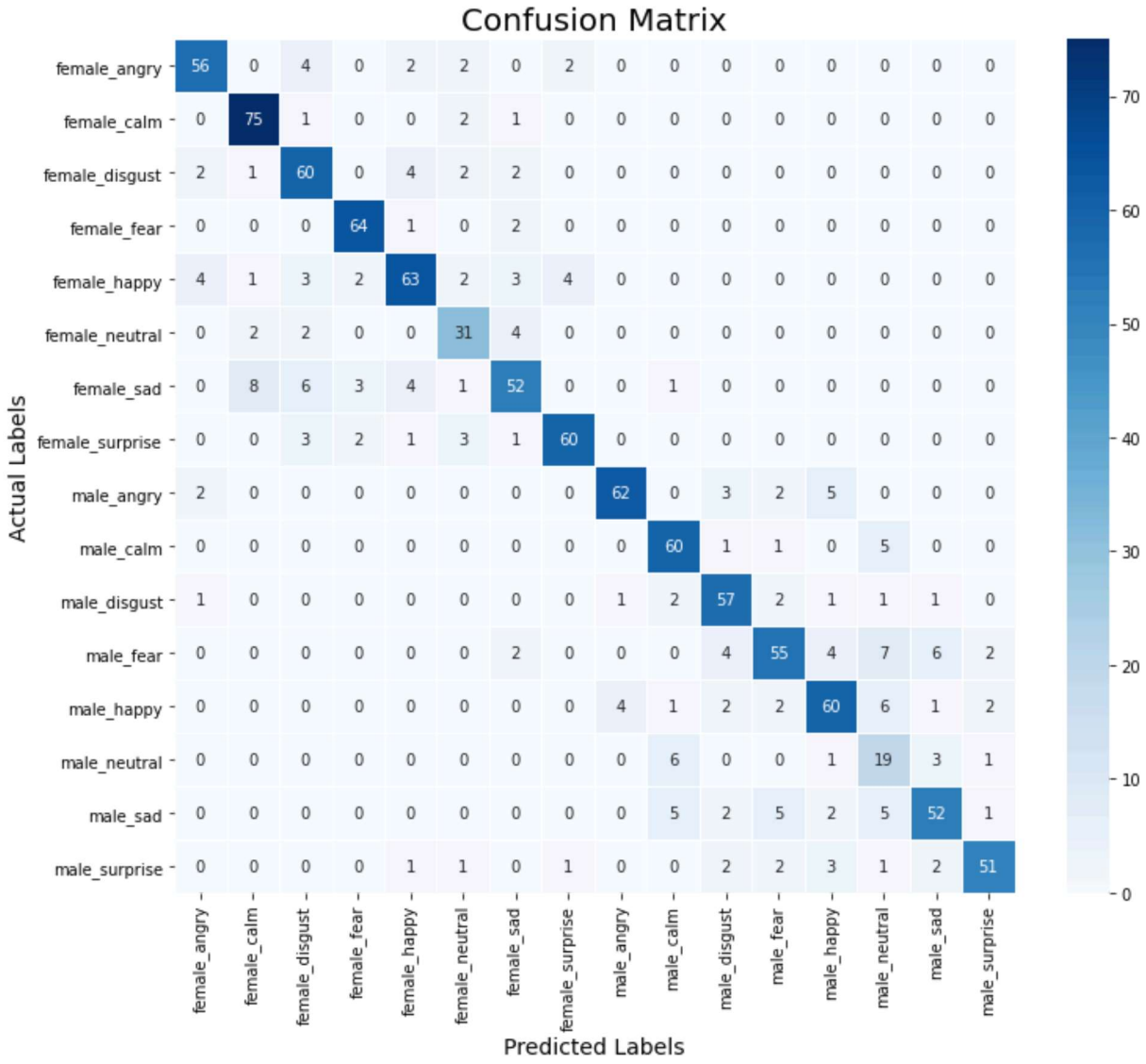
In [35]:
```
pred_test = model.predict(x_testcnn)
y_pred = encoder.inverse_transform(pred_test)

y_test = encoder.inverse_transform(y_test)

df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df['Predicted Labels'] = y_pred.flatten()
df['Actual Labels'] = y_test.flatten()

df.head(10)
```

