SPEECH EMOTION RECOGNITION

In [ ]:
```
#install kaggle
!pip install -q kaggle
```

In [ ]:
```
#create a kaggle folder
! mkdir ~/.kaggle/
```

In [ ]:
```
!kaggle datasets download -d ejlok1/toronto-emotional-speech-set-tess
```

```
Dataset URL: https://www.kaggle.com/datasets/ejlok1/toronto-emotional-speech-set-tess
License(s): Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0)
Downloading toronto-emotional-speech-set-tess.zip to /content
 96% 409M/428M [00:04<00:00, 130MB/s]
100% 428M/428M [00:04<00:00, 98.0MB/s]
```

In [ ]:
```
!unzip toronto-emotional-speech-set-tess.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_back_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_bar_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_base_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_bath_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_bean_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_beg_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_bite_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_boat_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_bone_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_book_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_bought_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_burn_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_cab_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_calm_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_came_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_cause_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_chain_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_chair_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_chalk_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_chat_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_check_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_cheek_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_chief_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_choice_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_cool_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_dab_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_date_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_dead_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_death_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_deep_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_dime_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_dip_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_ditch_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_dodge_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_dog_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_doll_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_door_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_fail_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_fall_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_far_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_fat_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_fit_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_five_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_food_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_gap_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_gas_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_gaze_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_germ_angry.wav
  inflating: TESS Toronto emotional speech set data/OAF_angry/OAF_get_angry.wav
```

Import Modules

In [ ]:
```
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 keras import utils
```

Load the Dataset

In [ ]:
```
paths =[]
labels = []

for dirname, _, filenames in os.walk('/content/tess toronto emotional speech set data'):
  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
```

In [ ]:
```
len(paths)
```

Out[ ]:
```
2800
```

In [ ]:
```
paths[:5]
```

Out[ ]:
```
['/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_rat_ps.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_puff_ps.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_five_ps.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_deep_ps.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_gin_ps.wav']
```

In [ ]:
```
labels[:5]
```

Out[ ]:
```
['ps', 'ps', 'ps', 'ps', 'ps']
```

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

Out[ ]:

	speech	label
0	/content/tess toronto emotional speech set dat...	ps
1	/content/tess toronto emotional speech set dat...	ps
2	/content/tess toronto emotional speech set dat...	ps
3	/content/tess toronto emotional speech set dat...	ps
4	/content/tess toronto emotional speech set dat...	ps

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

Out[ ]:
```
label
ps          400
neutral     400
fear        400
happy       400
disgust     400
angry       400
sad         400
Name: count, dtype: int64
```

In [ ]:
```
df['label_count']= df['label'].value_counts()
```

In [ ]:
```
df.drop('label_count', axis = 1)
```

Out[ ]:

	speech	label
0	/content/tess toronto emotional speech set dat...	ps
1	/content/tess toronto emotional speech set dat...	ps
2	/content/tess toronto emotional speech set dat...	ps
3	/content/tess toronto emotional speech set dat...	ps
4	/content/tess toronto emotional speech set dat...	ps
...	...	...
2795	/content/tess toronto emotional speech set dat...	ps
2796	/content/tess toronto emotional speech set dat...	ps
2797	/content/tess toronto emotional speech set dat...	ps
2798	/content/tess toronto emotional speech set dat...	ps
2799	/content/tess toronto emotional speech set dat...	ps

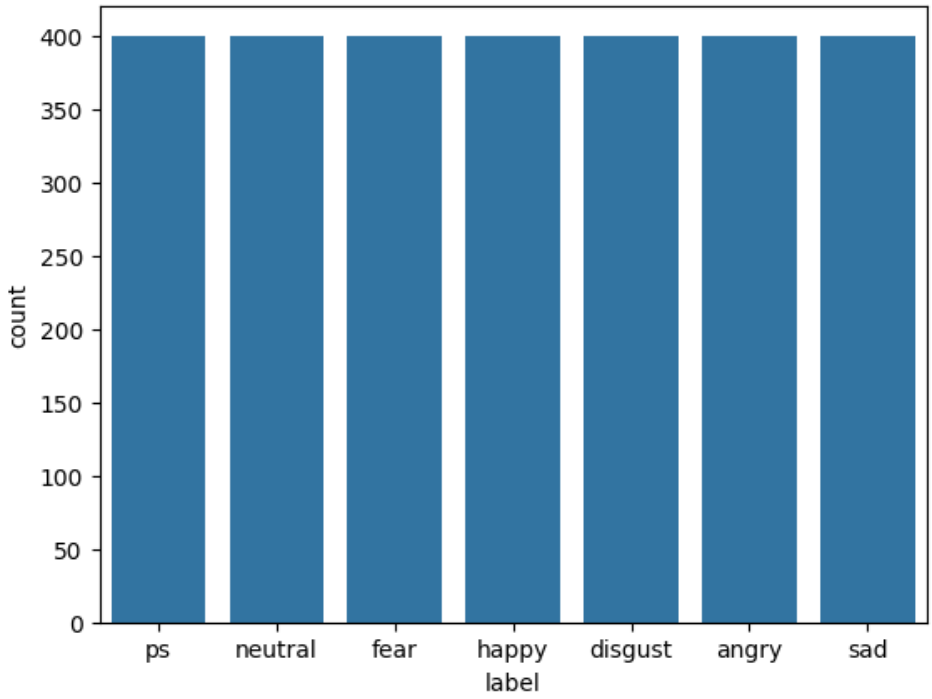
2800 rows × 2 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2800 entries, 0 to 2799
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   speech       2800 non-null   object 
 1   label        2800 non-null   object 
 2   label_count  0 non-null      float64
dtypes: float64(1), object(2)
memory usage: 65.8+ KB
```

In [ ]:
```
sns.countplot(data=df, x='label')
```

Out[ ]:
```
<Axes: xlabel='label', ylabel='count'>
```



In [ ]:
```
df
```

Out[ ]:

	speech	label	label_count
0	/content/tess toronto emotional speech set dat...	ps	NaN
1	/content/tess toronto emotional speech set dat...	ps	NaN
2	/content/tess toronto emotional speech set dat...	ps	NaN
3	/content/tess toronto emotional speech set dat...	ps	NaN
4	/content/tess toronto emotional speech set dat...	ps	NaN
...	...	...	...
2795	/content/tess toronto emotional speech set dat...	ps	NaN
2796	/content/tess toronto emotional speech set dat...	ps	NaN
2797	/content/tess toronto emotional speech set dat...	ps	NaN
2798	/content/tess toronto emotional speech set dat...	ps	NaN
2799	/content/tess toronto emotional speech set dat...	ps	NaN

2800 rows × 3 columns

In [ ]:
```
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()
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2800 entries, 0 to 2799
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   speech       2800 non-null   object 
 1   label        2800 non-null   object 
 2   label_count  0 non-null      float64
dtypes: float64(1), object(2)
memory usage: 65.8+ KB
```

In [ ]:
```
df.drop('label_count', axis=1, inplace = True)
```

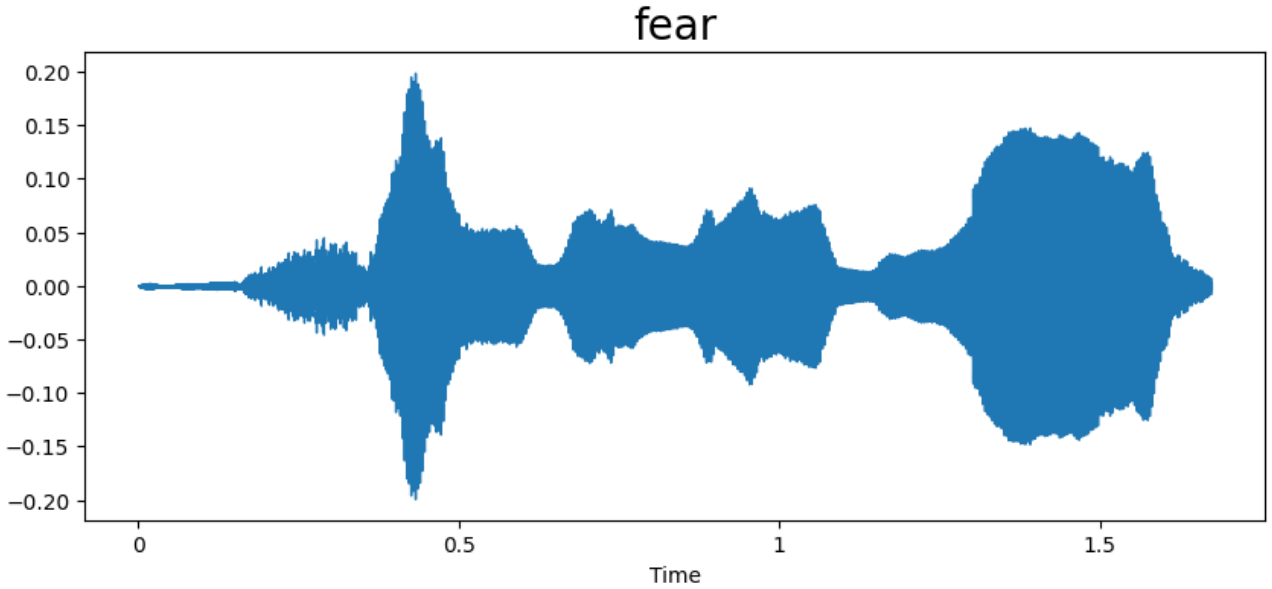
In [ ]:
```
print(df.head())
print(df['label'].unique())
```

```
                                              speech label
0  /content/tess toronto emotional speech set dat...    ps
1  /content/tess toronto emotional speech set dat...    ps
2  /content/tess toronto emotional speech set dat...    ps
3  /content/tess toronto emotional speech set dat...    ps
4  /content/tess toronto emotional speech set dat...    ps
['ps' 'neutral' 'fear' 'happy' 'disgust' 'angry' 'sad']
```

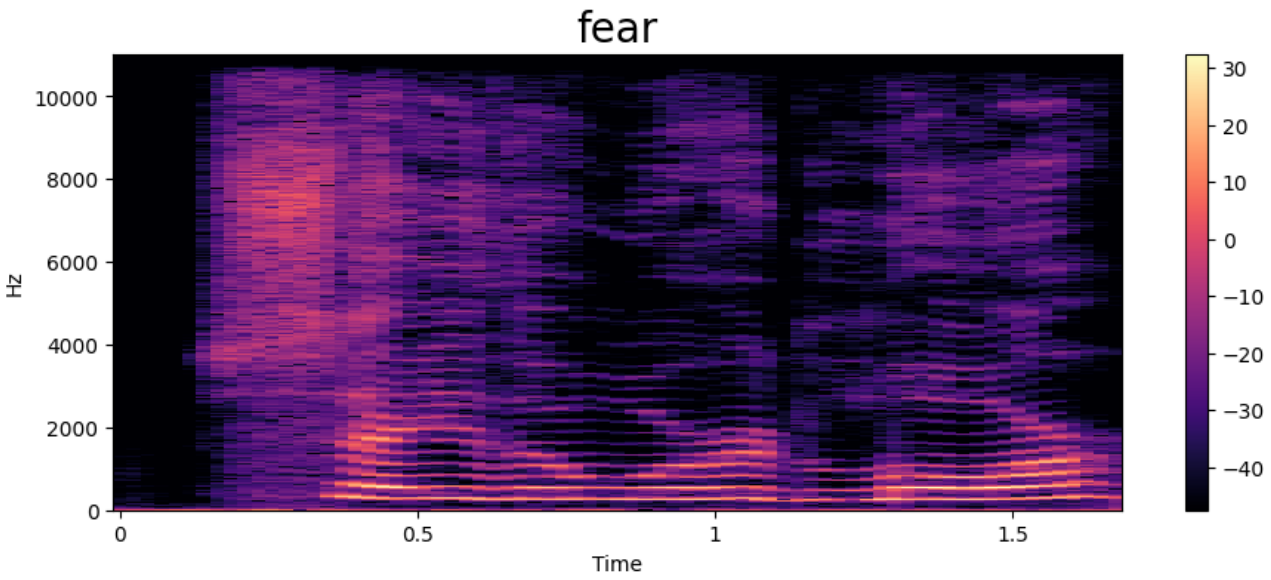
In [ ]:
```
df['speech'].unique()
```

Out[ ]:
```
array(['/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_rat_ps.wav',
       '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_puff_ps.wav',
       '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/OAF_Pleasant_surprise/OAF_five_ps.wav',
       ...,
       '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_pleasant_surprised/YAF_pick_ps.wav',
       '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_pleasant_surprised/YAF_learn_ps.wav',
       '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_pleasant_surprised/YAF_rain_ps.wav'],
      dtype=object)
```

In [ ]:
```
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)
```

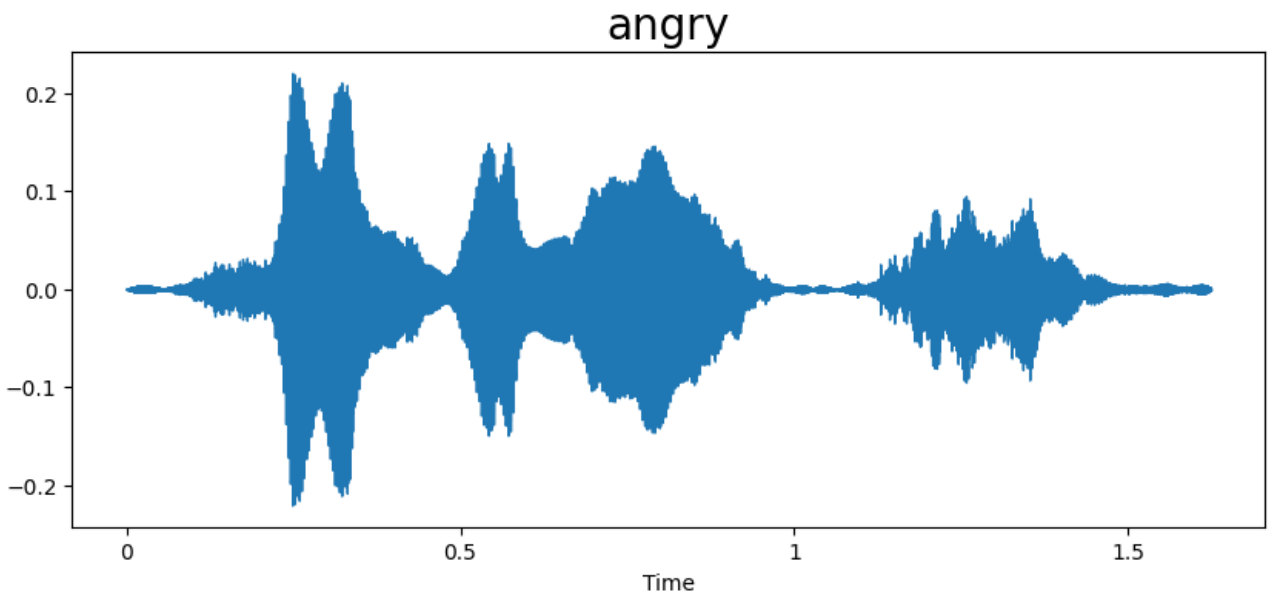


Out[ ]: 0:00 -0:01

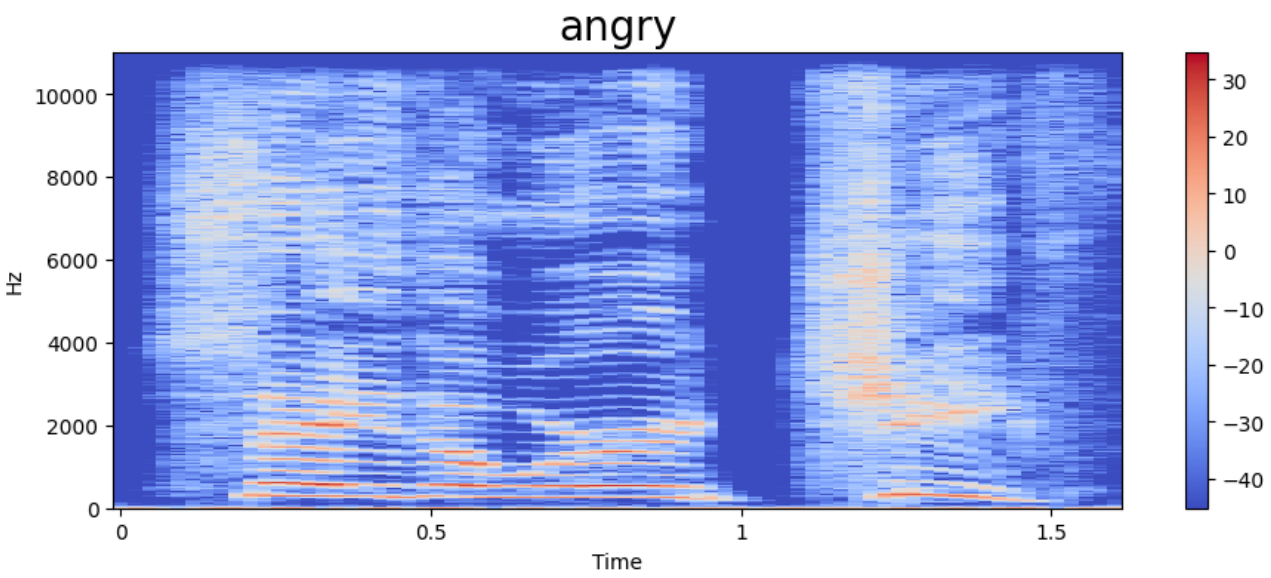


In [ ]:
```
emotion = 'angry'
```

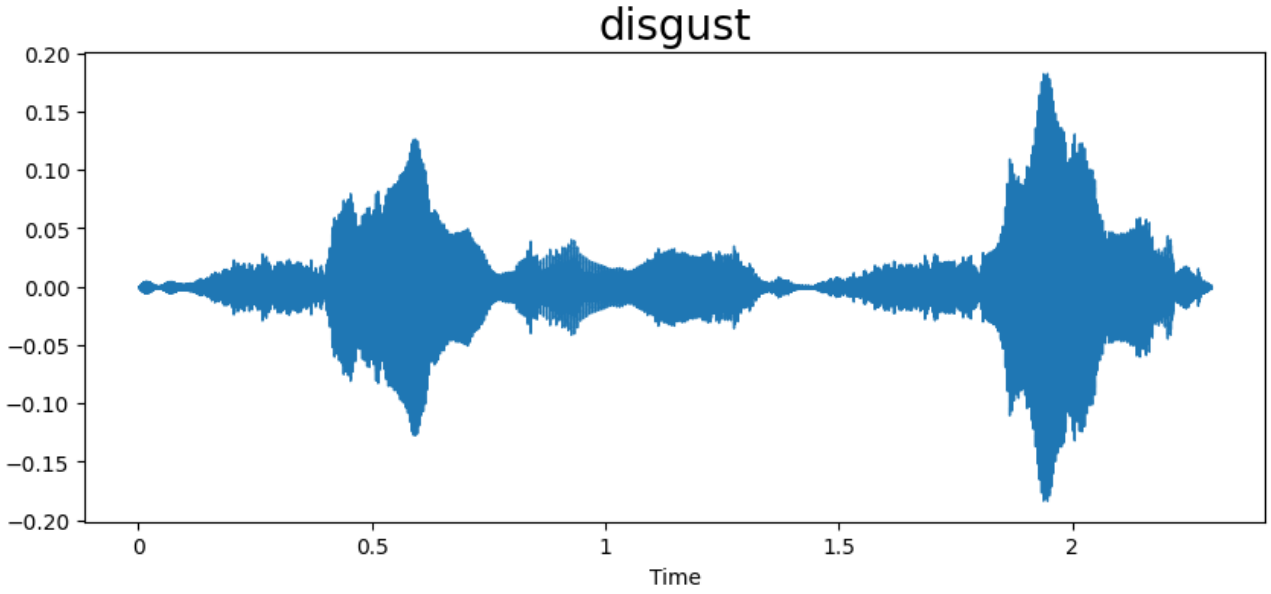
In [ ]:
```
path = np.array(df['speech'][df['label']==emotion])[1]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



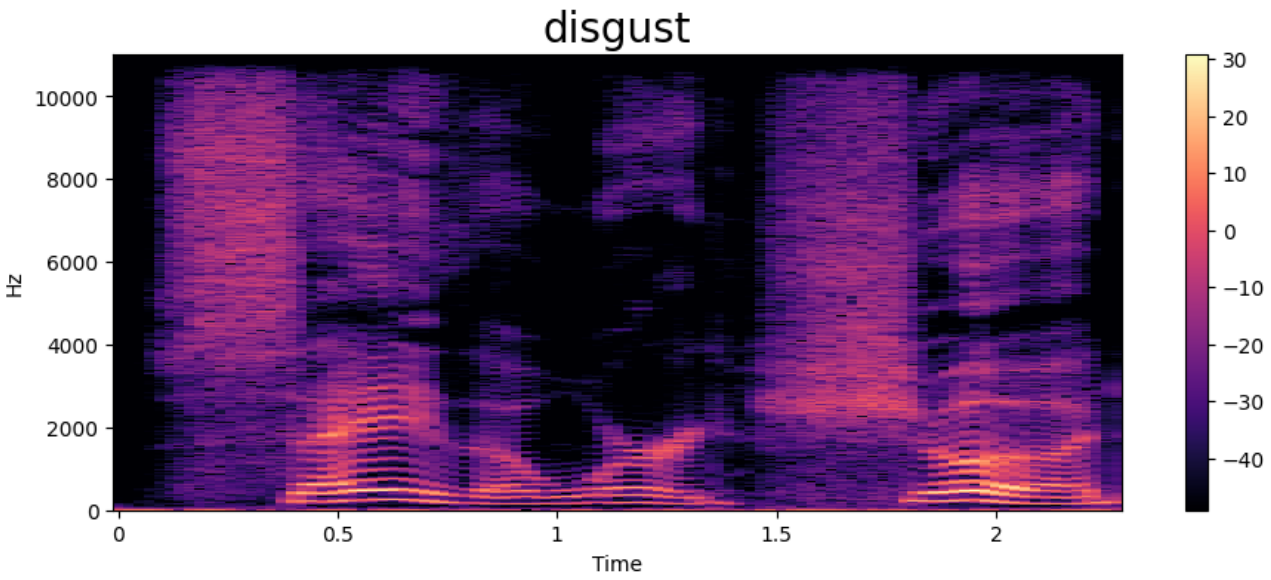
Out[ ]: 0:00 -0:01



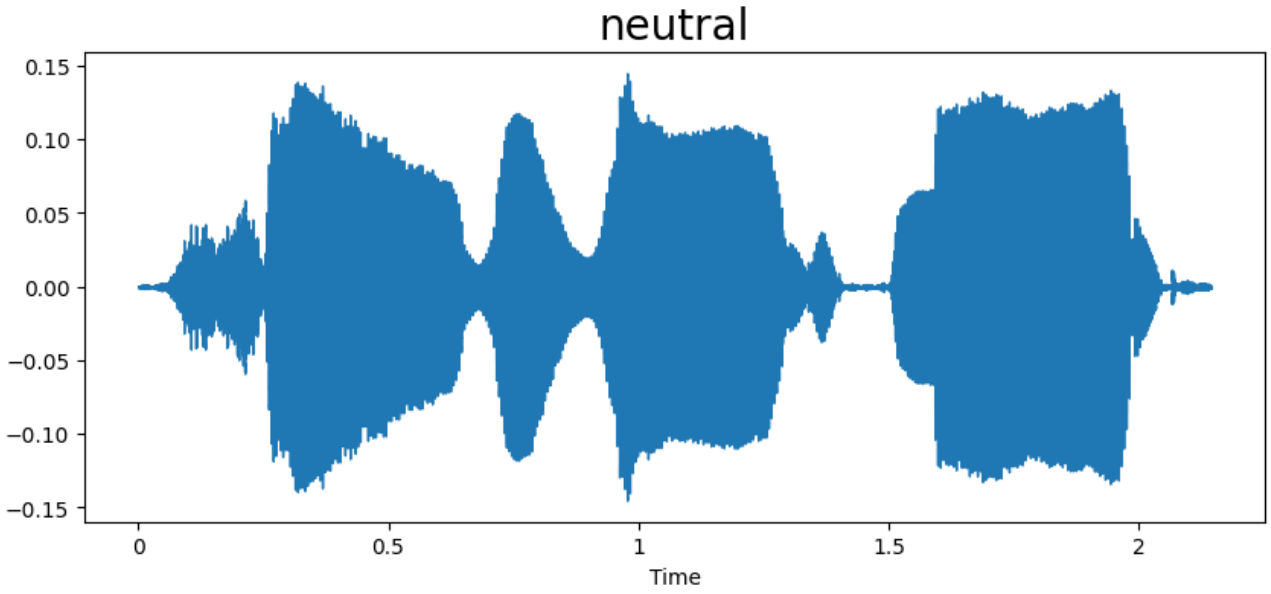
In [ ]:
```
emotion = 'disgust'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



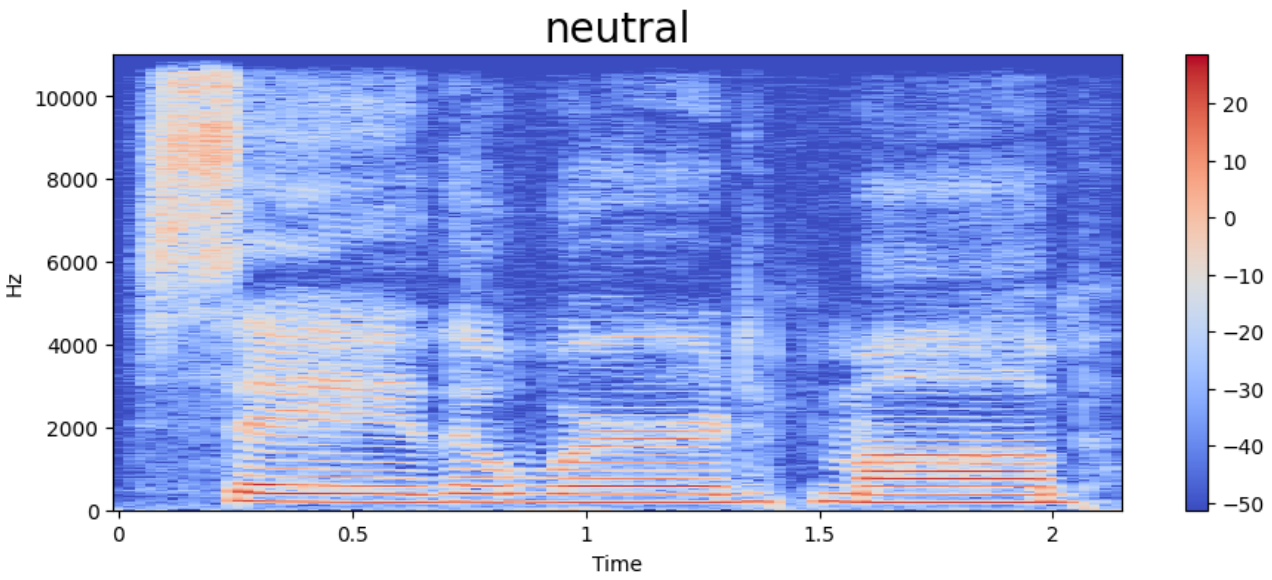
Out[ ]: 0:00 -0:02



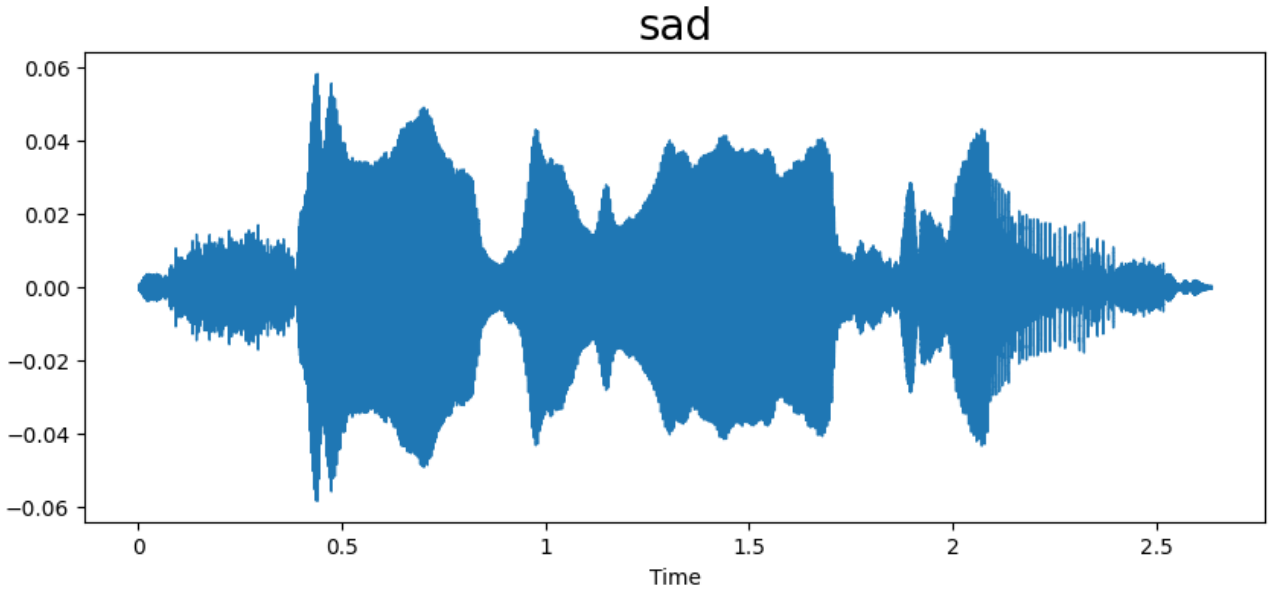
In [ ]:
```
emotion = 'neutral'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



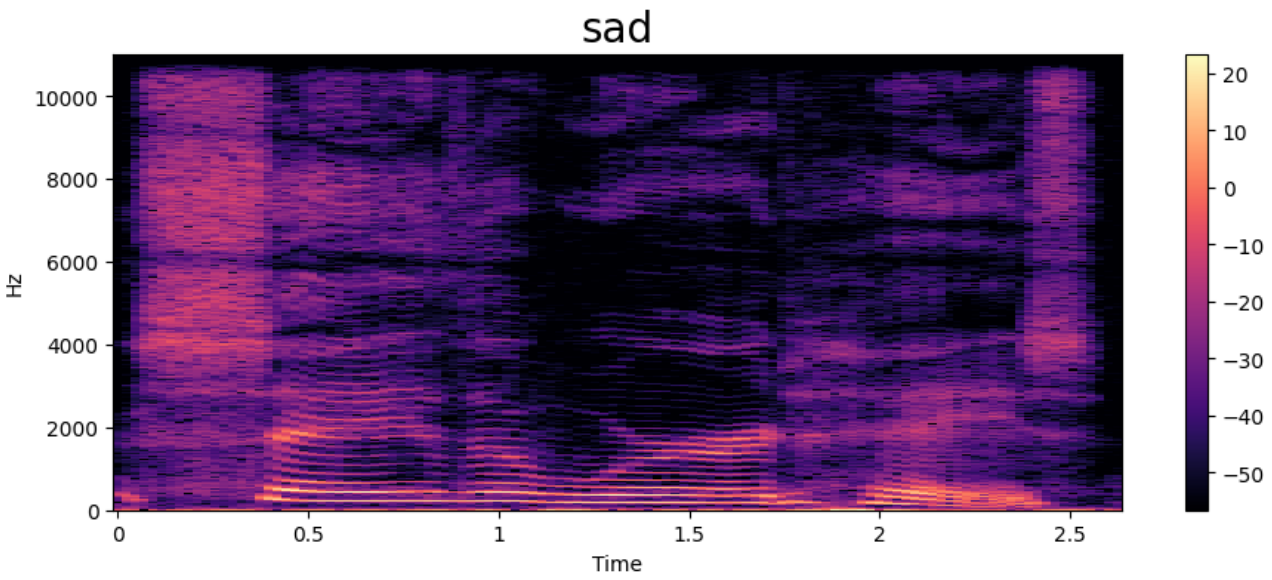
Out[ ]: 0:00 -0:02



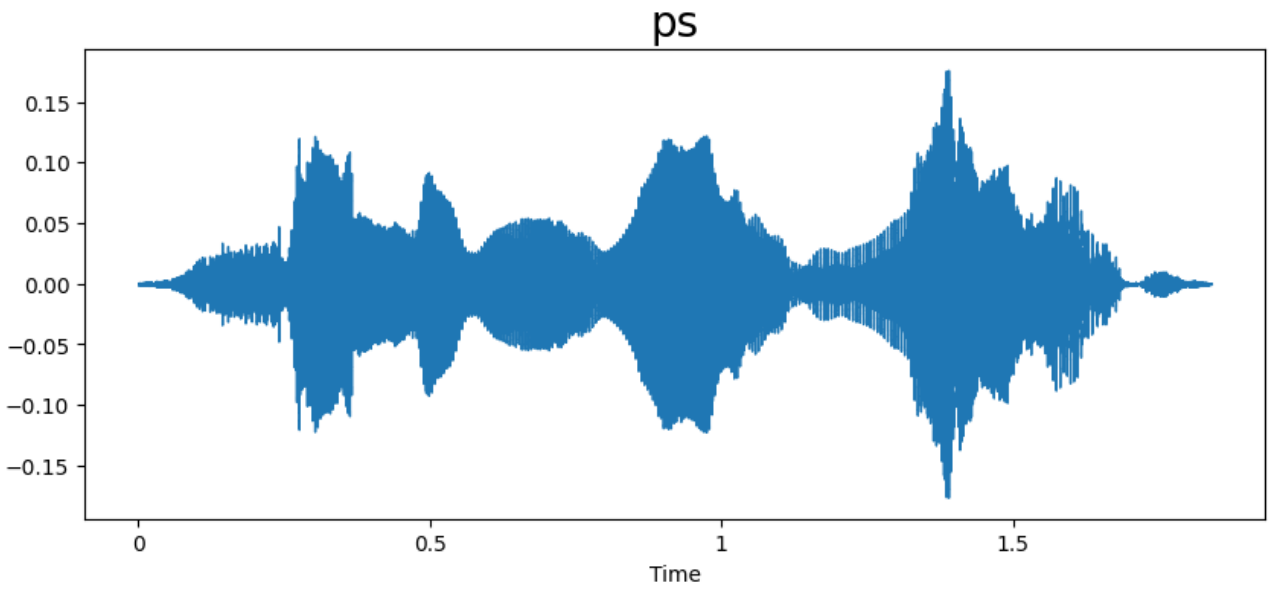
In [ ]:
```
emotion = 'sad'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



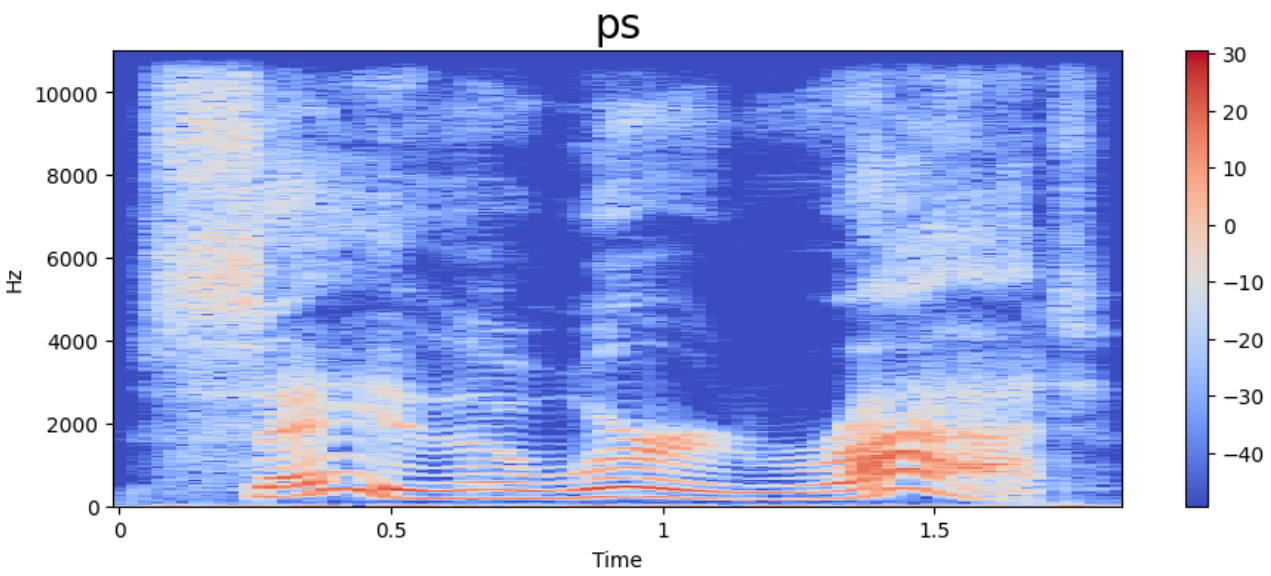
Out[ ]: 0:00 -0:02



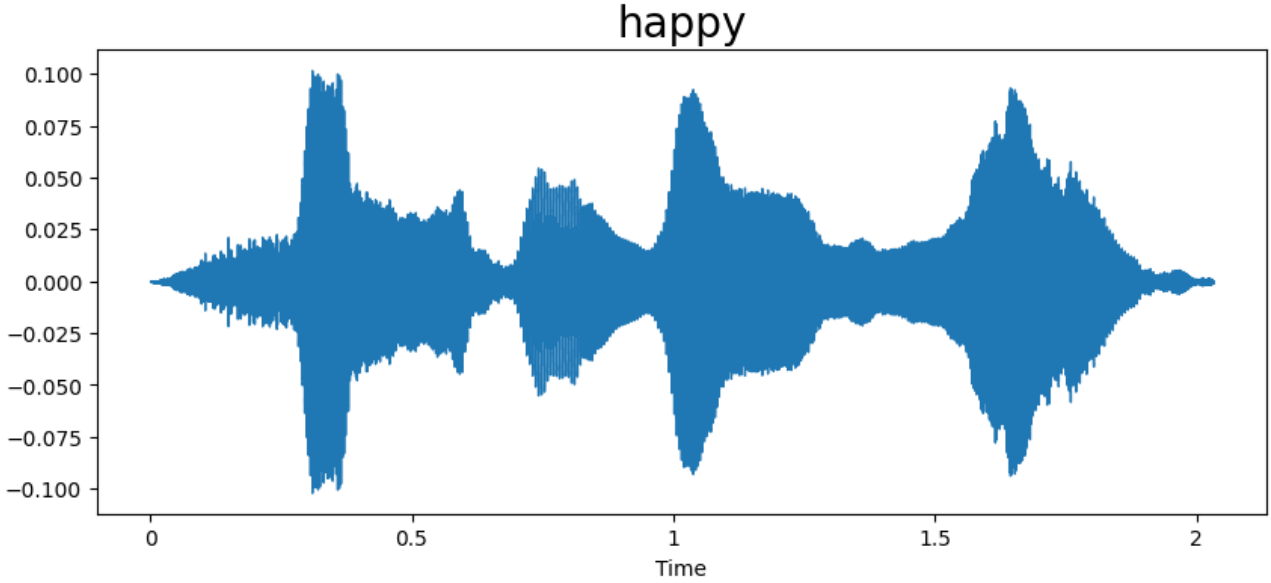
In [ ]:
```
emotion = 'ps'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



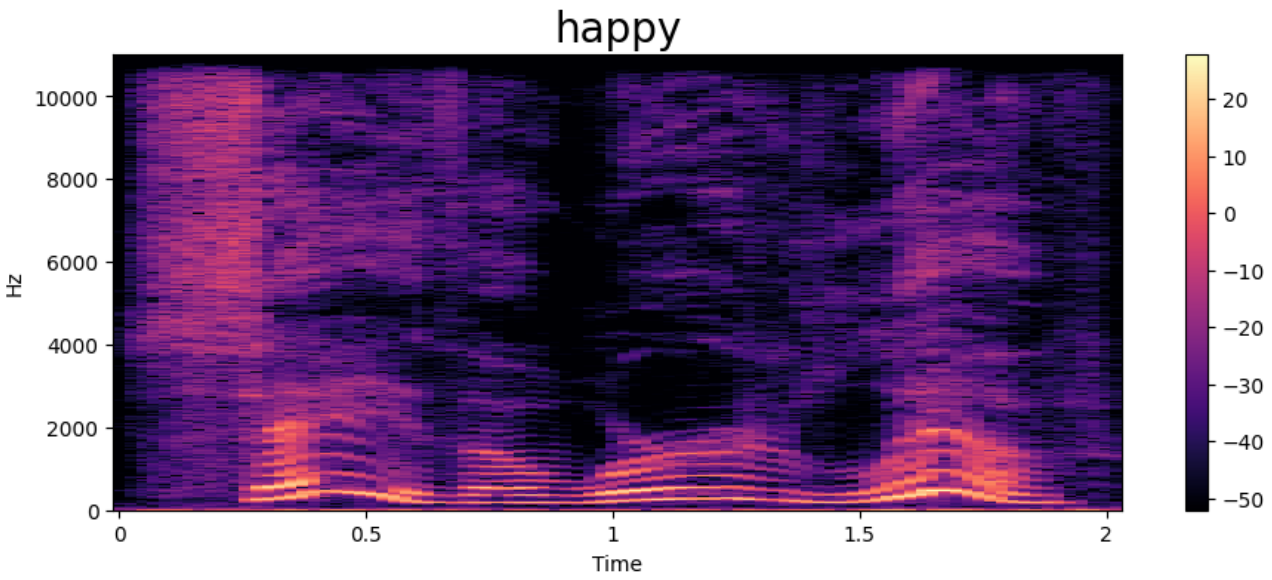
Out[ ]: 0:00 -0:01



In [ ]:
```
emotion = 'happy'
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)
```



Out[ ]: 0:00 -0:02



Feature Extraction:

In [ ]:
```
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
```

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

Out[ ]:
```
array([-3.9494641e+02,  1.1897662e+02, -7.5107522e+00, -3.8542004e+01,
        3.5122010e-01,  5.6513729e+00, -8.8136845e+00, -9.0428944e+00,
       -1.3719249e+01, -8.5861044e+00, -1.1364076e+01, -8.7831764e+00,
       -1.1838287e+01,  2.9407659e+00,  8.0493408e-01,  1.0273455e+00,
       -6.2382430e-01,  9.8965883e+00,  1.7884728e+00, -2.8778318e-01,
       -1.3282847e+00,  2.1907587e+00, -5.1745949e+00,  3.0393071e+00,
       -8.3139286e+00,  2.9301190e+00, -7.9847221e+00,  1.2253859e+00,
       -2.9079890e+00,  4.7295918e+00,  4.2148131e-01,  6.7323003e+00,
        2.9153802e+00,  6.9420042e+00,  7.7420430e+00,  1.1941119e+01,
        1.7751865e+01,  1.6171938e+01,  1.1081786e+01,  9.2769852e+00],
      dtype=float32)
```

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

In [ ]:
```
X_mfcc
```

Out[ ]:
```
0       [-394.9464, 118.97662, -7.510752, -38.542004, ...
1       [-445.4041, 93.353004, 2.4792054, -16.53722, 0...
2       [-394.86282, 103.175896, 1.1922144, -18.462278...
3       [-475.11322, 72.03712, 16.919844, 18.233713, 2...
4       [-426.6076, 80.58388, 12.451724, 13.206113, 12...
                              ...
2795    [-330.76318, 59.57205, -8.297699, -6.495669, -...
2796    [-376.9508, 53.77245, -4.322307, 16.0703, -10....
2797    [-354.5709, 64.17273, -31.288214, 14.938522, -...
2798    [-350.19223, 102.610146, -27.542908, -15.65984...
2799    [-365.01642, 77.617744, -11.532593, 12.601002,...
Name: speech, Length: 2800, dtype: object
```

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

Out[ ]:
```
(2800, 40)
```

In [ ]:
```
## input split
X = np.expand_dims(X, -1)
X.shape
```

Out[ ]:
```
(2800, 40, 1)
```

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

In [ ]:
```
y = y.toarray()
```

In [ ]:
```
y.shape
```

Out[ ]:
```
(2800, 7)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Create the LSTM Model

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential([
    LSTM(256, return_sequences=False, input_shape=(40, 1)),
    Dropout(0.5),  # Add dropout after LSTM layer
    Dense(128, activation='relu'),
    Dropout(0.5),  # Add dropout after dense layer
    Dense(64, activation='relu'),
    Dropout(0.5),  # Add dropout after dense layer
    Dense(7, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
Model: "sequential"
```

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

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 7)                 455

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

In [ ]:
```
#Train the model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=30, batch_size=64)
```

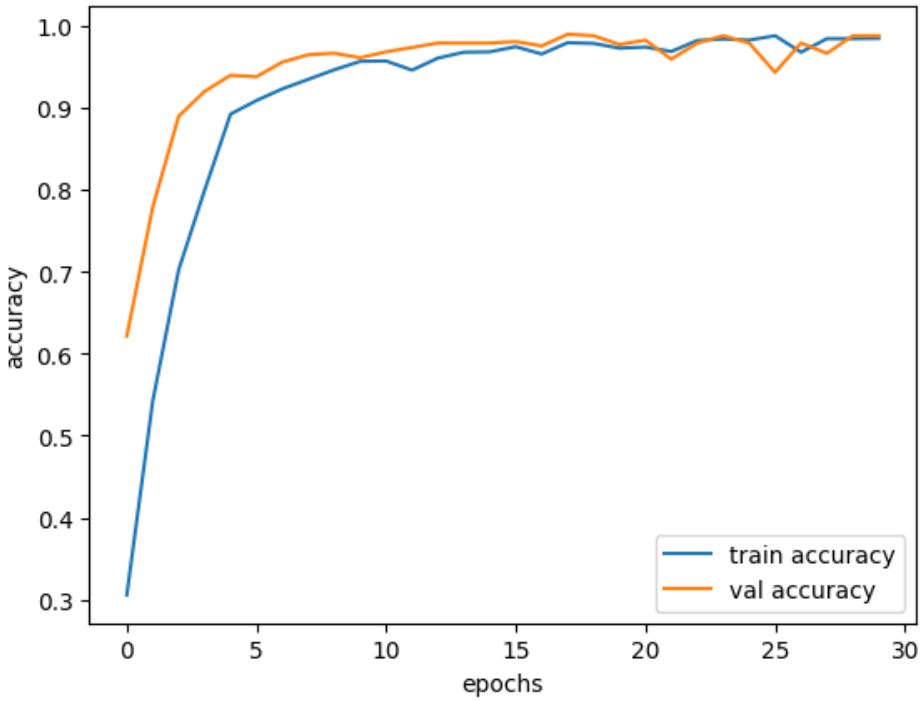
```
Epoch 1/30
35/35 [==============================] - 10s 185ms/step - loss: 1.7101 - accuracy: 0.3058 - val_lo
ss: 1.0430 - val_accuracy: 0.6214
Epoch 2/30
35/35 [==============================] - 7s 198ms/step - loss: 1.1247 - accuracy: 0.5420 - val_los
s: 0.6233 - val_accuracy: 0.7786
Epoch 3/30
35/35 [==============================] - 6s 158ms/step - loss: 0.7347 - accuracy: 0.7022 - val_los
s: 0.3540 - val_accuracy: 0.8893
Epoch 4/30
35/35 [==============================] - 6s 183ms/step - loss: 0.5757 - accuracy: 0.7991 - val_los
s: 0.2282 - val_accuracy: 0.9196
Epoch 5/30
35/35 [==============================] - 6s 183ms/step - loss: 0.3721 - accuracy: 0.8920 - val_los
```

```
s: 0.1750 - val_accuracy: 0.9393
Epoch 6/30
35/35 [==============================] - 6s 161ms/step - loss: 0.2994 - accuracy: 0.9085 - val_los
s: 0.1736 - val_accuracy: 0.9375
Epoch 7/30
35/35 [==============================] - 7s 207ms/step - loss: 0.2839 - accuracy: 0.9228 - val_los
s: 0.1516 - val_accuracy: 0.9554
Epoch 8/30
35/35 [==============================] - 6s 159ms/step - loss: 0.2310 - accuracy: 0.9344 - val_los
s: 0.1181 - val_accuracy: 0.9643
Epoch 9/30
35/35 [==============================] - 7s 210ms/step - loss: 0.2009 - accuracy: 0.9460 - val_los
s: 0.1075 - val_accuracy: 0.9661
Epoch 10/30
35/35 [==============================] - 6s 160ms/step - loss: 0.1625 - accuracy: 0.9563 - val_los
s: 0.1221 - val_accuracy: 0.9607
Epoch 11/30
35/35 [==============================] - 8s 217ms/step - loss: 0.1690 - accuracy: 0.9567 - val_los
s: 0.1191 - val_accuracy: 0.9679
Epoch 12/30
35/35 [==============================] - 6s 164ms/step - loss: 0.1855 - accuracy: 0.9455 - val_los
s: 0.0712 - val_accuracy: 0.9732
Epoch 13/30
35/35 [==============================] - 7s 204ms/step - loss: 0.1566 - accuracy: 0.9603 - val_los
s: 0.0863 - val_accuracy: 0.9786
Epoch 14/30
35/35 [==============================] - 6s 161ms/step - loss: 0.1124 - accuracy: 0.9674 - val_los
s: 0.0816 - val_accuracy: 0.9786
Epoch 15/30
35/35 [==============================] - 6s 176ms/step - loss: 0.1322 - accuracy: 0.9679 - val_los
s: 0.0513 - val_accuracy: 0.9786
Epoch 16/30
35/35 [==============================] - 6s 179ms/step - loss: 0.0949 - accuracy: 0.9741 - val_los
s: 0.0694 - val_accuracy: 0.9804
Epoch 17/30
35/35 [==============================] - 5s 156ms/step - loss: 0.1213 - accuracy: 0.9652 - val_los
s: 0.0770 - val_accuracy: 0.9750
Epoch 18/30
35/35 [==============================] - 7s 209ms/step - loss: 0.0748 - accuracy: 0.9790 - val_los
s: 0.0495 - val_accuracy: 0.9893
Epoch 19/30
35/35 [==============================] - 5s 156ms/step - loss: 0.0860 - accuracy: 0.9781 - val_los
s: 0.0425 - val_accuracy: 0.9875
Epoch 20/30
35/35 [==============================] - 7s 209ms/step - loss: 0.0938 - accuracy: 0.9723 - val_los
s: 0.0957 - val_accuracy: 0.9768
Epoch 21/30
35/35 [==============================] - 5s 156ms/step - loss: 0.1171 - accuracy: 0.9737 - val_los
s: 0.0746 - val_accuracy: 0.9821
Epoch 22/30
35/35 [==============================] - 7s 211ms/step - loss: 0.1149 - accuracy: 0.9683 - val_los
s: 0.1531 - val_accuracy: 0.9589
Epoch 23/30
35/35 [==============================] - 6s 164ms/step - loss: 0.0758 - accuracy: 0.9817 - val_los
s: 0.0783 - val_accuracy: 0.9786
Epoch 24/30
35/35 [==============================] - 6s 183ms/step - loss: 0.0639 - accuracy: 0.9835 - val_los
s: 0.0595 - val_accuracy: 0.9875
Epoch 25/30
35/35 [==============================] - 6s 180ms/step - loss: 0.0627 - accuracy: 0.9821 - val_los
s: 0.0430 - val_accuracy: 0.9786
Epoch 26/30
35/35 [==============================] - 6s 165ms/step - loss: 0.0591 - accuracy: 0.9875 - val_los
s: 0.3741 - val_accuracy: 0.9429
Epoch 27/30
35/35 [==============================] - 7s 198ms/step - loss: 0.1274 - accuracy: 0.9674 - val_los
s: 0.0619 - val_accuracy: 0.9786
Epoch 28/30
35/35 [==============================] - 6s 162ms/step - loss: 0.0688 - accuracy: 0.9839 - val_los
s: 0.1158 - val_accuracy: 0.9661
Epoch 29/30
35/35 [==============================] - 7s 210ms/step - loss: 0.0561 - accuracy: 0.9839 - val_los
s: 0.0263 - val_accuracy: 0.9875
Epoch 30/30
35/35 [==============================] - 5s 157ms/step - loss: 0.0488 - accuracy: 0.9844 - val_los
s: 0.0535 - val_accuracy: 0.9875
```

Plot the results

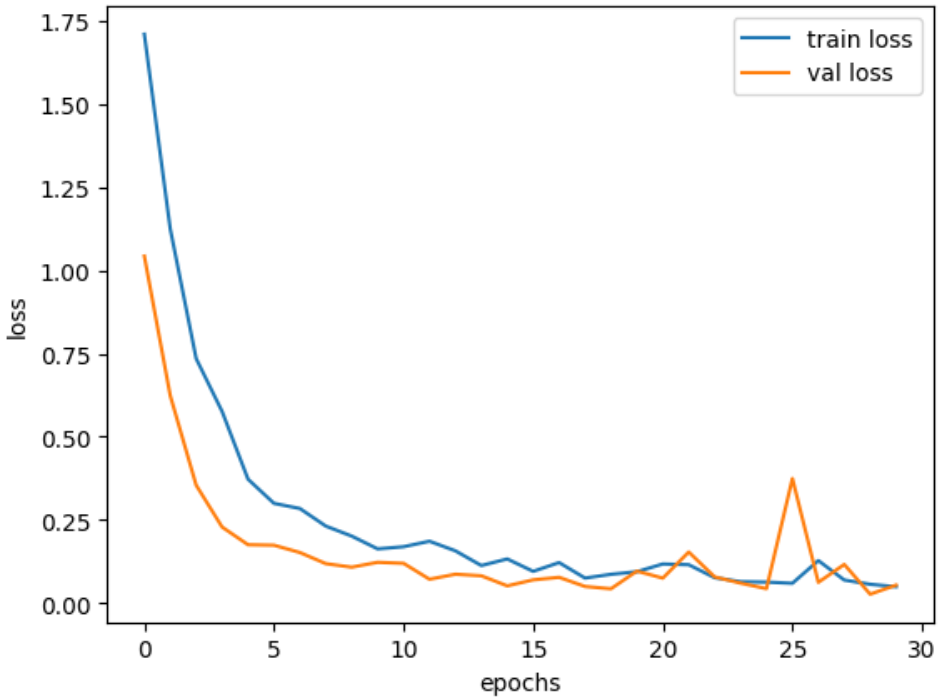
In [ ]:
```
epochs = list(range(30))
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, label='train accuracy')
plt.plot(epochs, val_acc, label='val accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



In [ ]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(epochs, loss, label='train loss')
plt.plot(epochs, val_loss, label='val loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



In [ ]:
```
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
y_val_classes = np.argmax(y_val, axis=1)
```

```
18/18 [==============================] - 1s 43ms/step
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [ ]:
```
# Compute confusion matrix
conf_matrix = confusion_matrix(y_val_classes, y_pred_classes)

# Print the confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[88  0  0  0  0  1  0]
 [ 0 83  0  0  0  1  2]
 [ 0  0 89  0  0  0  0]
 [ 0  0  0 65  0  2  0]
 [ 0  0  0  0 79  0  0]
 [ 0  1  0  0  0 69  0]
 [ 0  0  0  0  0  0 80]]
```

In [ ]:
```
#Print the classification report
target_names = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'ps', 'sad']
print("Classification Report:")
print(classification_report(y_val_classes, y_pred_classes, target_names=target_names))
```

```
Classification Report:
              precision    recall  f1-score   support

       angry       1.00      0.99      0.99        89
     disgust       0.99      0.97      0.98        86
        fear       1.00      1.00      1.00        89
       happy       1.00      0.97      0.98        67
     neutral       1.00      1.00      1.00        79
          ps       0.95      0.99      0.97        70
         sad       0.98      1.00      0.99        80

    accuracy                           0.99       560
   macro avg       0.99      0.99      0.99       560
weighted avg       0.99      0.99      0.99       560
```

In [ ]:
```
#Correlation HeatMap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=target_names, yticklabels=t
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```