speech-emotion-recognition

November 17, 2024

[218]:
```
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
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
import warnings
warnings.filterwarnings('ignore')
```

[220]:
```
paths = []
labels = []
for dirname, _, filenames in os.walk(r"C:\Users\tjyot\Downloads\archive"):
    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
```

[222]:
```
len(paths)
```

[222]: 2800

[224]:
```
paths[:5]
```

[224]:
```
['C:\\Users\\tjyot\\Downloads\\archive\\TESS Toronto emotional speech set
data\\OAF_angry\\OAF_back_angry.wav',
 'C:\\Users\\tjyot\\Downloads\\archive\\TESS Toronto emotional speech set
data\\OAF_angry\\OAF_bar_angry.wav',
```

```
 'C:\\Users\\tjyot\\Downloads\\archive\\TESS Toronto emotional speech set
data\\OAF_angry\\OAF_base_angry.wav',
 'C:\\Users\\tjyot\\Downloads\\archive\\TESS Toronto emotional speech set
data\\OAF_angry\\OAF_bath_angry.wav',
 'C:\\Users\\tjyot\\Downloads\\archive\\TESS Toronto emotional speech set
data\\OAF_angry\\OAF_bean_angry.wav']
```

[226]:
```
labels[:5]
```

[226]:
```
['angry', 'angry', 'angry', 'angry', 'angry']
```

[228]:
```
df = pd.DataFrame()
df['speech'] = paths
df['label'] = labels
df.head()
```

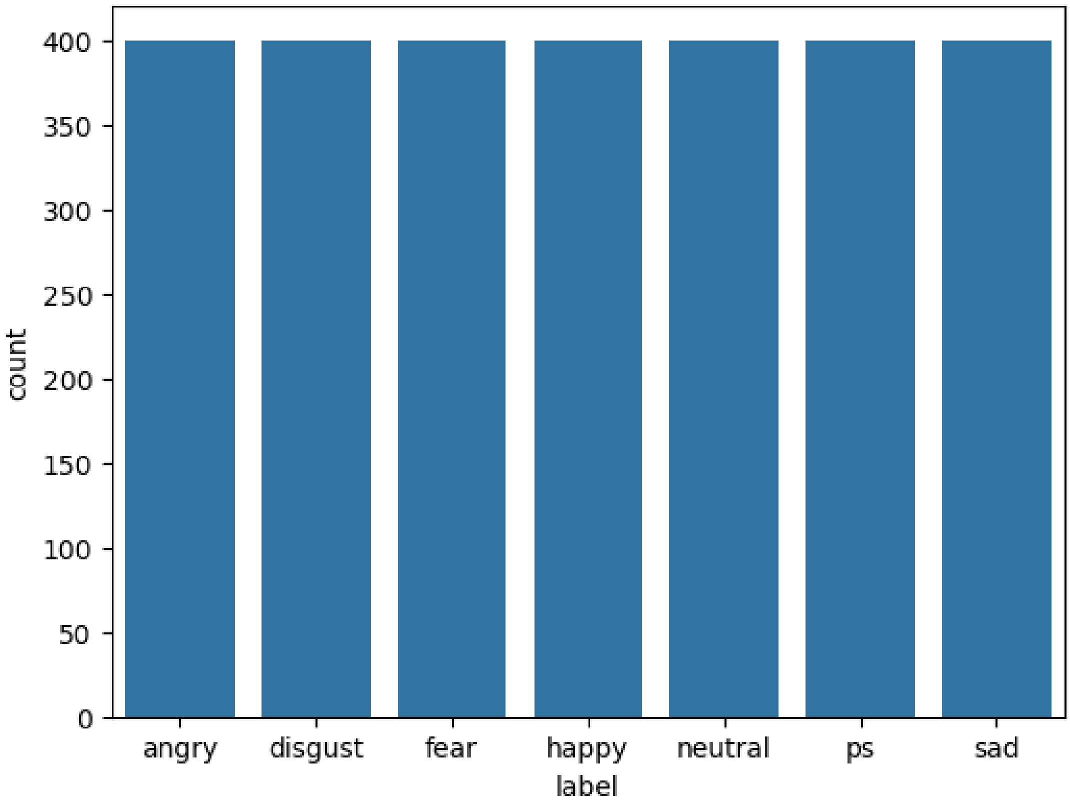
[228]:
```
                                              speech  label
0  C:\Users\tjyot\Downloads\archive\TESS Toronto …  angry
1  C:\Users\tjyot\Downloads\archive\TESS Toronto …  angry
2  C:\Users\tjyot\Downloads\archive\TESS Toronto …  angry
3  C:\Users\tjyot\Downloads\archive\TESS Toronto …  angry
4  C:\Users\tjyot\Downloads\archive\TESS Toronto …  angry
```

[230]:
```
df['label'].value_counts()
```

[230]:
```
label
angry      400
disgust    400
fear       400
happy      400
neutral    400
ps         400
sad        400
Name: count, dtype: int64
```

[232]:
```
sns.countplot(data=df, x='label')
```

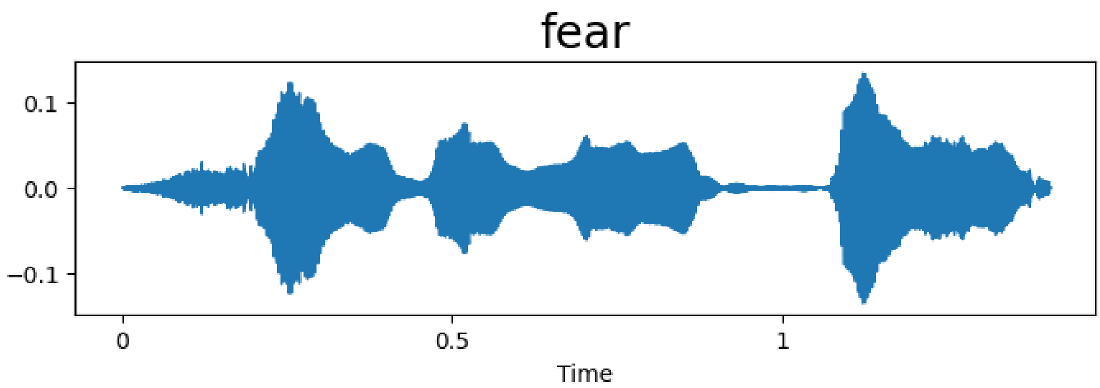
[232]:
```
<Axes: xlabel='label', ylabel='count'>
```