Out[35]:

	Predicted Labels	Actual Labels
0	female_angry	female_angry
1	male_sad	male_sad
2	female_surprise	female_angry
3	female_calm	female_calm
4	female_calm	female_calm
5	female_surprise	female_surprise
6	male_sad	male_sad
7	male_happy	male_happy
8	female_disgust	female_disgust
9	male_fear	male_fear

In [36]:

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (12, 10))
cm = pd.DataFrame(cm , index = [i for i in encoder.categories_] , columns = [i for
sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True, fmt='')
plt.title('Confusion Matrix', size=20)
plt.xlabel('Predicted Labels', size=14)
plt.ylabel('Actual Labels', size=14)
plt.show()
```



SAVING THE MODEL

In [37]:
```
model_name = 'Emotion_Voice_Detection_Model.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 /kaggle/working/saved_models/Emotion_Voice_Detection_Model.h5
```

In [38]:
```
import json
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

In [39]:
```
from keras.models import model_from_json
json_file = open('model.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("/kaggle/working/saved_models/Emotion_Voice_Detection_Mod
print("Loaded model from disk")
```

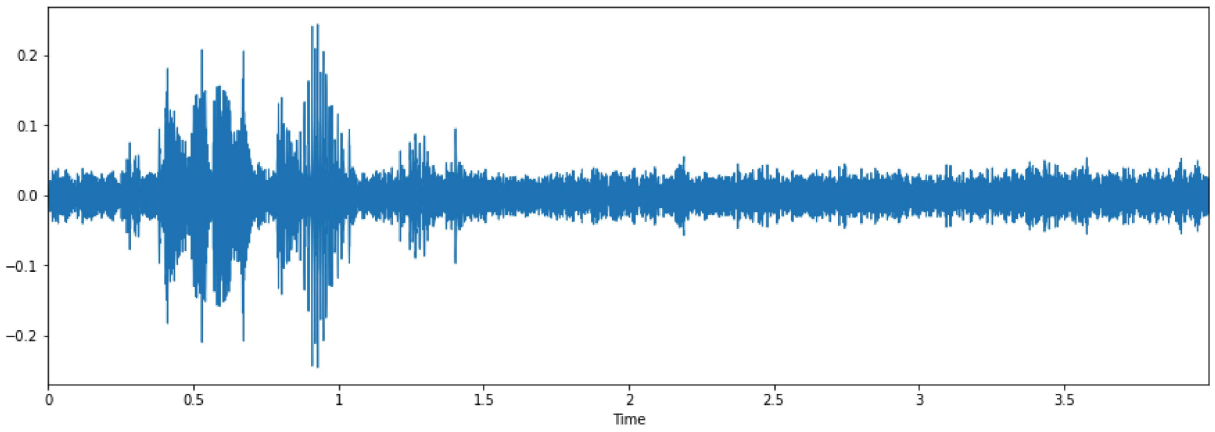
```
Loaded model from disk
```

LIVE DEMO

In [40]:
```
data, sampling_rate = librosa.load('../input/output/output10.wav')
```

In [42]:
```
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
```

Out[42]:
```
<matplotlib.collections.PolyCollection at 0x7f60842ecfd0>
```



In [43]:
```
#livedf= pd.DataFrame(columns=['feature'])
X, sample_rate = librosa.load('../input/output/output10.wav', res_type='kaiser_fast
sample_rate = np.array(sample_rate)
mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),axis=0)
featurelive = mfccs
livedf2 = featurelive
```

In [44]:
```
livedf2= pd.DataFrame(data=livedf2)
livedf2 = livedf2.stack().to_frame().T
livedf2
```

Out[44]:

	0	1	2	3	4	5	6	
	0	0	0	0	0	0	0	
0	-10.596061	-8.946511	-10.528728	-13.296783	-14.177729	-13.25804	-13.502738	-11.362(

1 rows × 216 columns

In [45]:
```
twodim= np.expand_dims(livedf2, axis=2)
```

In [46]:
```
livepreds = loaded_model.predict(twodim,
                         batch_size=32,
                         verbose=1)
```

```
1/1 [==============================] - 1s 957ms/step
```

In [47]:
```
livepreds
```

Out[47]:
```
array([[3.2250948e-06, 3.6100307e-03, 2.3342846e-03, 2.2406097e-05,
        1.7844244e-04, 9.2410314e-01, 6.9004118e-02, 1.7600438e-05,
        2.7654128e-06, 3.0049527e-04, 2.5601514e-06, 1.5482929e-06,
        4.4414824e-07, 7.9448392e-07, 6.9831702e-05, 3.4829479e-04]],
      dtype=float32)
```

In [48]:
```
livepreds.shape
```

Out[48]:
```
(1, 16)
```

In [49]:
```
livepredictions = (encoder.inverse_transform((livepreds)))
print(livepredictions)
```

```
[['female_neutral']]
```

COMPONENT 2: MOVIE RECOMMENDATION SYSTEM

In [50]:
```
livepredictions=livepredictions.tolist()
```

In [51]:
```
string_version = " ".join(str(x) for x in livepredictions)
print(string_version)
```

```
['female_neutral']
```

In [52]:
```
# the emotion detected previously from live demo is stored in 'emotion' variable wh
emotion=string_version.split("'")
emotion=emotion[1].split("_")
gender=emotion[0]
emotion=emotion[1]
```

In [53]:
```
pip install bs4
```

```
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.10.0-py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 669 kB/s eta 0:00:01
Collecting soupsieve>1.2
  Downloading soupsieve-2.2.1-py3-none-any.whl (33 kB)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) ... done
  Created wheel for bs4: filename=bs4-0.0.1-py3-none-any.whl size=1273 sha256=3c34cb
274e6553664b9a92287b52995355befeeb89dc4cb667002860ce5025c0
  Stored in directory: /root/.cache/pip/wheels/0a/9e/ba/20e5bbc1afef3a491f0b3bb74d50
8f99403aabe76eda2167ca
Successfully built bs4
Installing collected packages: soupsieve, beautifulsoup4, bs4
Successfully installed beautifulsoup4-4.10.0 bs4-0.0.1 soupsieve-2.2.1
Note: you may need to restart the kernel to use updated packages.
```

IMPORTING LIBRARIES

In [54]:
```
from bs4 import BeautifulSoup as SOUP
import re
import requests as HTTP
```

LIST OF LINKS FOR DIFFERENT GENRES

We have conducted a survey and obtained following results for male and female preference

Emotion	Male	Female
Sad	Drama	Drama/ Fantasy
Happy	Adventure/Mystery	Rom-Com/ Adventure
Fearful	Family	Family/ Game-Show
Surprise	Action-Comedy	Action-Comedy
Neutral	War/History	Adventure
Calm	Documentary/Biography	Musical
Disgust	Horror	Horror/Film-Noir
Angry	Action/Thriller	Action/Thriller

In [55]:
```
male_genre=["https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_seri
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_serie
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_serie
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_serie
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_serie
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_serie
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_serie
```

In [56]:
```
female_genre=["https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_se
              "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
              "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
              "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
```

```
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
            "https://www.imdb.com/search/title/?title_type=feature,tv_movie,tv_ser
```

In [57]:

```
if(gender == "male"):
    # IMDb Url for movie against emotion Neutral
    if(emotion == "neutral"):
        urlhere = male_genre[0]

    # IMDb Url for movie against emotion Happy
    elif(emotion == "happy"):
        urlhere = male_genre[1]

    # IMDb Url for movie against emotion Sad
    elif(emotion == "sad"):
        urlhere = male_genre[2]

    # IMDb Url for movie against emotion Calm
    elif(emotion == "calm"):
        urlhere = male_genre[3]

    # IMDb Url for movie against emotion Surprised
    elif(emotion == "surprised"):
        urlhere = male_genre[4]

    # IMDb Url for movie against emotion Angry
    elif(emotion == "angry"):
        urlhere = male_genre[5]

    # IMDb Url for  movie against emotion Fearful
    elif(emotion == "fear"):
        urlhere = male_genre[6]

    # IMDb Url for movie against emotion Disgust
    elif(emotion == "disgust"):
        urlhere = male_genre[7]

elif(gender=="female"):
    if(emotion == "neutral"):
        urlhere = female_genre[0]

    # IMDb Url for movie against emotion Happy
    elif(emotion == "happy"):
        urlhere = female_genre[1]

    # IMDb Url for movie against emotion Sad
    elif(emotion == "sad"):
        urlhere = female_genre[2]

    # IMDb Url for movie against emotion Calm
    elif(emotion == "calm"):
        urlhere = female_genre[3]

    # IMDb Url for movie against emotion Surprised
    elif(emotion == "surprised"):
```

```
        urlhere = female_genre[4]

    # IMDb Url for movie against emotion Angry
    elif(emotion == "angry"):
        urlhere = female_genre[5]

    # IMDb Url for movie against emotion Fearful
    elif(emotion == "fear"):
        urlhere = female_genre[6]

    # IMDb Url for movie against emotion Disgust
    elif(emotion == "disgust"):
        urlhere = female_genre[7]
```

In [58]:
```
response = HTTP.get(urlhere)
data = response.text
soup = SOUP(data, "lxml")
```

SCRAPING THE TITLE OF THE MOVIE FROM THE IMBb WEBSITE

In [59]:
```
vbLf='\n'
TITLES = []
titles = soup.find_all('h3')
for t in titles:
      TITLES.append(t.text[4:].replace(vbLf,""))
```

SCRAPING THE URLs CORRESPONDING TO EACH MOVIE

In [60]:
```
URL_list = []
for item in soup.find_all(attrs={'class':'lister-item-header'}):
    for link in item.find_all('a',href=True):
        href=link.get('href')
        URL_list.append("https://www.imdb.com/"+href+"?ref_=adv_li_tt")
```

In [61]:
```
TITLES
```

Out[61]:
```
['Squid Game(2021)',
 'No Time to Die(2021)',
 'Dune(2021)',
 'What If...?(2021– )',
 'Shang-Chi and the Legend of the Ten Rings(2021)',
 'Doom Patrol(2019– )',
 'Titans(I) (2018– )',
 'The Witcher(2019– )',
 'The Mandalorian(2019– )',
 'Loki(2021– )',
 'The Orville(2017– )',
 'Avengers: Endgame(2019)',
 'Boku no hîrô akademia(2016– )',
 'Killing Eve(2018–2022)',
 'Final Space(2018–2021)',
 'The Umbrella Academy(2019– )',
 "Zack Snyder's Justice League(2021)",
 'Superman and Lois(2021– )',
 'Invincible(2021– )',
 'Thor: Ragnarok(2017)',
 'Luca(2021)',
 'Demon Slayer: Kimetsu No Yaiba(2019– )',
 'Shadow and Bone(2021– )',
 'Deadpool 2(2018)',
 'Avengers: Infinity War(2018)',
 'Deadpool(2016)',
 'Star Trek: Picard(2020– )',
 'His Dark Materials(2019– )',
 'Spider-Man: Into the Spider-Verse(2018)',
 'Sweet Tooth(2021– )',
 'Guardians of the Galaxy Vol. 2(2017)',
 'Love, Death & Robots(2019– )',
 'The End of the F***ing World(2017–2019)',
 'Star Wars: The Bad Batch(2021– )',
 'Rogue One(2016)',
 'Black Clover(2017–2021)',
 'Coco(I) (2017)',
 'Doctor Strange(2016)',
 'Jujutsu Kaisen(2020– )',
 'Harley Quinn(2019– )',
 'Vaiana(I) (2016)',
 'Soul(2020)',
 'Castlevania(2017–2021)',
 'Kimetsu no Yaiba: Mugen Ressha-Hen(2020)',
 'The Terror(2018–2019)',
 "Barbaroslar: Akdeniz'in Kilici(2021– )",
 'Zootopia(2016)',
 'Captain America: Civil War(2016)',
 'Mission: Impossible - Fallout(2018)',
 'The Owl House(2020– )',
 'ntly Viewed']
```

SCRAPPING THE RATINGS OF EACH MOVIE

In [62]:
```
RATINGS = []
ratings =soup.find_all("div",{'class':"inline-block ratings-imdb-rating"})
for r in ratings:
    RATINGS.append(r.text.replace(vbLf,""))
RATINGS
```

Out[62]:
```
['8.3',
 '7.6',
 '8.4',
 '7.6',
 '7.9',
 '7.9',
 '7.6',
 '8.2',
 '8.8',
 '8.4',
 '8.0',
 '8.4',
 '8.4',
 '8.2',
 '8.3',
 '8.0',
 '8.1',
 '7.9',
 '8.7',
 '7.9',
 '7.5',
 '8.7',
 '7.7',
 '7.7',
 '8.4',
 '8.0',
 '7.5',
 '7.9',
 '8.4',
 '7.9',
 '7.6',
 '8.5',
 '8.1',
 '8.0',
 '7.8',
 '8.3',
 '8.4',
 '7.5',
 '8.7',
 '8.5',
 '7.6',
 '8.1',
 '8.3',
 '8.3',
 '7.9',
 '8.8',
 '8.0',
 '7.8',
 '7.7',
 '8.1']
```

In [63]:
```
# created the dataframe to store the scrapped information
data = pd.DataFrame(zip(TITLES, RATINGS, URL_list), columns = ["Title", "Ratings",
```

In [64]:
```
data
```

Out[64]:

	Title	Ratings	URL
0	Squid Game(2021)	8.3	https://www.imdb.com//title/tt10919420/?ref_=a...
1	No Time to Die(2021)	7.6	https://www.imdb.com//title/tt2382320/?ref_=ad...
2	Dune(2021)	8.4	https://www.imdb.com//title/tt1160419/?ref_=ad...
3	What If...?(2021–)	7.6	https://www.imdb.com//title/tt10168312/?ref_=a...
4	Shang-Chi and the Legend of the Ten Rings(2021)	7.9	https://www.imdb.com//title/tt9376612/?ref_=ad...
5	Doom Patrol(2019–)	7.9	https://www.imdb.com//title/tt8416494/?ref_=ad...
6	Titans(I) (2018–)	7.6	https://www.imdb.com//title/tt1043813/?ref_=ad...
7	The Witcher(2019–)	8.2	https://www.imdb.com//title/tt5180504/?ref_=ad...
8	The Mandalorian(2019–)	8.8	https://www.imdb.com//title/tt8111088/?ref_=ad...
9	Loki(2021–)	8.4	https://www.imdb.com//title/tt9140554/?ref_=ad...
10	The Orville(2017–)	8.0	https://www.imdb.com//title/tt5691552/?ref_=ad...
11	Avengers: Endgame(2019)	8.4	https://www.imdb.com//title/tt4154796/?ref_=ad...
12	Boku no hîrô akademia(2016–)	8.4	https://www.imdb.com//title/tt5626028/?ref_=ad...
13	Killing Eve(2018–2022)	8.2	https://www.imdb.com//title/tt7016936/?ref_=ad...
14	Final Space(2018–2021)	8.3	https://www.imdb.com//title/tt6317068/?ref_=ad...
15	The Umbrella Academy(2019–)	8.0	https://www.imdb.com//title/tt1312171/?ref_=ad...
16	Zack Snyder's Justice League(2021)	8.1	https://www.imdb.com//title/tt12361974/?ref_=a...
17	Superman and Lois(2021–)	7.9	https://www.imdb.com//title/tt11192306/?ref_=a...
18	Invincible(2021–)	8.7	https://www.imdb.com//title/tt6741278/?ref_=ad...

	Title	Ratings	URL
19	Thor: Ragnarok(2017)	7.9	https://www.imdb.com//title/tt3501632/?ref_=ad...
20	Luca(2021)	7.5	https://www.imdb.com//title/tt12801262/?ref_=a...
21	Demon Slayer: Kimetsu No Yaiba(2019–)	8.7	https://www.imdb.com//title/tt9335498/?ref_=ad...
22	Shadow and Bone(2021–)	7.7	https://www.imdb.com//title/tt2403776/?ref_=ad...
23	Deadpool 2(2018)	7.7	https://www.imdb.com//title/tt5463162/?ref_=ad...
24	Avengers: Infinity War(2018)	8.4	https://www.imdb.com//title/tt4154756/?ref_=ad...
25	Deadpool(2016)	8.0	https://www.imdb.com//title/tt1431045/?ref_=ad...
26	Star Trek: Picard(2020–)	7.5	https://www.imdb.com//title/tt8806524/?ref_=ad...
27	His Dark Materials(2019–)	7.9	https://www.imdb.com//title/tt5607976/?ref_=ad...
28	Spider-Man: Into the Spider-Verse(2018)	8.4	https://www.imdb.com//title/tt4633694/?ref_=ad...
29	Sweet Tooth(2021–)	7.9	https://www.imdb.com//title/tt12809988/?ref_=a...
30	Guardians of the Galaxy Vol. 2(2017)	7.6	https://www.imdb.com//title/tt3896198/?ref_=ad...
31	Love, Death & Robots(2019–)	8.5	https://www.imdb.com//title/tt9561862/?ref_=ad...
32	The End of the F***ing World(2017–2019)	8.1	https://www.imdb.com//title/tt6257970/?ref_=ad...
33	Star Wars: The Bad Batch(2021–)	8.0	https://www.imdb.com//title/tt12708542/?ref_=a...
34	Rogue One(2016)	7.8	https://www.imdb.com//title/tt3748528/?ref_=ad...
35	Black Clover(2017–2021)	8.3	https://www.imdb.com//title/tt7441658/?ref_=ad...
36	Coco(I) (2017)	8.4	https://www.imdb.com//title/tt2380307/?ref_=ad...
37	Doctor Strange(2016)	7.5	https://www.imdb.com//title/tt1211837/?ref_=ad...

	Title	Ratings	URL
38	Jujutsu Kaisen(2020–)	8.7	https://www.imdb.com//title/tt12343534/?ref_=a...
39	Harley Quinn(2019–)	8.5	https://www.imdb.com//title/tt7658402/?ref_=ad...
40	Vaiana(I) (2016)	7.6	https://www.imdb.com//title/tt3521164/?ref_=ad...
41	Soul(2020)	8.1	https://www.imdb.com//title/tt2948372/?ref_=ad...
42	Castlevania(2017–2021)	8.3	https://www.imdb.com//title/tt6517102/?ref_=ad...
43	Kimetsu no Yaiba: Mugen Ressha-Hen(2020)	8.3	https://www.imdb.com//title/tt11032374/?ref_=a...
44	The Terror(2018–2019)	7.9	https://www.imdb.com//title/tt2708480/?ref_=ad...
45	Barbaroslar: Akdeniz'in Kilici(2021–)	8.8	https://www.imdb.com//title/tt14473896/?ref_=a...
46	Zootopia(2016)	8.0	https://www.imdb.com//title/tt2948356/?ref_=ad...
47	Captain America: Civil War(2016)	7.8	https://www.imdb.com//title/tt3498820/?ref_=ad...
48	Mission: Impossible - Fallout(2018)	7.7	https://www.imdb.com//title/tt4912910/?ref_=ad...
49	The Owl House(2020–)	8.1	https://www.imdb.com//title/tt8050756/?ref_=ad...

In [ ]: