Make the necessary imports:

```
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
import librosa
import librosa.display
from IPython.display import Audio
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import keras
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization

from keras.callbacks import ModelCheckpoint
```

Mount on google

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Load the Dataset

For this Python project, we'll use the 24 Actors dataset; this is the Ryerson Audio-Visual Database of Emotional Speech and Song dataset, and is free to download. This dataset has 7356 files rated by 247 individuals 10 times on emotional validity, intensity, and genuineness. The entire dataset is 24.8GB from 24 actors, but we've lowered the sample rate on all the files

```
paths = []
labels = []
for dirname, _, filenames in os.walk('/content/drive/MyDrive/Task-2/Tess'):
    for filename in filenames:
        paths.append(os.path.join(dirname, filename))
        label = filename.split('_')[-1]
        label = label.split('.')[0]
        labels.append(label.lower())
    if len(paths) == 2800:
        break
print('Dataset is Loaded')
```

```
Dataset is Loaded
```

```
print(len(paths))
print(paths[:5])
print(labels[:5])
```

```
2800
['/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF_bone_sad.wav', '/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF_bar_sad.wav', '/cont
['sad', 'sad', 'sad', 'sad', 'sad']
```

```
# Paths for data.
Ravdess = "/content/drive/MyDrive/Task-2/speech-emotion-recognition-ravdess-data"
Crema ="/content/drive/MyDrive/Task-2/Crema"
Tess = "/content/drive/MyDrive/Task-2/Tess"
Savee = "/content/drive/MyDrive/Task-2/Savee"
```

Load the Dataset

```
## Create a dataframe
df = pd.DataFrame()
df['speech'] = paths
df['label'] = labels
df.head()
```

	speech	label
0	/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF...	sad
1	/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF...	sad
2	/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF...	sad
3	/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF...	sad
4	/content/drive/MyDrive/Task-2/Tess/YAF_sad/YAF...	sad

Next steps: Generate code with df View recommended plots

```
df['label'].value_counts()
```

```
sad        400
ps         400
fear       400
happy      400
neutral    400
angry      400
disgust    400
Name: label, dtype: int64
```

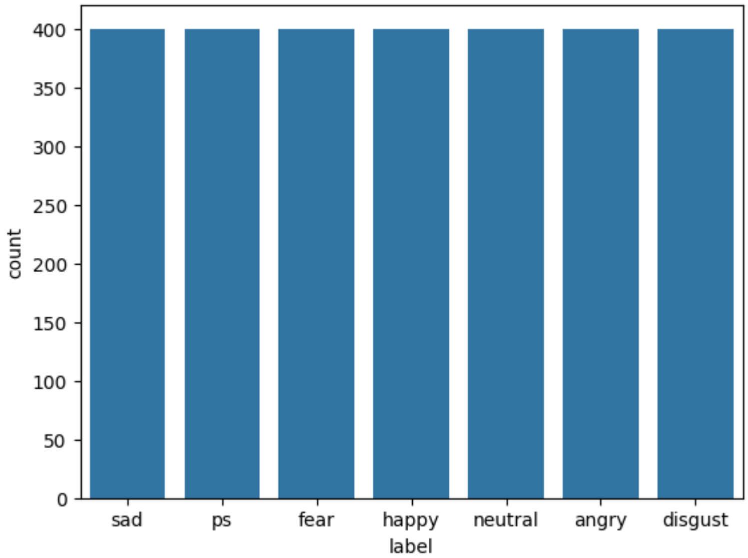
```
df.label.unique()
```

```
array(['sad', 'ps', 'fear', 'happy', 'neutral', 'angry', 'disgust'],
      dtype=object)
```

Exploratory Data Analysis

```
sns.countplot(data=df, x='label')
```

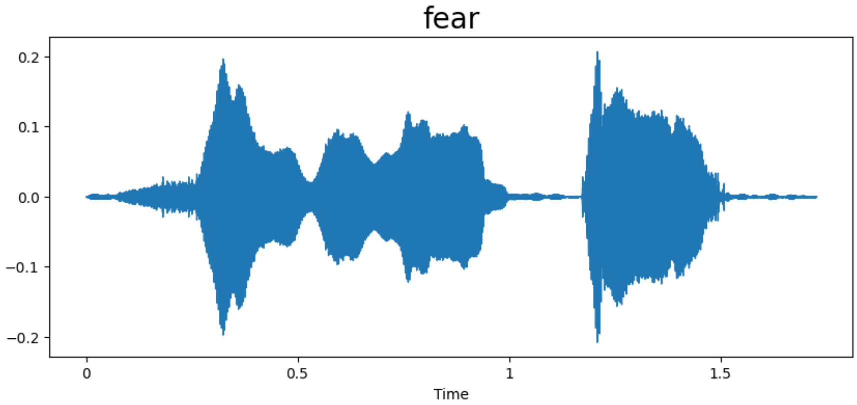
<Axes: xlabel='label', ylabel='count'>



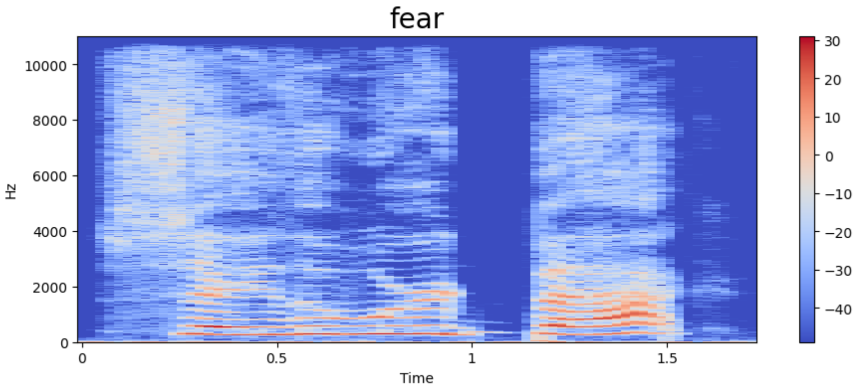

```
def waveplot(data, sr, emotion):
    plt.figure(figsize=(10,4))
    plt.title(emotion, size=20)
    librosa.display.waveshow(data, sr=sr)
    plt.show()

def spectogram(data, sr, emotion):
    x = librosa.stft(data)
    xdb = librosa.amplitude_to_db(abs(x))
    plt.figure(figsize=(11,4))
    plt.title(emotion, size=20)
    librosa.display.specshow(xdb, sr=sr, x_axis='time', y_axis='hz')
    plt.colorbar()
```

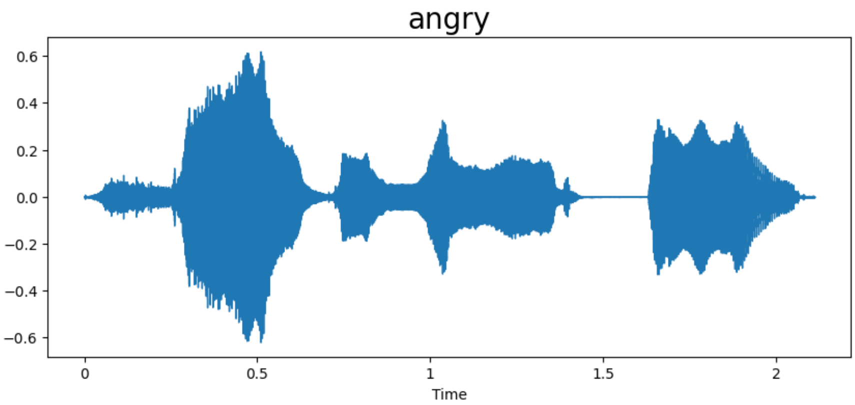
```
emotion = 'fear'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



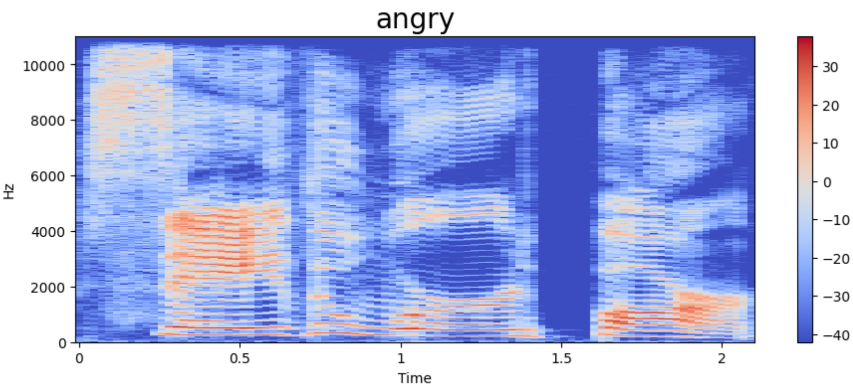
0:00 / 0:01



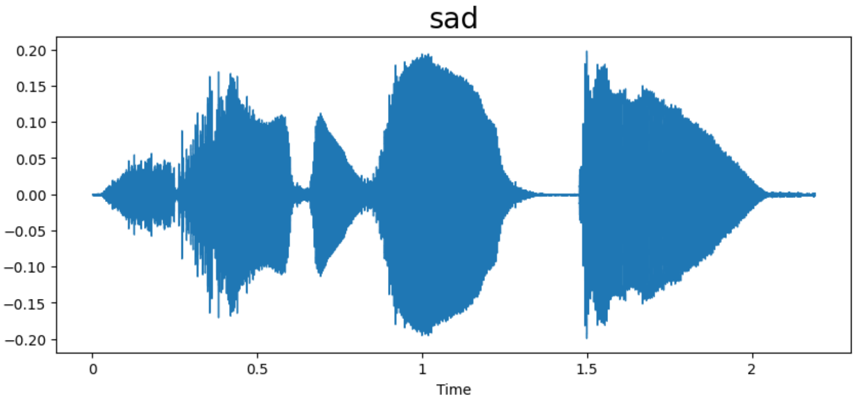

```
emotion = 'angry'
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



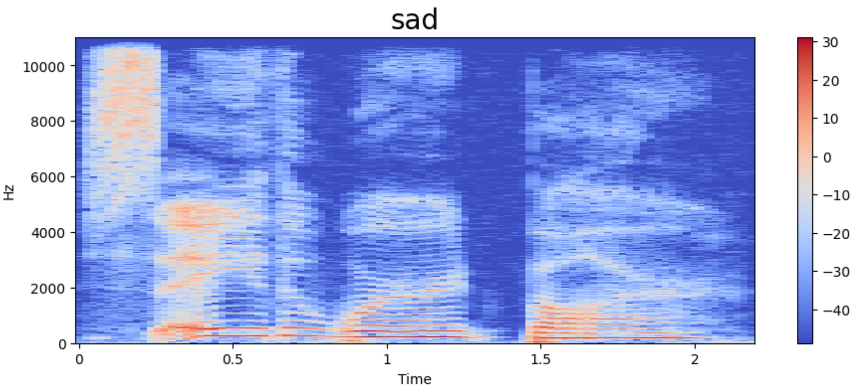
0:00 / 0:02



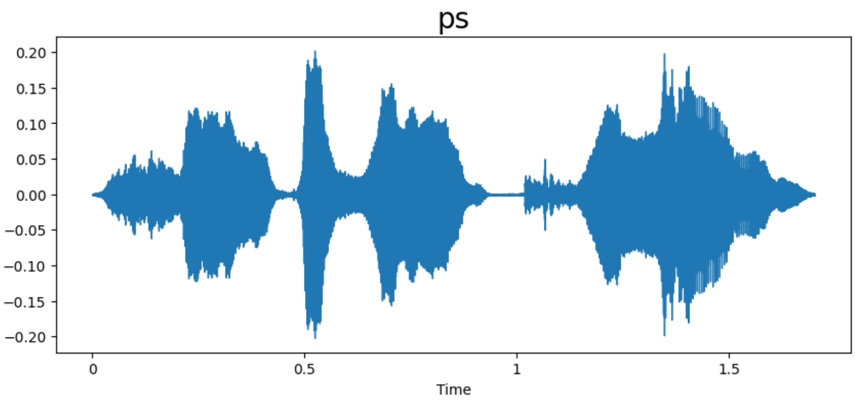

```
emotion = 'sad'
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



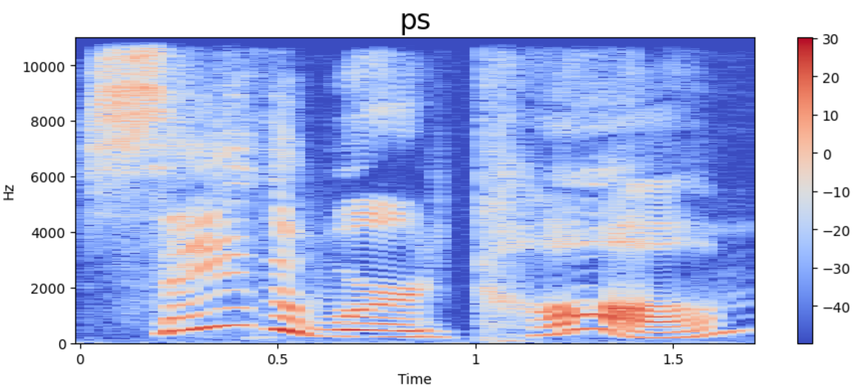
0:00 / 0:02



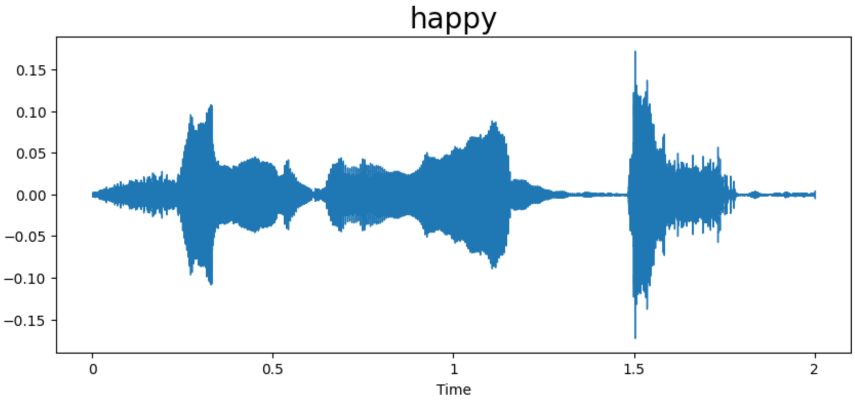

```
emotion = 'ps'
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



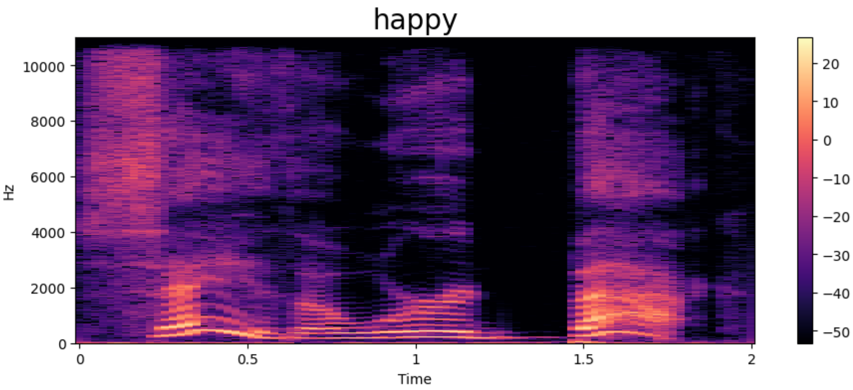
0:00 / 0:01



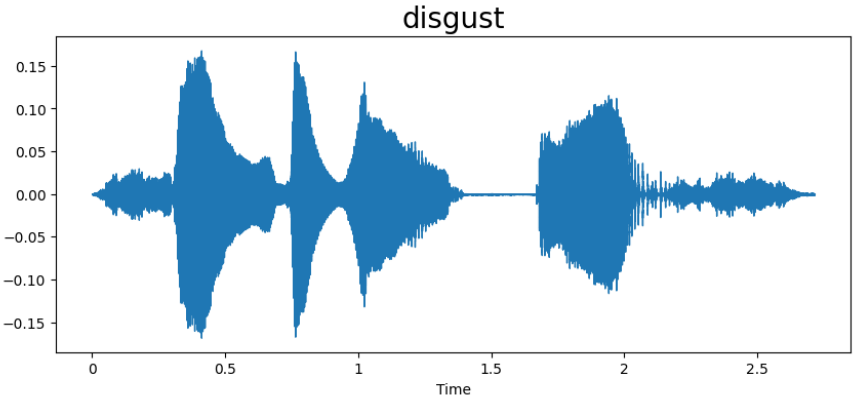

```
emotion = 'happy'
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



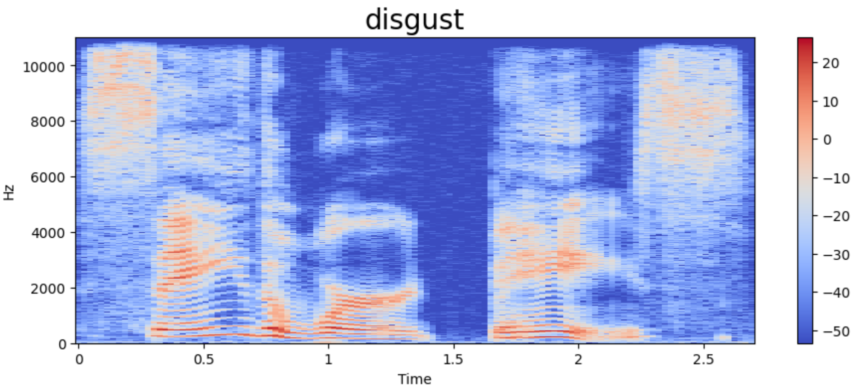
0:00 / 0:02




```
emotion = 'disgust'
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



0:00 / 0:02



mfcc: Mel Frequency Cepstral Coefficient, represents the short-term power spectrum of a sound

```
def extract_mfcc(filename):
    y, sr = librosa.load(filename, duration=3, offset=0.5)
    mfcc = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40).T, axis=0)
    return mfcc
```

```
extract_mfcc(df['speech'][0])
```

```
array([-4.27889557e+02,  1.11618225e+02,  3.15030766e+01,  3.18502483e+01,
        4.45114040e+00,  1.62448082e+01,  6.74070930e+00, -1.65567245e+01,
        5.66905451e+00, -7.20294094e+00, -5.17142057e+00, -2.20476389e+00,
       -8.94921112e+00,  9.00503349e+00, -6.54508305e+00,  6.90415084e-01,
        4.18817848e-01, -4.89057302e+00, -3.60782075e+00, -5.82673311e+00,
       -5.94607735e+00, -8.06733894e+00, -8.47742939e+00,  4.22039986e+00,
       -1.69765019e+00,  6.72037077e+00, -1.59601188e+00, -4.84941196e+00,
       -3.47141099e+00, -3.58782125e+00, -1.62143731e+00,  1.18874550e+01,
        7.91357088e+00,  9.63916206e+00,  5.96135759e+00,  6.07998800e+00,
        6.57818747e+00,  7.85023165e+00,  1.04019518e+01,  1.05001049e+01],
      dtype=float32)
```

```
X_mfcc = df['speech'].apply(lambda x: extract_mfcc(x))
# X_mfcc
```

```
X = [x for x in X_mfcc]
X = np.array(X)
X.shape
```

```
(2800, 40)
```

Input split

```
X = np.expand_dims(X, -1)
X.shape
```

```
(2800, 40, 1)
```

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
y = enc.fit_transform(df[['label']])
```

```
y = y.toarray()
y.shape
```

```
(2800, 7)
```

splitting data

```
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((2100, 40, 1), (2100, 7), (700, 40, 1), (700, 7))
```

Seqential Model:

```
model=Sequential()
model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu', input_shape=(x_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(128, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))
model.add(Dropout(0.2))

model.add(Conv1D(64, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Flatten())
model.add(Dense(units=32, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(units=7, activation='softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])

model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_12 (Conv1D)          (None, 40, 256)           1536

 max_pooling1d_14 (MaxPooli  (None, 20, 256)           0
 ng1D)

 conv1d_13 (Conv1D)          (None, 20, 256)           327936

 max_pooling1d_15 (MaxPooli  (None, 10, 256)           0
 ng1D)

 conv1d_14 (Conv1D)          (None, 10, 128)           163968

 max_pooling1d_16 (MaxPooli  (None, 5, 128)            0
 ng1D)

 dropout_12 (Dropout)        (None, 5, 128)            0

 conv1d_15 (Conv1D)          (None, 5, 64)             41024

 max_pooling1d_17 (MaxPooli  (None, 3, 64)             0
```

```
 ng1D)

 flatten_3 (Flatten)         (None, 192)               0

 dense_12 (Dense)            (None, 32)                6176

 dropout_13 (Dropout)        (None, 32)                0

 dense_13 (Dense)            (None, 7)                 231

=================================================================
Total params: 540871 (2.06 MB)
Trainable params: 540871 (2.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
rlrp = ReduceLROnPlateau(monitor='loss', factor=0.4, verbose=0, patience=2, min_lr=0.0000001)
history=model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), callbacks=[rlrp])
```

```
33/33 [==============================] - 4s 134ms/step - loss: 0.0654 - accuracy: 0.9757 - val_loss: 0.0261 - val_accuracy: 0.992
Epoch 23/50
33/33 [==============================] - 4s 134ms/step - loss: 0.0589 - accuracy: 0.9814 - val_loss: 0.0322 - val_accuracy: 0.991
Epoch 24/50
33/33 [==============================] - 6s 182ms/step - loss: 0.0581 - accuracy: 0.9805 - val_loss: 0.0168 - val_accuracy: 0.991
Epoch 25/50
33/33 [==============================] - 5s 137ms/step - loss: 0.0747 - accuracy: 0.9752 - val_loss: 0.0407 - val_accuracy: 0.988
Epoch 26/50
33/33 [==============================] - 5s 152ms/step - loss: 0.0638 - accuracy: 0.9776 - val_loss: 0.0740 - val_accuracy: 0.975
Epoch 27/50
33/33 [==============================] - 6s 166ms/step - loss: 0.0616 - accuracy: 0.9776 - val_loss: 0.0094 - val_accuracy: 0.997
Epoch 28/50
33/33 [==============================] - 4s 134ms/step - loss: 0.0619 - accuracy: 0.9781 - val_loss: 0.0427 - val_accuracy: 0.987
Epoch 29/50
33/33 [==============================] - 5s 168ms/step - loss: 0.0543 - accuracy: 0.9790 - val_loss: 0.0235 - val_accuracy: 0.991
Epoch 30/50
33/33 [==============================] - 5s 149ms/step - loss: 0.0486 - accuracy: 0.9824 - val_loss: 0.0152 - val_accuracy: 0.992
Epoch 31/50
33/33 [==============================] - 4s 136ms/step - loss: 0.0390 - accuracy: 0.9881 - val_loss: 0.0197 - val_accuracy: 0.994
Epoch 32/50
33/33 [==============================] - 6s 186ms/step - loss: 0.0513 - accuracy: 0.9829 - val_loss: 0.0138 - val_accuracy: 0.995
Epoch 33/50
33/33 [==============================] - 7s 215ms/step - loss: 0.0430 - accuracy: 0.9848 - val_loss: 0.0142 - val_accuracy: 0.994
Epoch 34/50
33/33 [==============================] - 6s 168ms/step - loss: 0.0445 - accuracy: 0.9833 - val_loss: 0.0120 - val_accuracy: 0.995
Epoch 35/50
33/33 [==============================] - 5s 151ms/step - loss: 0.0458 - accuracy: 0.9805 - val_loss: 0.0135 - val_accuracy: 0.995
Epoch 36/50
33/33 [==============================] - 4s 134ms/step - loss: 0.0452 - accuracy: 0.9848 - val_loss: 0.0124 - val_accuracy: 0.995
Epoch 37/50
33/33 [==============================] - 7s 207ms/step - loss: 0.0458 - accuracy: 0.9848 - val_loss: 0.0115 - val_accuracy: 0.995
Epoch 38/50
33/33 [==============================] - 4s 135ms/step - loss: 0.0503 - accuracy: 0.9790 - val_loss: 0.0119 - val_accuracy: 0.995
Epoch 39/50
33/33 [==============================] - 4s 134ms/step - loss: 0.0412 - accuracy: 0.9857 - val_loss: 0.0122 - val_accuracy: 0.995
Epoch 40/50
33/33 [==============================] - 6s 187ms/step - loss: 0.0419 - accuracy: 0.9843 - val_loss: 0.0127 - val_accuracy: 0.995
Epoch 41/50
33/33 [==============================] - 5s 137ms/step - loss: 0.0357 - accuracy: 0.9905 - val_loss: 0.0128 - val_accuracy: 0.995
Epoch 42/50
33/33 [==============================] - 5s 139ms/step - loss: 0.0403 - accuracy: 0.9843 - val_loss: 0.0132 - val_accuracy: 0.995
Epoch 43/50
33/33 [==============================] - 6s 176ms/step - loss: 0.0459 - accuracy: 0.9814 - val_loss: 0.0130 - val_accuracy: 0.995
Epoch 44/50
33/33 [==============================] - 4s 136ms/step - loss: 0.0444 - accuracy: 0.9829 - val_loss: 0.0130 - val_accuracy: 0.995
Epoch 45/50
33/33 [==============================] - 5s 160ms/step - loss: 0.0560 - accuracy: 0.9771 - val_loss: 0.0132 - val_accuracy: 0.995
Epoch 46/50
33/33 [==============================] - 5s 157ms/step - loss: 0.0372 - accuracy: 0.9890 - val_loss: 0.0132 - val_accuracy: 0.995
Epoch 47/50
33/33 [==============================] - 4s 134ms/step - loss: 0.0459 - accuracy: 0.9838 - val_loss: 0.0132 - val_accuracy: 0.995
Epoch 48/50
33/33 [==============================] - 6s 177ms/step - loss: 0.0375 - accuracy: 0.9867 - val_loss: 0.0132 - val_accuracy: 0.995
Epoch 49/50
33/33 [==============================] - 5s 141ms/step - loss: 0.0456 - accuracy: 0.9829 - val_loss: 0.0132 - val_accuracy: 0.995
Epoch 50/50
33/33 [==============================] - 4s 135ms/step - loss: 0.0443 - accuracy: 0.9871 - val_loss: 0.0133 - val_accuracy: 0.995
```

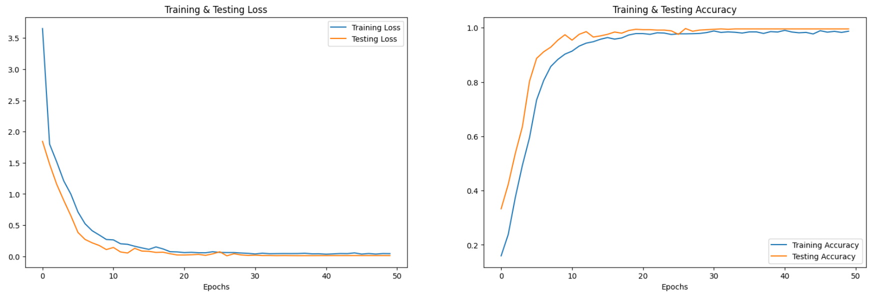
```
print("Accuracy of our model on test data : " , model.evaluate(x_test,y_test)[1]*100 , "%")

epochs = [i for i in range(50)]
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()
```

```
22/22 [==============================] - 0s 14ms/step - loss: 0.0133 - accuracy: 0.99
Accuracy of our model on test data :  99.57143068313599 %
```



We can see our model is more accurate in predicting surprise, angry emotions and it makes sense also because audio files of these emotions differ to other audio files in a lot of ways like pitch, speed etc.. We overall achieved 99% accuracy on our test data and its decent but we can improve it more by applying more augmentation techniques and using other feature extraction methods.

LSTM Model

```
model1 = Sequential([
    LSTM(256, return_sequences=False, input_shape=(40,1)),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(7, activation='softmax')
])

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model1.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 256)               264192

 dropout_14 (Dropout)        (None, 256)               0

 dense_14 (Dense)            (None, 128)               32896
```

```
 dropout_15 (Dropout)        (None, 128)               0

 dense_15 (Dense)            (None, 64)                8256

 dropout_16 (Dropout)        (None, 64)                0

 dense_16 (Dense)            (None, 7)                 455

=================================================================
Total params: 305799 (1.17 MB)
Trainable params: 305799 (1.17 MB)
Non-trainable params: 0 (0.00 Byte)
```