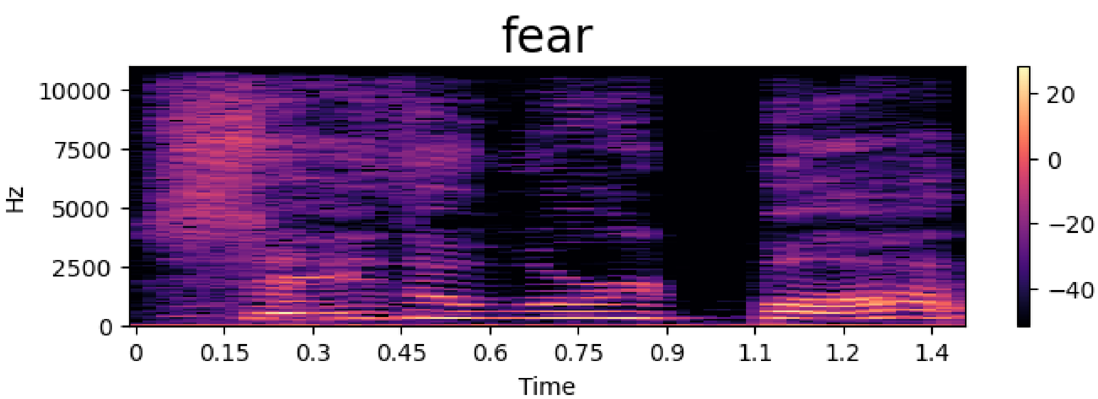
[234]:
```
def waveplot(data, sr, emotion):
    plt.figure(figsize=(8,2))
    plt.title(emotion, size=20)
    librosa.display.waveshow(data, sr=sr)  # Displaying waveplot using librosa
    plt.show()
```

[236]:
```
def spectogram(data, sr, emotion):
    x = librosa.stft(data)  # Applying short-time Fourier transform
    xdb = librosa.amplitude_to_db(abs(x))  # Converting amplitude to decibels
    plt.figure(figsize=(8,2))
    plt.title(emotion, size=20)
    librosa.display.specshow(xdb, sr=sr, x_axis='time', y_axis='hz')  #␣
 ↪Displaying spectrogram
    plt.colorbar()
```

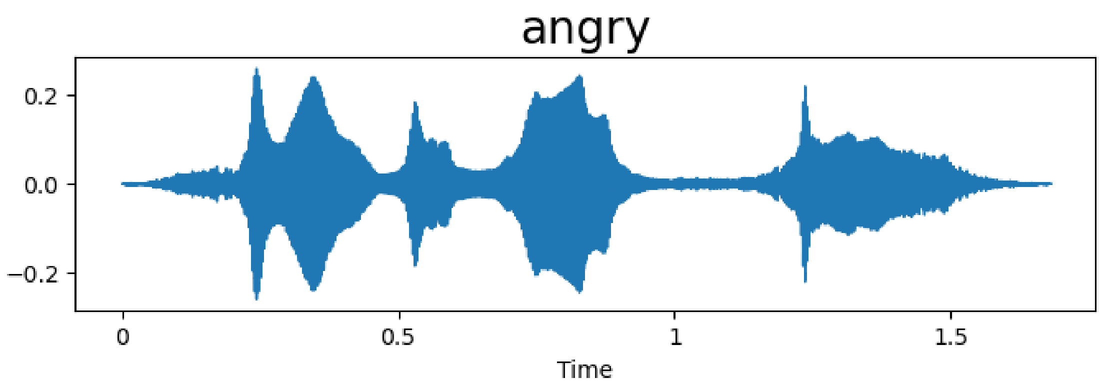
[238]:
```
emotion = 'fear'
path = np.array(df['speech'][df['label']==emotion])[210]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



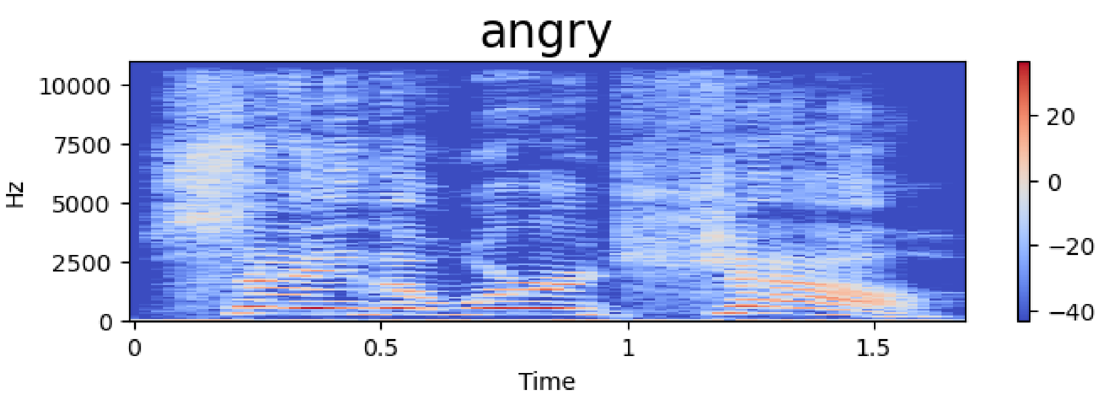
[238]: <IPython.lib.display.Audio object>



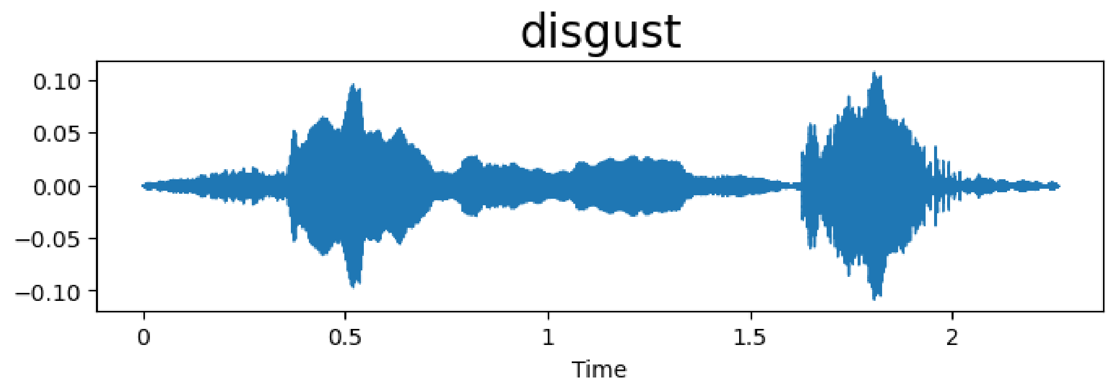
[240]:
```
emotion = 'angry'
path = np.array(df['speech'][df['label']==emotion])[150]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



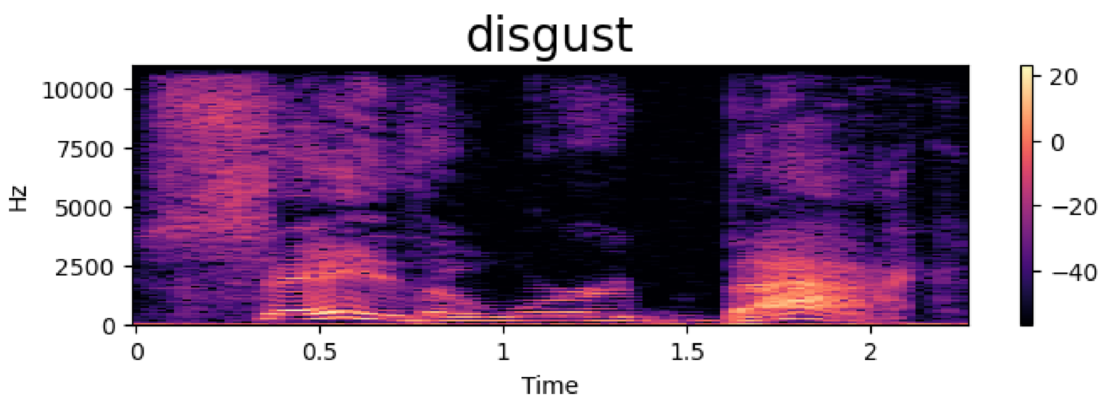
[240]: <IPython.lib.display.Audio object>



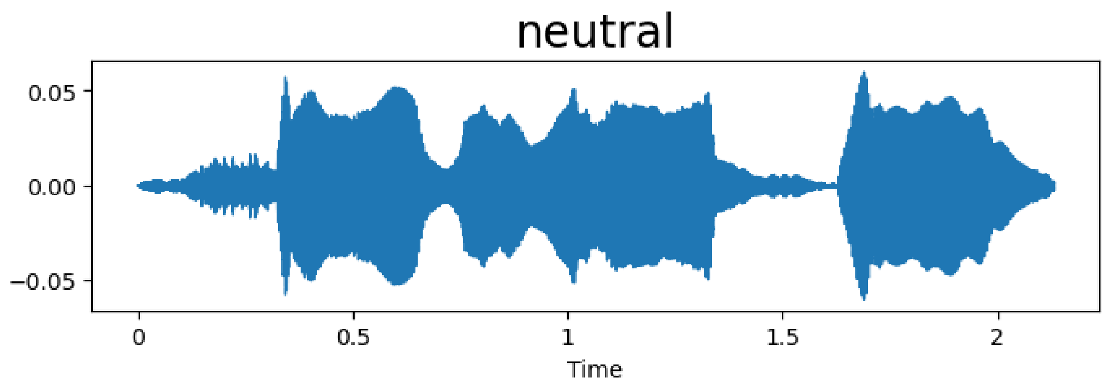
[242]:
```
emotion = 'disgust'
path = np.array(df['speech'][df['label']==emotion])[200]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



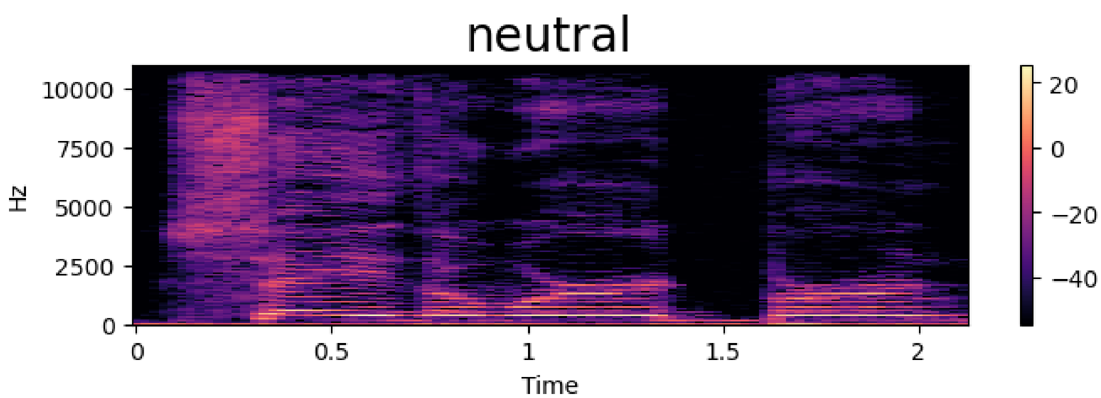
[242]: <IPython.lib.display.Audio object>



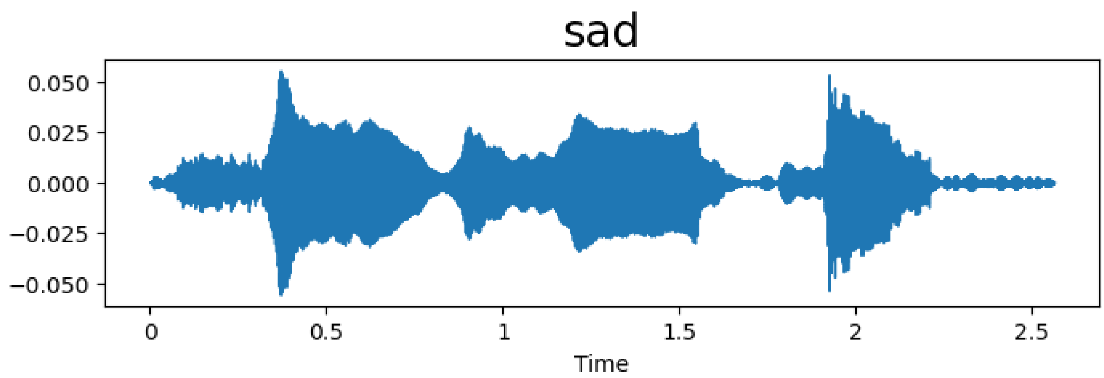
[244]:
```
emotion = 'neutral'
path = np.array(df['speech'][df['label']==emotion])[210]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



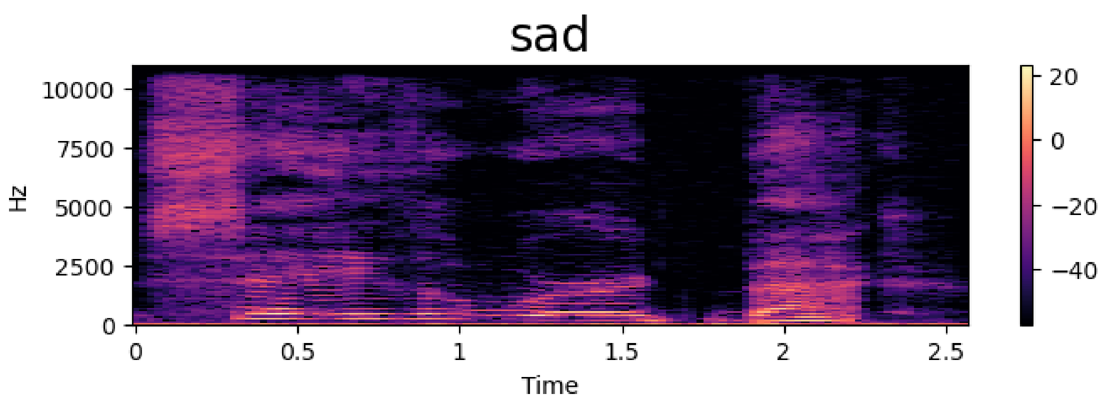
[244]: <IPython.lib.display.Audio object>



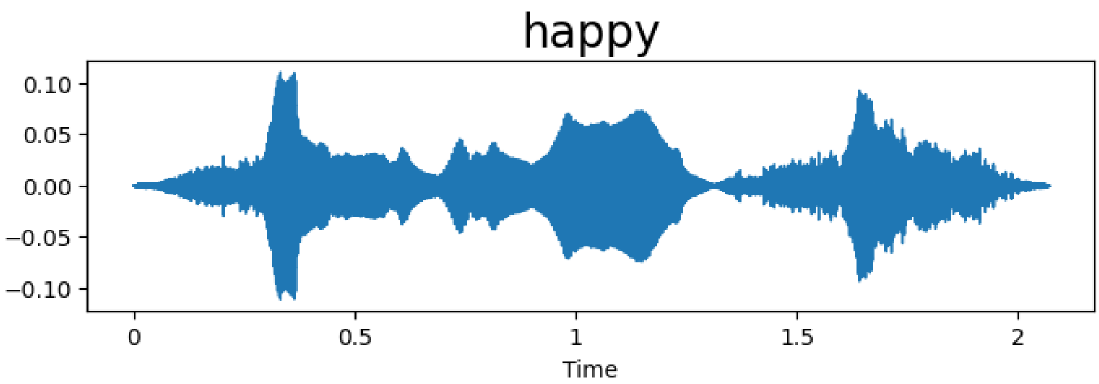
[246]:
```
emotion = 'sad'
path = np.array(df['speech'][df['label']==emotion])[0]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



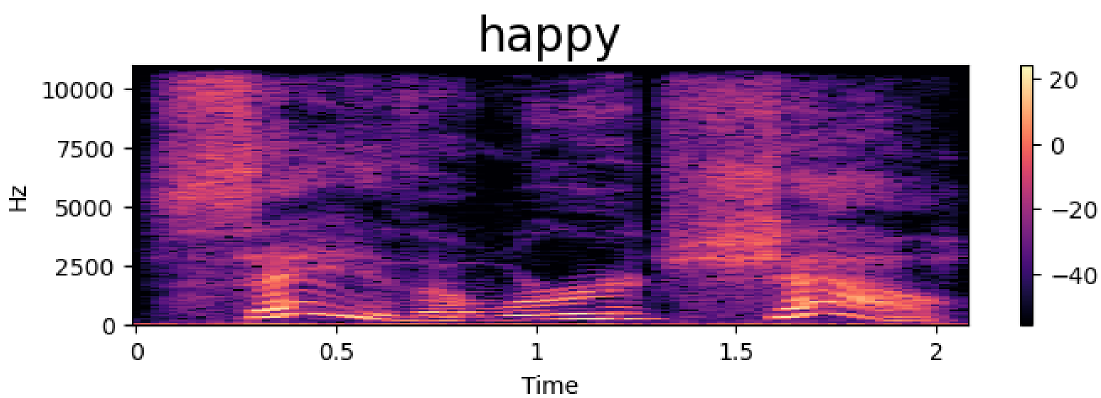
[246]: <IPython.lib.display.Audio object>



[248]:
```
emotion = 'happy'
path = np.array(df['speech'][df['label'] == emotion])[150]
data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectogram(data, sampling_rate, emotion)
Audio(path)
```



[248]: <IPython.lib.display.Audio object>



[250]:
```
################### # Data Augmentation Functions Adding more variations to the␣
↪training data(tech: time_stretch, pitch_shift, noise injection,)
def augment_audio(data, sr):
    # Randomly apply one or more augmentations
    if np.random.random() < 0.5:
        data = librosa.effects.time_stretch(data, rate=np.random.uniform(0.8, 1.
↪2))
    # if np.random.random() < 0.5:          # Pitch Shifting: Modify the pitch␣
↪of the audio.
    #     n_steps = np.random.uniform(-5, 5)      # Randomly shift pitch between␣
↪-5 and +5 semitones
    #     data = librosa.effects.pitch_shift(data, sr, n_steps)
    if np.random.random() < 0.5:
        noise = np.random.randn(len(data))     #Noise Injection: Add random␣
↪noise to the audio.
        data = data + 0.005 * noise
```

```
    if np.random.random() < 0.5:   # Time Shifting: Shift the audio signal in␣
↪time.
        shift = np.random.randint(-sr // 10, sr // 10)  # Shift by up to 0.1␣
↪seconds
        data = np.roll(data, shift)  # Roll the array elements
        if shift > 0:
            data[:shift] = 0   # # Zero out the empty indices if rolling␣
↪exceeds length
        else:
            data[shift:] = 0
    return data
```

[252]:
```
def extract_mfcc(filename, augment=False):
    y, sr = librosa.load(filename, duration=3, offset=0.5) # Load the audio␣
↪file with a maximum duration of 3 seconds and an offset of 0.5 seconds.

    # If we want to apply audio augmentation (changing the audio slightly), do␣
↪it here.
    if augment:
        y = augment_audio(y, sr)  # Call the augment_audio function to change␣
↪the audio.

        # Check if the augmented audio data is a 1D array (like a single line␣
↪of numbers).
        if y.ndim != 1:
            raise ValueError("Augmented audio data is not a 1D array.")  #␣
↪Raise an error if the shape is wrong.

    # Extract the MFCC features from the audio. # MFCC (Mel-Frequency Cepstral␣
↪Coefficients) are special features that help us understand the sound.
    mfcc = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40).T, axis=0)  #␣
↪Calculate and average the MFCCs.

    return mfcc
```

[254]:
```
extract_mfcc(df['speech'][0])
```

[254]:
```
array([-3.9698621e+02,  7.7440536e+01, -1.9592791e+01, -2.1666689e+01,
       -2.1127560e+00,  1.0075363e+01, -2.0366707e+01, -6.0924492e+00,
       -7.2122831e+00, -5.5736607e-01, -1.8325533e+00,  2.0210145e-01,
        7.2755075e-01,  1.3177377e+00,  2.8863375e+00,  2.8557906e+00,
       -4.7129216e+00, -4.4365110e+00, -1.6211596e+00, -1.0239839e+01,
       -7.5512629e+00, -1.7968802e+00, -7.0376525e+00,  9.4365845e+00,
        8.3558550e+00,  2.1712360e+01,  1.9216991e+01,  2.0348930e+01,
        1.3413366e+01,  8.3391724e+00,  3.9472219e-01,  5.1113148e+00,
```

```
   9.5687389e+00,  5.4548678e+00,  2.5099635e+00, -1.8239073e+00,
   4.8689618e+00,  9.3139229e+00,  2.0891502e+00, -1.9064913e+00],
 dtype=float32)
```

[256]:
```
# Now we extract features from all the audio files in the 'speech' column of
↪our DataFrame.
# We apply augmentation to some and keep the original audio for comparison.
X_mfcc_augmented = df['speech'].apply(lambda x: extract_mfcc(x, augment=True))
↪# Extract features with audio changes.
X_mfcc_original = df['speech'].apply(lambda x: extract_mfcc(x, augment=False))
↪# Extract features without changes.
```

[258]:
```
X_mfcc_augmented
```

[258]:
```
0       [-400.29376, 79.432945, -20.012371, -24.039331…
1       [-477.77942, 100.76463, -3.1481955, -36.190834…
2       [-429.79196, 46.12401, 1.5550478, -0.21709459,…
3       [-419.83862, 72.80277, -11.526895, -20.391947,…
4       [-441.13748, 69.31054, 7.8406296, 14.772851, 6…
                              …
2795    [-308.5871350344851, 16.14873085590186, 12.611…
2796    [-608.57043, 96.20862, 35.57011, 12.663162, 28…
2797    [-556.6771, 86.21727, 27.288998, 18.112206, 22…
2798    [-578.8551, 107.73781, 32.53618, 12.129623, 22…
2799    [-314.25272971051595, 16.505312321659105, 14.2…
Name: speech, Length: 2800, dtype: object
```

[260]:
```
X_mfcc_original
```

[260]:
```
0       [-396.9862, 77.44054, -19.59279, -21.666689, -…
1       [-465.73267, 98.77373, 0.6560086, -32.74544, -…
2       [-429.79196, 46.12401, 1.5550478, -0.21709459,…
3       [-403.46118, 76.32369, -12.531775, -22.288858,…
4       [-434.05756, 77.4455, 10.865501, 16.092943, 8…
                              …
2795    [-553.2201, 89.83577, 27.215466, 16.407124, 19…
2796    [-589.23676, 96.20407, 36.96118, 15.014446, 28…
2797    [-533.41815, 85.43242, 27.791998, 19.307178, 2…
2798    [-548.6142, 110.16424, 31.91024, 12.572518, 22…
2799    [-549.2962, 102.374565, 32.268833, 26.261614, …
Name: speech, Length: 2800, dtype: object
```

[262]:
```
X_mfcc_augmented.shape, X_mfcc_augmented.shape
```

[262]:
```
((2800,), (2800,))
```

[264]:
```
# Combine augmented and original MFCC features into one dataset.
# This helps us use both types of data (original and changed) for training our␣
↪model.
X_combined = pd.concat([X_mfcc_augmented, X_mfcc_original], axis=0)  #␣
↪Concatenate along rows (axis=0).
X = np.array([x for x in X_combined])  # Convert the combined features into a␣
↪NumPy array for easier processing.
X = np.expand_dims(X, -1)  # Add an extra dimension to make the shape␣
↪compatible with the LSTM model.
X.shape
```

[264]:
```
(5600, 40, 1)
```

[266]:
```
# Combine original and augmented labels & # Encoding the labels (emotion␣
↪classes) into one-hot encoded format
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
y_combined = pd.concat([df['label'], df['label']], axis=0).to_numpy()  #␣
↪Convert to NumPy array
# Fit the OneHotEncoder to our combined labels and transform them into a␣
↪one-hot encoded format.
# One-hot encoding turns labels into binary format, which is easier for the␣
↪model to understand.
y = enc.fit_transform(y_combined[:, np.newaxis]).toarray()
y, y.shape
```

[266]:
```
(array([[1., 0., 0., …, 0., 0., 0.],
        [1., 0., 0., …, 0., 0., 0.],
        [1., 0., 0., …, 0., 0., 0.],
        …,
        [0., 0., 0., …, 0., 0., 1.],
        [0., 0., 0., …, 0., 0., 1.],
        [0., 0., 0., …, 0., 0., 1.]]),
 (5600, 7))
```

[268]:
```
############## Creating/Building LSTM model for Speech Emotion Recognition
# Import the necessary libraries for building the LSTM model.
from keras.models import Sequential  # Import the Sequential model type.
from keras.layers import Dense, LSTM, Dropout  # Import layers for the model.

# Build the LSTM model using a sequential approach, layer by layer.
model = Sequential([
    # First LSTM layer with 256 units (neurons).
    # 'return_sequences=False' means the output will be the last output in the␣
↪sequence.
```

```
    # 'input_shape=(40,1)' specifies that the input data has 40 features and 1␣
 ↪time step.
    LSTM(256, return_sequences=False, input_shape=(40,1)),

    # First dense (fully connected) layer with 128 units and ReLU activation␣
 ↪function.
    # ReLU helps the model learn complex patterns by allowing it to output zero␣
 ↪for negative inputs.
    Dense(128, activation='relu'),

    # Second dense layer with 64 units and ReLU activation.
    Dense(64, activation='relu'),

    # Third dense layer with 64 units and ReLU activation.
    Dense(64, activation='relu'),

    # Dropout layer to prevent overfitting by randomly setting 20% of the␣
 ↪neurons to zero during training.
    Dropout(0.2),

    # Fourth dense layer with 32 units and ReLU activation.
    Dense(32, activation='relu'),

    # Another dropout layer to further help with regularization, reducing␣
 ↪overfitting.
    Dropout(0.1),

    # Final output layer with 7 units and softmax activation function.
    # Softmax converts the output to probabilities for 7 emotion classes (e.g.,␣
 ↪happy, sad, angry).
    Dense(7, activation='softmax')
])

# Compile the model by specifying the loss function, optimizer, and metrics to␣
 ↪track.
# 'categorical_crossentropy' is used for multi-class classification problems.
# 'adam' is a popular optimizer that adjusts the learning rate during training.
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[270]:

```
model.summary()
```

```
Model: "sequential_16"
```

```
Layer (type)                          Output Shape                    ␣
↪Param #

lstm_11 (LSTM)                        (None, 256)                     ␣
↪264,192

dense_48 (Dense)                      (None, 128)                      ␣
↪32,896

dense_49 (Dense)                      (None, 64)                        ␣
↪8,256

dense_50 (Dense)                      (None, 64)                        ␣
↪4,160

dropout_35 (Dropout)                  (None, 64)                          ␣
↪  0

dense_51 (Dense)                      (None, 32)                        ␣
↪2,080

dropout_36 (Dropout)                  (None, 32)                          ␣
↪  0

dense_52 (Dense)                      (None, 7)                           ␣
↪231
```

Total params: 311,815 (1.19 MB)

Trainable params: 311,815 (1.19 MB)

Non-trainable params: 0 (0.00 B)

[272]:

```
# IMPORT LIBRARIES TO MONITOR AND CONTROL TRAINING
from keras.callbacks import ModelCheckpoint, EarlyStopping  # Import tools to␣
 ↪control model training and save progress.
from sklearn.model_selection import train_test_split

# Split data into training and testing sets (80% for training, 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# ModelCheckpoint: Save the model during training whenever it improves.
```

```python
checkpoint = ModelCheckpoint("mymodel.keras", monitor='val_accuracy',␣
↪verbose=1, save_best_only=True, save_weights_only=False)

# EarlyStopping: Stop training early if the validation accuracy doesn't improve.
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=10,␣
↪verbose=1, mode='auto')

# Train the model and use the callbacks for monitoring:
history = model.fit(X_train, y_train,
                    validation_split=0.2,
                    epochs=50,
                    batch_size=64,
                    callbacks=[checkpoint, early])

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50
56/56                0s 89ms/step -
accuracy: 0.4058 - loss: 1.4739
Epoch 1: val_accuracy improved from -inf to 0.81138, saving model to
mymodel.keras
56/56                13s 115ms/step -
accuracy: 0.4087 - loss: 1.4671 - val_accuracy: 0.8114 - val_loss: 0.5647
Epoch 2/50
56/56                0s 91ms/step -
accuracy: 0.8356 - loss: 0.4711
Epoch 2: val_accuracy improved from 0.81138 to 0.93638, saving model to
mymodel.keras
56/56                6s 105ms/step -
accuracy: 0.8362 - loss: 0.4696 - val_accuracy: 0.9364 - val_loss: 0.1944
Epoch 3/50
56/56                0s 91ms/step -
accuracy: 0.9275 - loss: 0.2129
Epoch 3: val_accuracy did not improve from 0.93638
56/56                6s 106ms/step -
accuracy: 0.9275 - loss: 0.2128 - val_accuracy: 0.9353 - val_loss: 0.1683
Epoch 4/50
56/56                0s 81ms/step -
accuracy: 0.9259 - loss: 0.2230
Epoch 4: val_accuracy improved from 0.93638 to 0.94643, saving model to
mymodel.keras
56/56                5s 95ms/step -
accuracy: 0.9261 - loss: 0.2225 - val_accuracy: 0.9464 - val_loss: 0.1270
Epoch 5/50
56/56                0s 79ms/step -
```

```
accuracy: 0.9512 - loss: 0.1438
Epoch 5: val_accuracy improved from 0.94643 to 0.95536, saving model to
mymodel.keras
56/56            5s 92ms/step -
accuracy: 0.9512 - loss: 0.1440 - val_accuracy: 0.9554 - val_loss: 0.1133
Epoch 6/50
56/56            0s 90ms/step -
accuracy: 0.9545 - loss: 0.1452
Epoch 6: val_accuracy did not improve from 0.95536
56/56            6s 103ms/step -
accuracy: 0.9545 - loss: 0.1453 - val_accuracy: 0.9431 - val_loss: 0.1501
Epoch 7/50
56/56            0s 84ms/step -
accuracy: 0.9518 - loss: 0.1317
Epoch 7: val_accuracy improved from 0.95536 to 0.95871, saving model to
mymodel.keras
56/56            6s 97ms/step -
accuracy: 0.9520 - loss: 0.1313 - val_accuracy: 0.9587 - val_loss: 0.1219
Epoch 8/50
55/56            0s 69ms/step -
accuracy: 0.9697 - loss: 0.0896
Epoch 8: val_accuracy improved from 0.95871 to 0.96652, saving model to
mymodel.keras
56/56            4s 79ms/step -
accuracy: 0.9697 - loss: 0.0897 - val_accuracy: 0.9665 - val_loss: 0.0888
Epoch 9/50
56/56            0s 79ms/step -
accuracy: 0.9673 - loss: 0.1169
Epoch 9: val_accuracy did not improve from 0.96652
56/56            5s 95ms/step -
accuracy: 0.9673 - loss: 0.1169 - val_accuracy: 0.9632 - val_loss: 0.0944
Epoch 10/50
56/56            0s 81ms/step -
accuracy: 0.9690 - loss: 0.0984
Epoch 10: val_accuracy did not improve from 0.96652
56/56            5s 95ms/step -
accuracy: 0.9690 - loss: 0.0985 - val_accuracy: 0.9420 - val_loss: 0.2218
Epoch 11/50
56/56            0s 85ms/step -
accuracy: 0.9636 - loss: 0.1249
Epoch 11: val_accuracy improved from 0.96652 to 0.96875, saving model to
mymodel.keras
56/56            6s 100ms/step -
accuracy: 0.9636 - loss: 0.1247 - val_accuracy: 0.9688 - val_loss: 0.0906
Epoch 12/50
56/56            0s 62ms/step -
accuracy: 0.9683 - loss: 0.1005
Epoch 12: val_accuracy improved from 0.96875 to 0.97433, saving model to
```

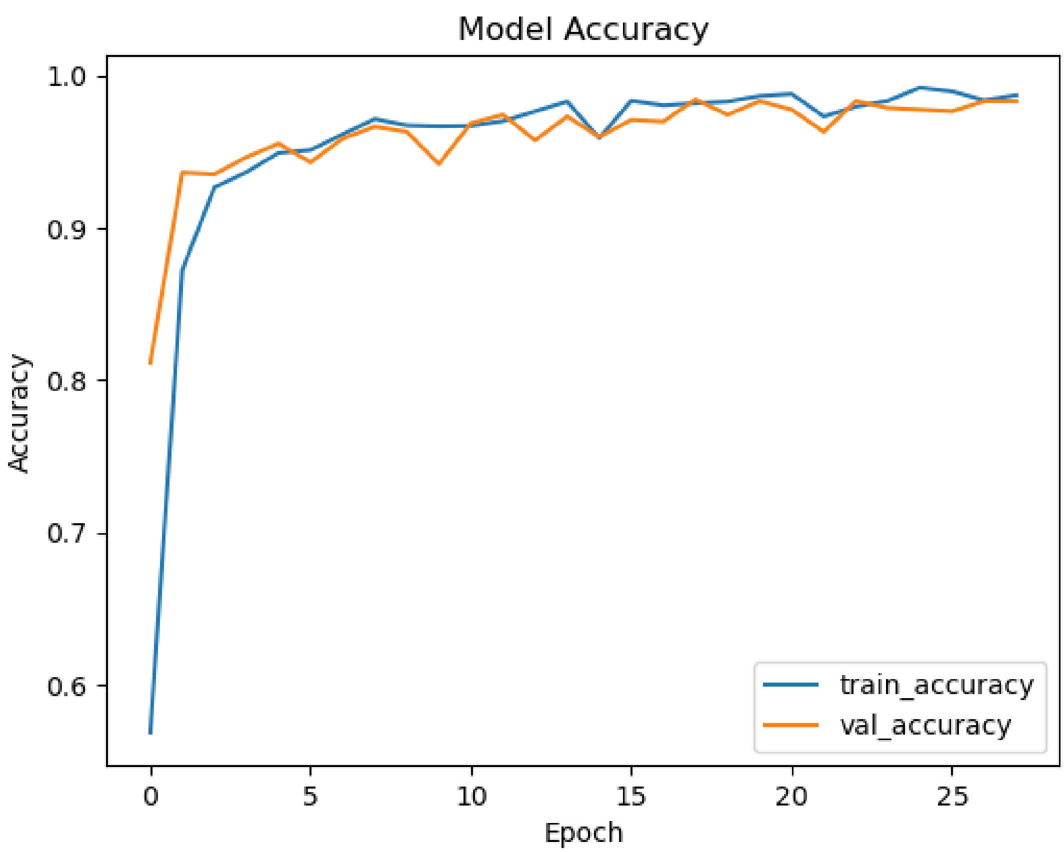
```
mymodel.keras
56/56             4s 75ms/step -
accuracy: 0.9683 - loss: 0.1004 - val_accuracy: 0.9743 - val_loss: 0.0713
Epoch 13/50
56/56             0s 75ms/step -
accuracy: 0.9784 - loss: 0.0647
Epoch 13: val_accuracy did not improve from 0.97433
56/56             5s 86ms/step -
accuracy: 0.9783 - loss: 0.0649 - val_accuracy: 0.9576 - val_loss: 0.1308
Epoch 14/50
56/56             0s 84ms/step -
accuracy: 0.9826 - loss: 0.0615
Epoch 14: val_accuracy did not improve from 0.97433
56/56             6s 98ms/step -
accuracy: 0.9826 - loss: 0.0613 - val_accuracy: 0.9732 - val_loss: 0.0892
Epoch 15/50
56/56             0s 87ms/step -
accuracy: 0.9620 - loss: 0.1225
Epoch 15: val_accuracy did not improve from 0.97433
56/56             6s 98ms/step -
accuracy: 0.9620 - loss: 0.1226 - val_accuracy: 0.9598 - val_loss: 0.1172
Epoch 16/50
56/56             0s 86ms/step -
accuracy: 0.9847 - loss: 0.0614
Epoch 16: val_accuracy did not improve from 0.97433
56/56             6s 100ms/step -
accuracy: 0.9847 - loss: 0.0614 - val_accuracy: 0.9710 - val_loss: 0.0881
Epoch 17/50
56/56             0s 82ms/step -
accuracy: 0.9807 - loss: 0.0604
Epoch 17: val_accuracy did not improve from 0.97433
56/56             5s 92ms/step -
accuracy: 0.9807 - loss: 0.0604 - val_accuracy: 0.9699 - val_loss: 0.1055
Epoch 18/50
56/56             0s 77ms/step -
accuracy: 0.9814 - loss: 0.0640
Epoch 18: val_accuracy improved from 0.97433 to 0.98438, saving model to
mymodel.keras
56/56             5s 92ms/step -
accuracy: 0.9814 - loss: 0.0640 - val_accuracy: 0.9844 - val_loss: 0.0550
Epoch 19/50
56/56             0s 85ms/step -
accuracy: 0.9814 - loss: 0.0594
Epoch 19: val_accuracy did not improve from 0.98438
56/56             6s 98ms/step -
accuracy: 0.9814 - loss: 0.0594 - val_accuracy: 0.9743 - val_loss: 0.0574
Epoch 20/50
56/56             0s 83ms/step -
```

```
accuracy: 0.9897 - loss: 0.0299
Epoch 20: val_accuracy did not improve from 0.98438
56/56              6s 98ms/step -
accuracy: 0.9896 - loss: 0.0302 - val_accuracy: 0.9833 - val_loss: 0.0471
Epoch 21/50
56/56              0s 76ms/step -
accuracy: 0.9885 - loss: 0.0386
Epoch 21: val_accuracy did not improve from 0.98438
56/56              5s 87ms/step -
accuracy: 0.9885 - loss: 0.0386 - val_accuracy: 0.9777 - val_loss: 0.0725
Epoch 22/50
56/56              0s 67ms/step -
accuracy: 0.9776 - loss: 0.0709
Epoch 22: val_accuracy did not improve from 0.98438
56/56              5s 80ms/step -
accuracy: 0.9775 - loss: 0.0711 - val_accuracy: 0.9632 - val_loss: 0.1301
Epoch 23/50
56/56              0s 85ms/step -
accuracy: 0.9749 - loss: 0.0908
Epoch 23: val_accuracy did not improve from 0.98438
56/56              6s 98ms/step -
accuracy: 0.9750 - loss: 0.0904 - val_accuracy: 0.9833 - val_loss: 0.0594
Epoch 24/50
56/56              0s 87ms/step -
accuracy: 0.9847 - loss: 0.0519
Epoch 24: val_accuracy did not improve from 0.98438
56/56              6s 101ms/step -
accuracy: 0.9847 - loss: 0.0519 - val_accuracy: 0.9788 - val_loss: 0.0539
Epoch 25/50
56/56              0s 64ms/step -
accuracy: 0.9946 - loss: 0.0201
Epoch 25: val_accuracy did not improve from 0.98438
56/56              4s 75ms/step -
accuracy: 0.9946 - loss: 0.0202 - val_accuracy: 0.9777 - val_loss: 0.0838
Epoch 26/50
55/56              0s 61ms/step -
accuracy: 0.9895 - loss: 0.0422
Epoch 26: val_accuracy did not improve from 0.98438
56/56              4s 73ms/step -
accuracy: 0.9895 - loss: 0.0420 - val_accuracy: 0.9766 - val_loss: 0.0680
Epoch 27/50
56/56              0s 89ms/step -
accuracy: 0.9848 - loss: 0.0480
Epoch 27: val_accuracy did not improve from 0.98438
56/56              6s 104ms/step -
accuracy: 0.9848 - loss: 0.0480 - val_accuracy: 0.9833 - val_loss: 0.0689
Epoch 28/50
56/56              0s 80ms/step -
```

```
accuracy: 0.9932 - loss: 0.0256
Epoch 28: val_accuracy did not improve from 0.98438
56/56              5s 92ms/step -
accuracy: 0.9931 - loss: 0.0260 - val_accuracy: 0.9833 - val_loss: 0.0572
Epoch 28: early stopping
35/35              2s 33ms/step -
accuracy: 0.9778 - loss: 0.0592
Test accuracy: 97.59%
```

[274]:

```python
# Plotting the training and validation accuracy across epochs
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[ ]: