```python
import pandas as pd
import numpy as np
import os #deal with files

#visuals
import seaborn as sns #built on top of matplotlib with similar functionalities
import matplotlib.pyplot as plt

import librosa
import librosa.display

#audio play
from IPython.display import Audio
import warnings
warnings.filterwarnings('ignore')
```

1. Load the Dataset

```python
import os
import kagglehub

# Tải bộ dữ liệu và lấy đường dẫn
path = kagglehub.dataset_download("ejlok1/toronto-emotional-speech-set-tess")
print("Đường dẫn tới bộ dữ liệu:", path)

paths = []
labels = []

for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        # Bỏ qua các file không phải là file âm thanh
        if not filename.endswith('.wav'):
            continue

        paths.append(os.path.join(dirname, filename))

        # Trích xuất label từ tên file
        # Ví dụ tên file: "OAF_sad_sad.wav" -> label là "sad"
        label = filename.split('_')[2]
        label = label.split('.')[0]
        labels.append(label.lower())

# Xóa điều kiện dừng sớm nếu bạn muốn duyệt qua toàn bộ dataset
# if len(paths) == 2800:
#     break
```

```
print('Dataset is Loaded')
```

```
Đường dẫn tới bộ dữ liệu: /kaggle/input/toronto-emotional-speech-set-
tess
Dataset is Loaded
```

```
labels[:5]
```

```
['fear', 'fear', 'fear', 'fear', 'fear']
```

```
paths[:5]
```

```
['/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto
emotional speech set data/YAF_fear/YAF_home_fear.wav',
 '/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto
emotional speech set data/YAF_fear/YAF_youth_fear.wav',
 '/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto
emotional speech set data/YAF_fear/YAF_near_fear.wav',
 '/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto
emotional speech set data/YAF_fear/YAF_search_fear.wav',
 '/kaggle/input/toronto-emotional-speech-set-tess/TESS Toronto
emotional speech set data/YAF_fear/YAF_pick_fear.wav']
```

2. Tạo dataframe

```
df = pd.DataFrame()
df['speech'] = paths  # Input: Cột 'speech' chứa đường dẫn đến file
âm thanh
df['label'] = labels  # Output: # Cột 'label' chứa nhãn cảm xúc
df.head()
```

```
                                              speech label
0  /kaggle/input/toronto-emotional-speech-set-tes...  fear
1  /kaggle/input/toronto-emotional-speech-set-tes...  fear
2  /kaggle/input/toronto-emotional-speech-set-tes...  fear
3  /kaggle/input/toronto-emotional-speech-set-tes...  fear
4  /kaggle/input/toronto-emotional-speech-set-tes...  fear
```

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

```
label
fear       800
angry      800
disgust    800
neutral    800
sad        800
ps         800
```

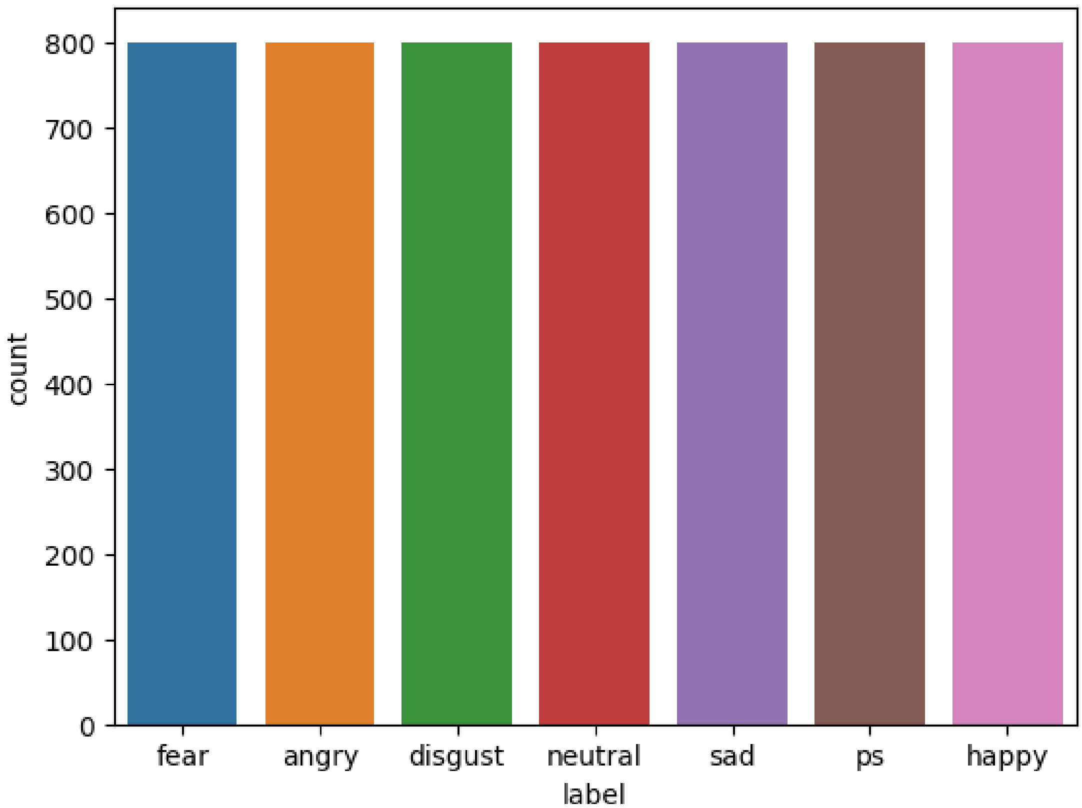
```
happy      800
Name: count, dtype: int64
```

Như vậy ta đã load dataset và chia frame thành công!

3. Exploratory Data Analysis

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

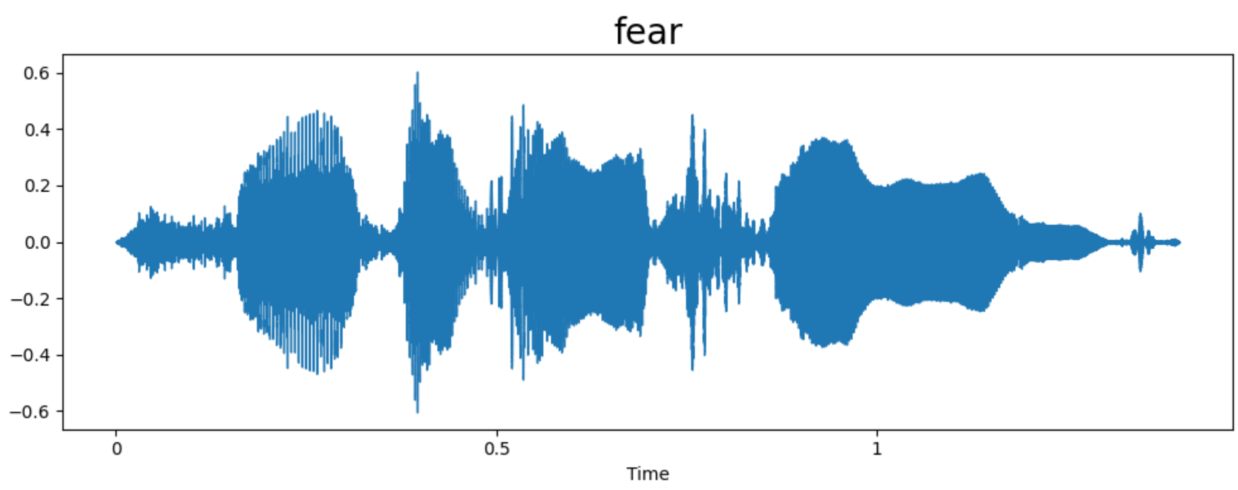
```
<Axes: xlabel='label', ylabel='count'>
```

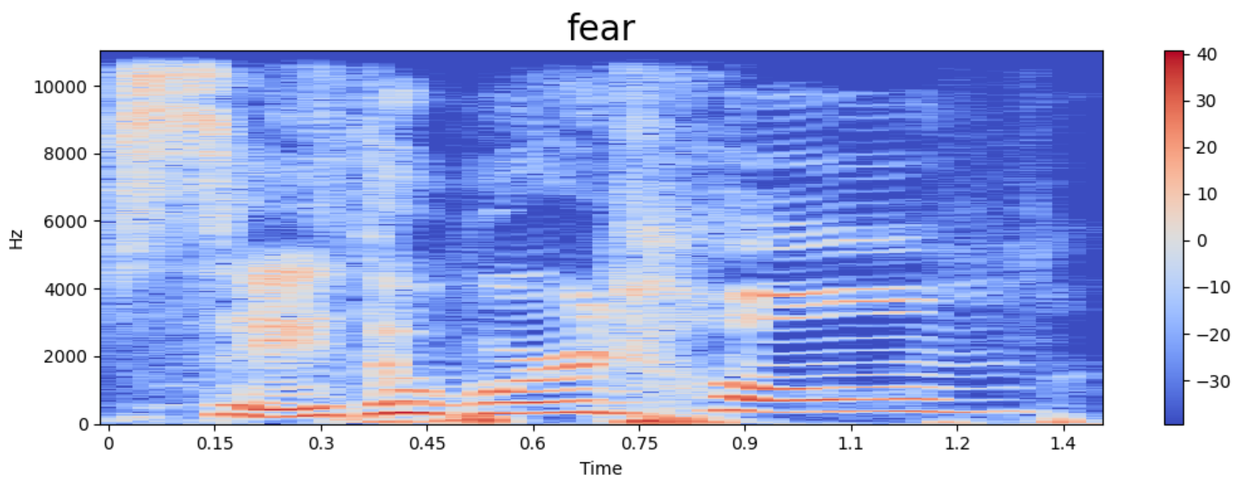


Viết function để vẽ waveplot và spectrogram

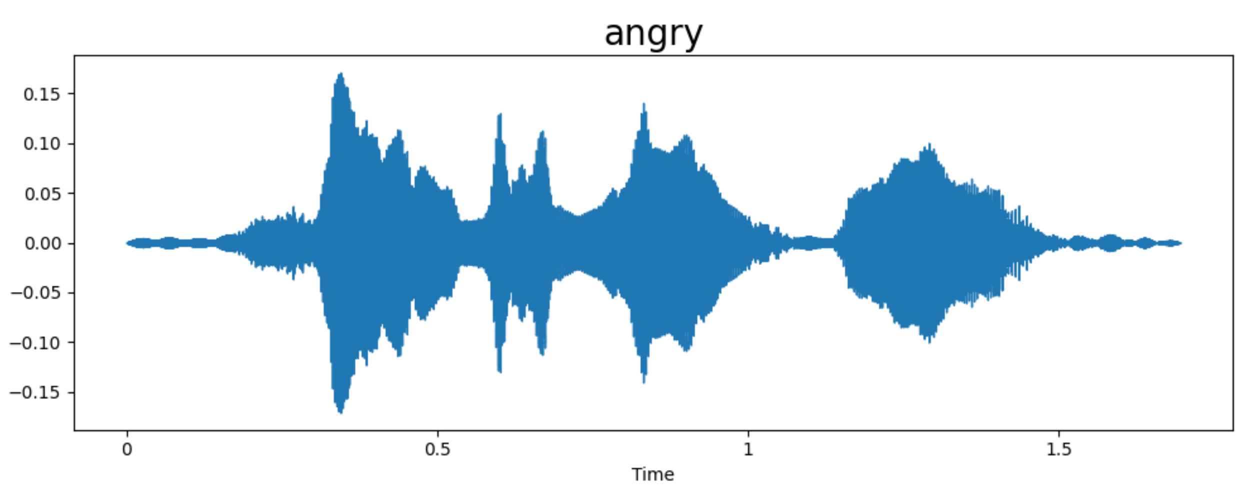
```
def waveplot(data, sr, emotion):
    plt.figure(figsize=(10,4))
    plt.title(emotion, size=20)
    librosa.display.waveshow(data, sr=sr) #data: mảng numpy
    plt.tight_layout()
    plt.show()

def spectrogram(data, sr, emotion):
    # 1. Áp dụng phép biến đổi Fourier thời gian ngắn (STFT) để
```

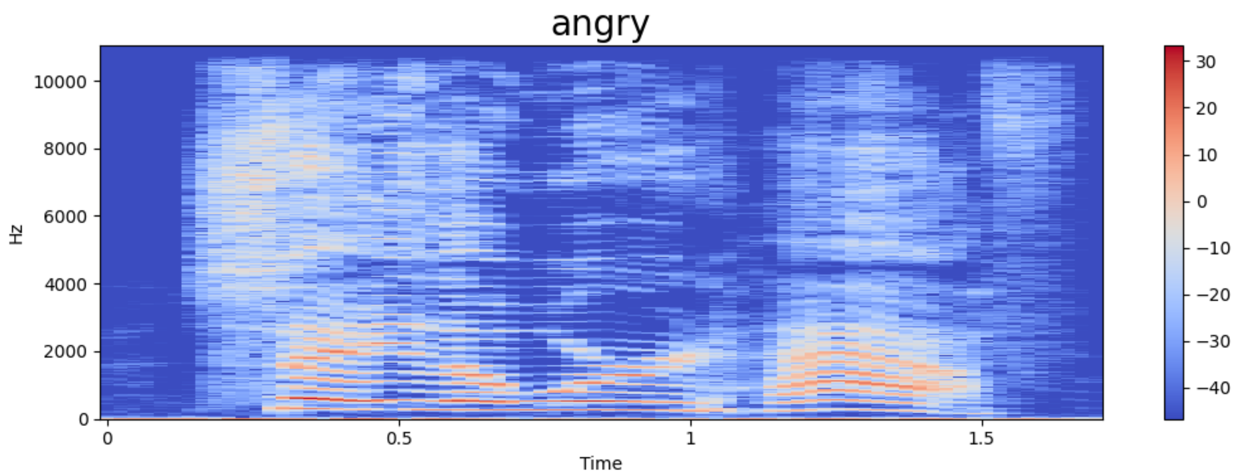
```
phân tích tần số.
    x = librosa.stft(data)

    # 2. Chuyển đổi biên độ sang thang đo decibel (dB) để dễ quan
sát hơn
    xdb = librosa.amplitude_to_db(abs(x))

    plt.figure(figsize=(11,4))
    plt.title(emotion, size=20)

    #Vẽ biểu đồ phổ.
    #xdb: Dữ liệu phổ đã chuyển sang dB.
    #sr: Tỷ lệ lấy mẫu.
    #x_axis': Đặt trục x là 'time' (thời gian).
    #y_axis': Đặt trục y là 'hz' (tần số).
    librosa.display.specshow(xdb, sr=sr, x_axis='time', y_axis='hz')
    plt.colorbar()
    plt.tight_layout()
    plt.show()

emotion = 'fear'

# "Trong bảng dữ liệu df, hãy tìm tất cả các hàng có nhãn cảm xúc
là fear, sau đó
# lấy ra danh sách các đường dẫn speech tương ứng, và cuối cùng
chỉ chọn lấy đường
# dẫn đầu tiên trong danh sách đó."
path = np.array(df['speech'][df['label']==emotion])[0]

data, sampling_rate = librosa.load(path)
waveplot(data, sampling_rate, emotion)
spectrogram(data, sampling_rate, emotion)
Audio(path)
```



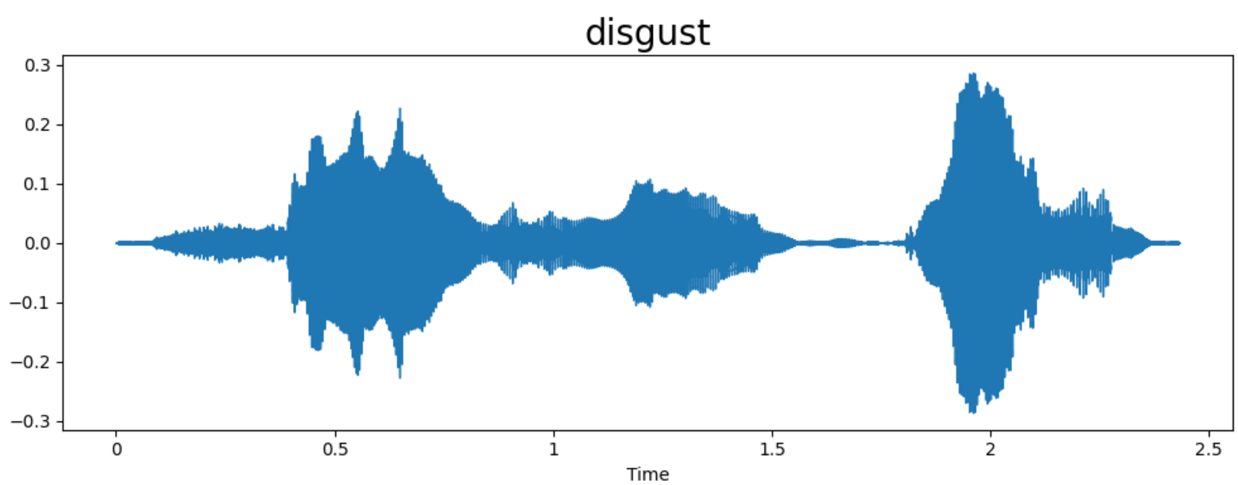


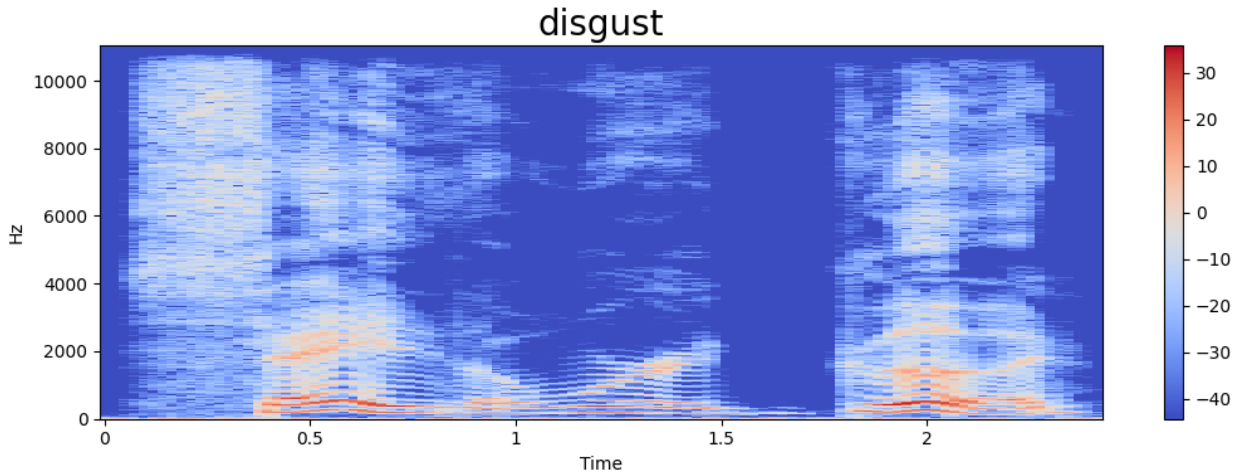

```
<IPython.lib.display.Audio object>

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



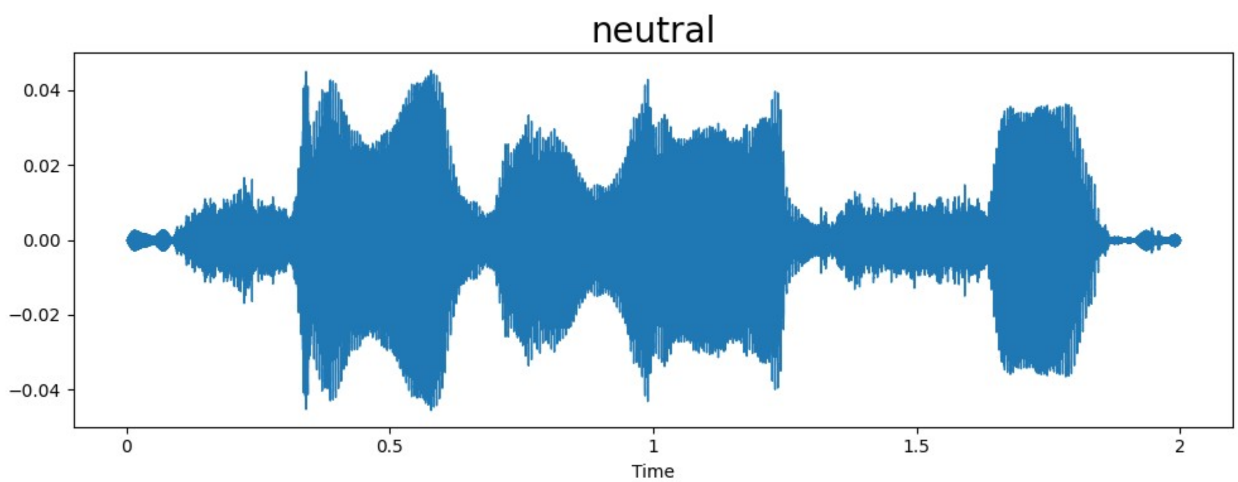


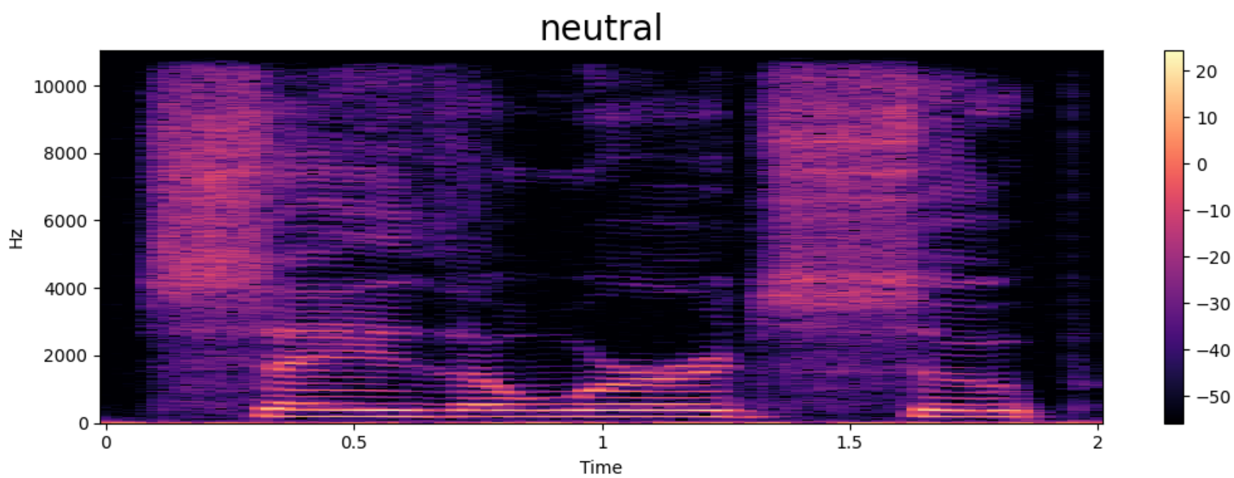

```
<IPython.lib.display.Audio object>

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



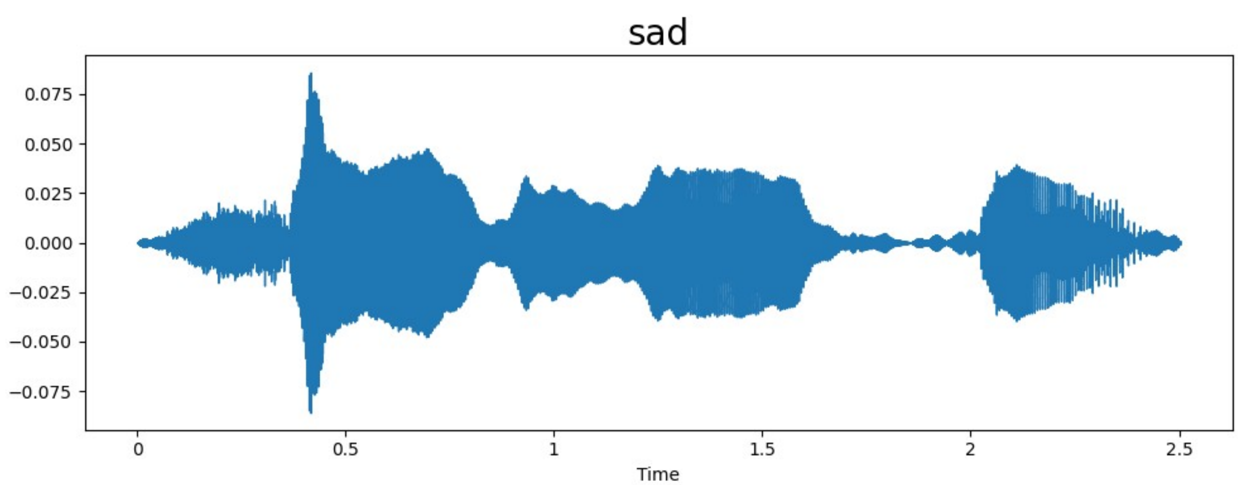


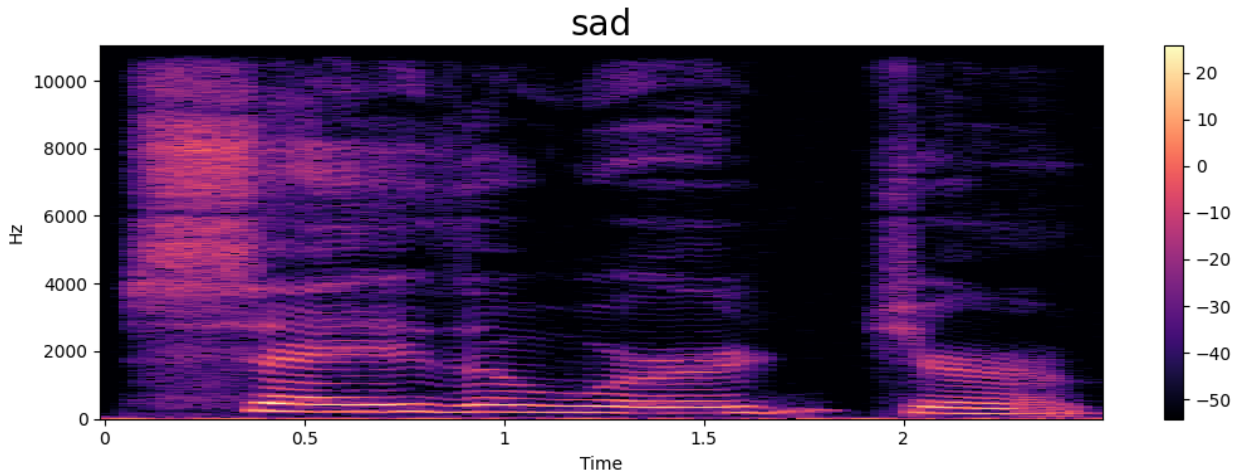

```
<IPython.lib.display.Audio object>

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



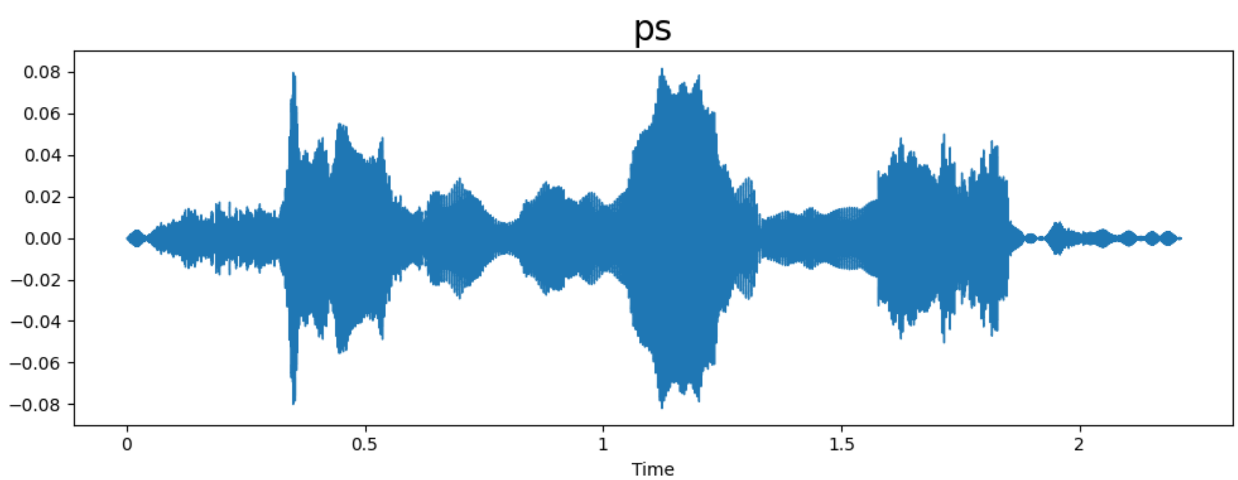


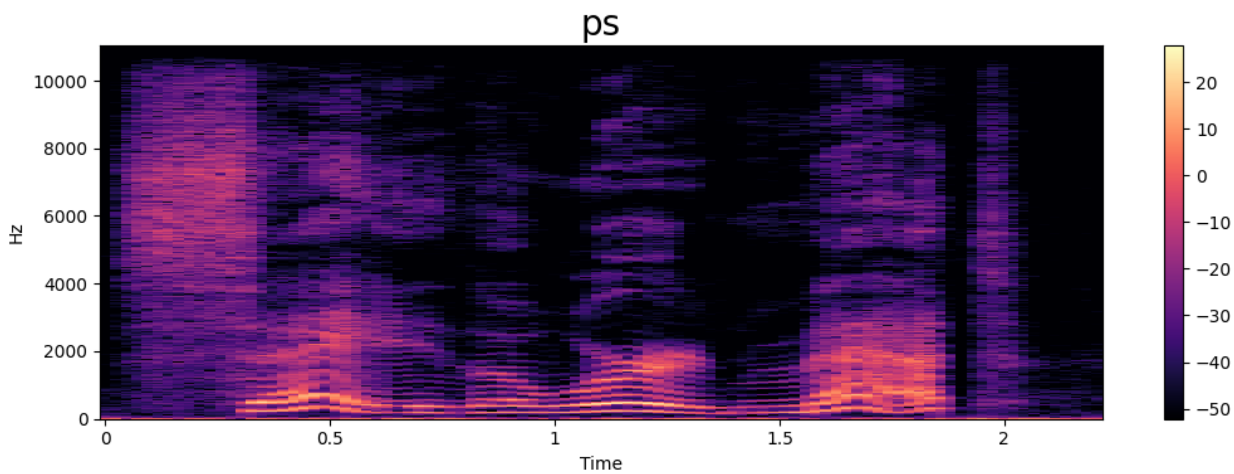

```
<IPython.lib.display.Audio object>

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



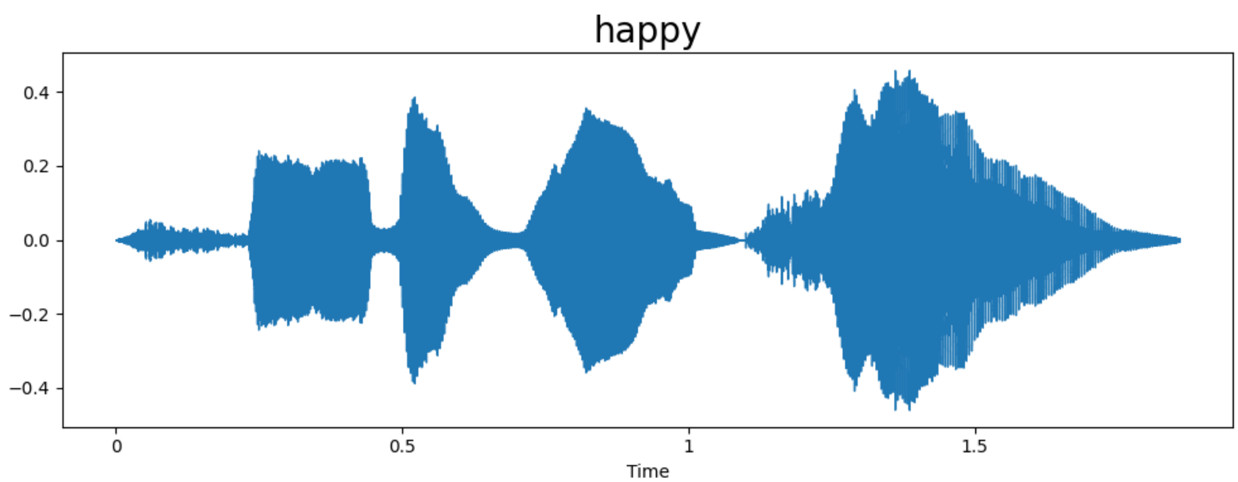


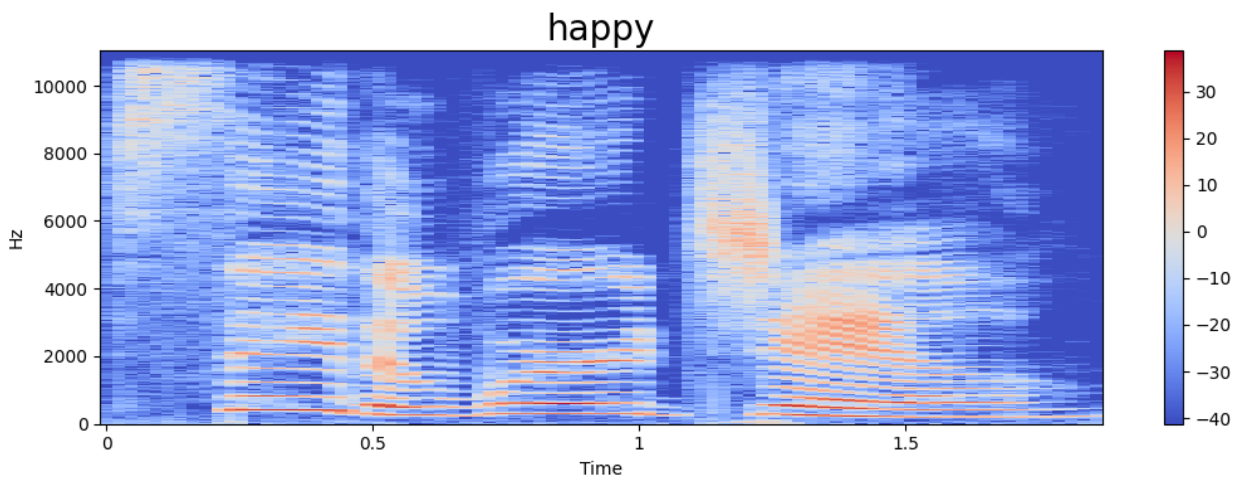

```
<IPython.lib.display.Audio object>

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






```
<IPython.lib.display.Audio object>

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






```
<IPython.lib.display.Audio object>
```

4. Feature Extraction

a. Viết hàm extract mfcc

- np.mean - Tóm tắt đặc trưng: Lấy giá trị trung bình của 40 hệ số đó qua toàn bộ 3 giây để tạo ra một "dấu vân tay" duy nhất cho tệp âm thanh đó – một vector gồm 40 con số.
- Kết quả trả về: trả về mảng mfcc (vector 40 chiều) vừa được tính toán. Vector này tóm tắt đặc trưng âm sắc của 3 giây âm thanh đã xử lý.

```python
def extract_mfcc(filename):
    # đọc một đoạn âm thanh kéo dài 3 giây, bắt đầu từ thời điểm 
0.5 giây
    # đến 3.5 giây của tệp gốc.
    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

# phần tử đầu tiên (dòng số 0) trong cột 'speech' của bảng dữ 
liệu df.
extract_mfcc(df['speech'][0])
```

```
array([-285.73727   ,   85.78295   ,   -2.1689117 ,   22.125532  ,
        -14.757396  ,   11.051346  ,   12.41245   ,   -3.000262  ,
          1.0844991 ,   11.078273  ,  -17.419657  ,   -8.093213  ,
          6.5879726 ,   -4.220953  ,   -9.155081  ,    3.5214806 ,
        -13.186381  ,   14.078853  ,   19.66973   ,   22.725618  ,
         32.574635  ,   16.325033  ,   -3.8427296 ,    0.89629626,
        -11.239264  ,    6.653462  ,   -2.5883694 ,   -7.714016  ,
        -10.941658  ,   -2.4007545 ,   -5.281286  ,    4.271157  ,
```

```
        -11.202216   ,    -9.024621   ,    -3.6669843 ,     4.8697433 ,
         -1.602798   ,     2.5600514 ,    11.454374   ,    11.233449   ],
      dtype=float32)
```

b. Áp dụng công thức lên từng file audio trong dataset:

- Kết quả: X_mfcc, là một danh sách chứa các "dấu vân tay" (các vector 40 chiều) của tất cả các tệp âm thanh.

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

X_mfcc

0       [-285.73727, 85.78295, -2.1689117, 22.125532, ...
1       [-348.34332, 35.193233, -3.8413274, 14.658876,...
2       [-340.11435, 53.79644, -14.267782, 20.884027, ...
3       [-306.63422, 21.259705, -4.4110823, 6.487154, ...
4       [-344.7548, 46.329193, -24.171413, 19.392921, ...
                              ...
5595    [-374.3952, 60.864998, 0.02505878, 8.431058, -...
5596    [-313.96478, 39.847843, -5.649306, -3.8675754,...
5597    [-357.54883, 77.88606, -15.224756, 2.1946328, ...
5598    [-353.1474, 101.6839, -14.175897, -12.037376, ...
5599    [-389.4595, 54.042767, 1.346998, -1.4258981, -...
Name: speech, Length: 5600, dtype: object
```

c. Gom tất cả lại thành một bảng (np.array):

- Lấy tất cả các vector 40 chiều riêng lẻ từ danh sách trên và "xếp chồng" chúng lên nhau để tạo thành một bảng dữ liệu (ma trận 2D) duy nhất.

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

(5600, 40)
```

d. Định dạng lại cho mô hình (np.expand_dims):

- Đây là bước cuối cùng để dữ liệu tương thích với các mô hình như LSTM: thêm một chiều nữa vào ma trận 2D để biến nó thành một khối dữ liệu 3D.
- Mục đích chính là để định dạng dữ liệu cho phù hợp với đầu vào của các mô hình học sâu (Deep Learning). Các lớp LSTM trong Keras/TensorFlow yêu cầu đầu vào phải là một tensor 3D có dạng: (batch_size, timesteps, features)

Ý nghĩa của 3 chiều:

- Chiều thứ 1 (5600): Số lượng mẫu trong bộ dữ liệu.
- Chiều thứ 2 (40): Số bước trong chuỗi dữ liệu (tương ứng 40 đặc trưng MFCC).
- Chiều thứ 3 (1): Số lượng kênh đặc trưng tại mỗi bước (ở đây chỉ có 1).

```
# thêm một chiều mới vào mảng X (expand dimension) để chuyển
# mảng X thành một mảng input được chấp nhận bởi model LSTM
X = np.expand_dims(X, -1) # -1: thêm vào vị trí cuối cùng

X.shape
#1st dim: length of the dataset
#2nd dim: column length

(5600, 40, 1)
```

e. One-Hot Encoding

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

#thứ tự các label (kiểm tra cho phần code dưới cùng)
print(enc.categories_)

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

y = y.toarray()

y.shape

(5600, 7)
```

5. Create LSTM Model

Dùng thư viện Keras để xây dựng và cấu hình một mô hình Recurrent Neural Network - RNN, cụ thể là sử dụng lớp LSTM, cho bài toán phân loại.

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

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

model.compile(loss='categorical_crossentropy', optimizer='adam',
```

```
metrics=['accuracy'])
model.summary()
```

```
2025-06-11 10:07:00.171592: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1749636420.364406      19 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1749636420.424244      19 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
I0000 00:00:1749636432.810369      19 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 15513 MB
memory:  -> device: 0, name: Tesla P100-PCIE-16GB, pci bus id:
0000:00:04.0, compute capability: 6.0
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ lstm (LSTM)                          │ (None, 256)                 │
264,192 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout (Dropout)                    │ (None, 256)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense (Dense)                        │ (None, 128)                 │
32,896 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 128)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_1 (Dense)                      │ (None, 64)                  │
8,256 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_2 (Dropout)                  │ (None, 64)                  │
0 │
```

```
├─────────────────────────────────────┼────────────────────────────┼
─────────────────┤
│ dense_2 (Dense)                     │ (None, 7)                  │
455 │
└─────────────────────────────────────┴────────────────────────────┴
─────────────────┘

 Total params: 305,799 (1.17 MB)

 Trainable params: 305,799 (1.17 MB)

 Non-trainable params: 0 (0.00 B)

#Train the model
history = model.fit(X, y, validation_split=0.2,
                    epochs=50, batch_size=64)

Epoch 1/50

I0000 00:00:1749636437.265356      72 cuda_dnn.cc:529] Loaded cuDNN
version 90300

70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5347 - loss:
1.2677 - val_accuracy: 0.7196 - val_loss: 0.6941
Epoch 2/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8965 - loss:
0.3082 - val_accuracy: 0.9339 - val_loss: 0.2338
Epoch 3/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9395 - loss:
0.1914 - val_accuracy: 0.9062 - val_loss: 0.3513
Epoch 4/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9578 - loss:
0.1304 - val_accuracy: 0.9580 - val_loss: 0.1240
Epoch 5/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9655 - loss:
0.1095 - val_accuracy: 0.9732 - val_loss: 0.0713
Epoch 6/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9787 - loss:
0.0692 - val_accuracy: 0.9464 - val_loss: 0.1475
Epoch 7/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9800 - loss:
0.0700 - val_accuracy: 0.9134 - val_loss: 0.3015
Epoch 8/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9795 - loss:
0.0764 - val_accuracy: 0.9768 - val_loss: 0.0745
Epoch 9/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9814 - loss:
0.0578 - val_accuracy: 0.9598 - val_loss: 0.1202
Epoch 10/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9829 - loss:
0.0617 - val_accuracy: 0.9714 - val_loss: 0.0610
```

```
Epoch 11/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9834 - loss:
0.0528 - val_accuracy: 0.9759 - val_loss: 0.0574
Epoch 12/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9834 - loss:
0.0480 - val_accuracy: 0.9696 - val_loss: 0.0710
Epoch 13/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9819 - loss:
0.0581 - val_accuracy: 0.9884 - val_loss: 0.0391
Epoch 14/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9793 - loss:
0.0710 - val_accuracy: 0.9884 - val_loss: 0.0334
Epoch 15/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9870 - loss:
0.0414 - val_accuracy: 0.9795 - val_loss: 0.0523
Epoch 16/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9890 - loss:
0.0362 - val_accuracy: 0.9937 - val_loss: 0.0149
Epoch 17/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9972 - loss:
0.0111 - val_accuracy: 0.9777 - val_loss: 0.0472
Epoch 18/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9907 - loss:
0.0270 - val_accuracy: 0.9946 - val_loss: 0.0165
Epoch 19/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9930 - loss:
0.0176 - val_accuracy: 0.9607 - val_loss: 0.1308
Epoch 20/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9762 - loss:
0.0987 - val_accuracy: 0.9679 - val_loss: 0.0878
Epoch 21/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9904 - loss:
0.0314 - val_accuracy: 0.9920 - val_loss: 0.0189
Epoch 22/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9954 - loss:
0.0176 - val_accuracy: 0.9893 - val_loss: 0.0329
Epoch 23/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9962 - loss:
0.0151 - val_accuracy: 0.9920 - val_loss: 0.0224
Epoch 24/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9929 - loss:
0.0219 - val_accuracy: 0.9723 - val_loss: 0.0734
Epoch 25/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9907 - loss:
0.0360 - val_accuracy: 0.9643 - val_loss: 0.0931
Epoch 26/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9854 - loss:
0.0452 - val_accuracy: 0.9937 - val_loss: 0.0220
Epoch 27/50
```

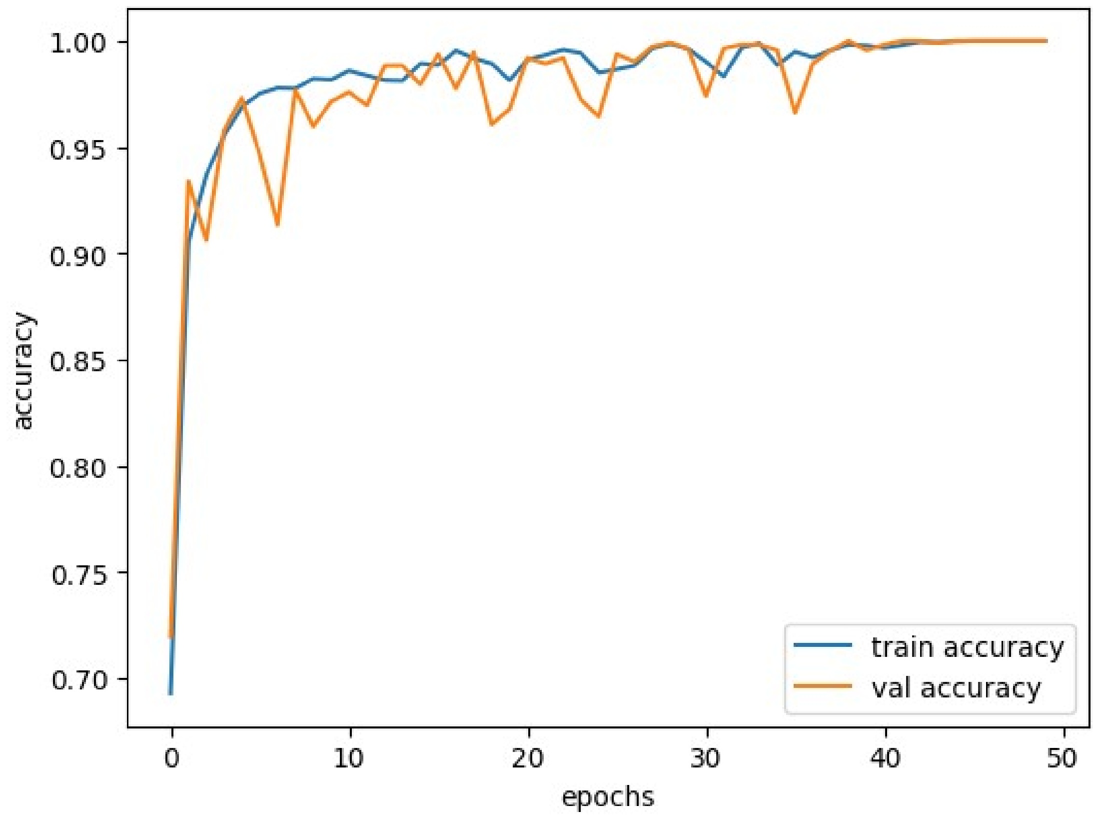
```
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9871 - loss: 0.0454 - val_accuracy: 0.9902 - val_loss: 0.0310
Epoch 28/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9973 - loss: 0.0124 - val_accuracy: 0.9973 - val_loss: 0.0117
Epoch 29/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9980 - loss: 0.0068 - val_accuracy: 0.9991 - val_loss: 0.0034
Epoch 30/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9975 - loss: 0.0086 - val_accuracy: 0.9964 - val_loss: 0.0111
Epoch 31/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9960 - loss: 0.0161 - val_accuracy: 0.9741 - val_loss: 0.0781
Epoch 32/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9795 - loss: 0.0573 - val_accuracy: 0.9964 - val_loss: 0.0129
Epoch 33/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9956 - loss: 0.0149 - val_accuracy: 0.9982 - val_loss: 0.0057
Epoch 34/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9998 - loss: 0.0029 - val_accuracy: 0.9982 - val_loss: 0.0051
Epoch 35/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9904 - loss: 0.0403 - val_accuracy: 0.9955 - val_loss: 0.0131
Epoch 36/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9957 - loss: 0.0129 - val_accuracy: 0.9661 - val_loss: 0.1242
Epoch 37/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9915 - loss: 0.0285 - val_accuracy: 0.9893 - val_loss: 0.0317
Epoch 38/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9961 - loss: 0.0137 - val_accuracy: 0.9955 - val_loss: 0.0125
Epoch 39/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9989 - loss: 0.0044 - val_accuracy: 1.0000 - val_loss: 0.0016
Epoch 40/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9989 - loss: 0.0048 - val_accuracy: 0.9955 - val_loss: 0.0197
Epoch 41/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9966 - loss: 0.0111 - val_accuracy: 0.9982 - val_loss: 0.0032
Epoch 42/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9990 - loss: 0.0042 - val_accuracy: 1.0000 - val_loss: 0.0014
Epoch 43/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 1.0000 - loss:
```

```
0.0016 - val_accuracy: 1.0000 - val_loss: 5.8243e-04
Epoch 44/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9995 - loss: 
0.0011 - val_accuracy: 0.9991 - val_loss: 0.0085
Epoch 45/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9996 - loss: 
0.0055 - val_accuracy: 1.0000 - val_loss: 5.5832e-04
Epoch 46/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 
3.6494e-04 - val_accuracy: 1.0000 - val_loss: 1.8547e-04
Epoch 47/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 
5.5885e-04 - val_accuracy: 1.0000 - val_loss: 1.4839e-04
Epoch 48/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 
1.5301e-04 - val_accuracy: 1.0000 - val_loss: 1.4719e-04
Epoch 49/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 
2.9121e-04 - val_accuracy: 1.0000 - val_loss: 1.1288e-04
Epoch 50/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 
3.1666e-04 - val_accuracy: 1.0000 - val_loss: 5.6901e-05
```

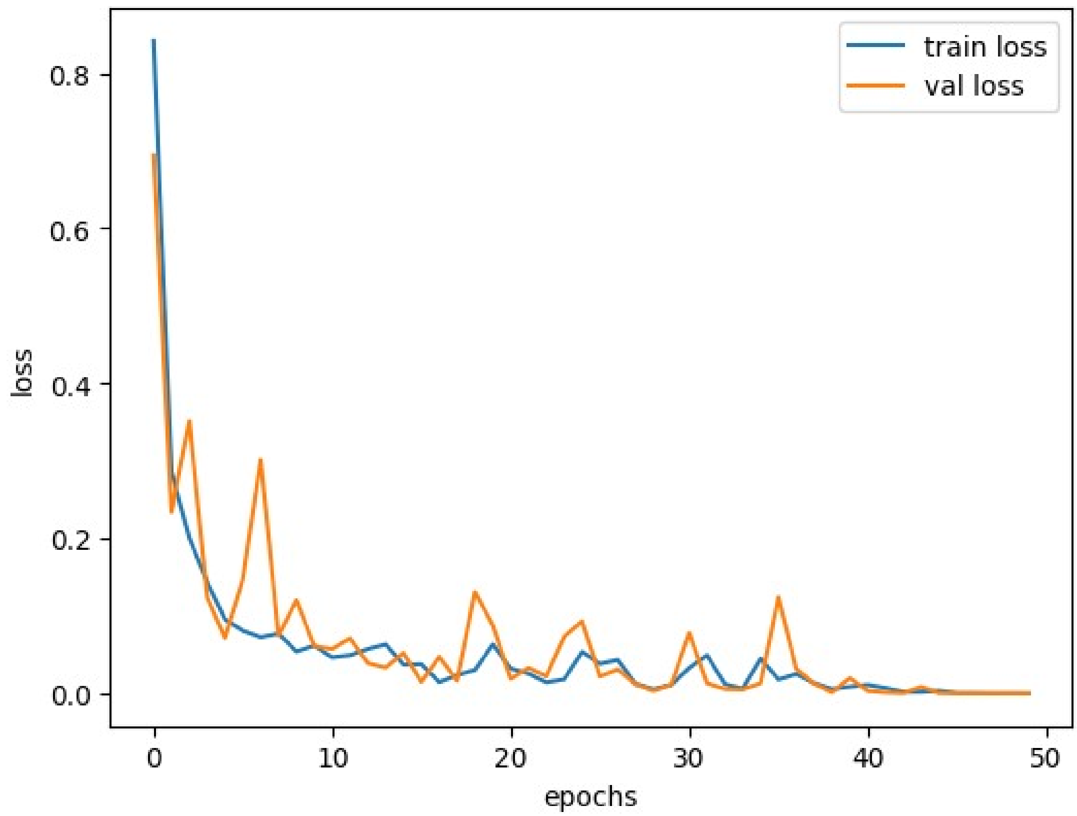
```python
model.save('voice_emotion.keras')
print("Model 'voice_emotion.keras' đã được lưu thành công")
```

```
Model 'voice_emotion.keras' đã được lưu thành công
```

```python
epochs = list(range(50))
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()
```




```
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()
```




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

6. Testing - Đánh giá kết quả dự đoán của Model

Import

```
import os
import numpy as np
import librosa
from keras.models import load_model
```

Kiểm tra đường dẫn đến TestFiles và model voice_emotion.keras

```
import os

# folder_path = '/content/drive/MyDrive/2. Voice Mood Recognition'
# test_folder_path = folder_path + '/TestFiles - Moods'
# model_path = folder_path + '/voice_emotion.keras'
# os.chdir(test_folder_path)
```

```python
test_folder_path = '/kaggle/input/2-voice-mood-recognition/TestFiles - Moods'
model_path = '/kaggle/input/2-voice-mood-recognition/voice_emotion.keras'

print(f"Thư mục test: {test_folder_path}")
print(f"Model path: {model_path}")
print("Các file trong thư mục test:", os.listdir(test_folder_path))

labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'ps', 'sad']
```

```
Thư mục test: /kaggle/input/2-voice-mood-recognition/TestFiles - Moods
Model path: /kaggle/input/2-voice-mood-recognition/voice_emotion.keras
Các file trong thư mục test: ['disgust1.mp3', 'sad1.mp3',
'happy1.mp3', 'fear1.wav', 'happy2.mp3', 'disgusted1.wav',
'disgust2.mp3', 'angry1.mp3', 'fear2.wav', 'neutral1.mp3',
'angry2.mp3', 'disgusted2.wav', 'sad2.wav']
```

```python
# Copy mfcc từ training
def extract_features(filename):
    try:
        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)
    except Exception as e:
        print(f"Lỗi khi xử lý file {filename}: {e}")
        return None
    return mfcc

try:
  model = load_model(model_path)
  print("Tải mô hình thành công!")
except Exception as e:
  print(f"Lỗi khi tải model: {e}")
```

```
Tải mô hình thành công!
```

```python
# sắp xếp lại để in ra kết quả theo thứ tự abc
test_folder = os.listdir(test_folder_path)
test_folder.sort()

if 'model' in locals():
    print("\n--- BẮT ĐẦU DỰ ĐOÁN ---")

    for file_name in test_folder:
        if file_name.endswith(('.wav', '.mp3')):
            full_path = os.path.join(test_folder_path, file_name)

            # Trích xuất đặc trưng
            features = extract_features(full_path)
```

```python
            if features is not None:
                # Reshape dữ liệu để phù hợp với đầu vào của model
                features = np.expand_dims(features, axis=0)
                features = np.expand_dims(features, axis=-1)

                # Dự đoán
                prediction = model.predict(features)
                predicted_index = np.argmax(prediction, axis=1)[0]
                predicted_emotion = labels[predicted_index]

                print(f"File: {file_name} -> Dự đoán:
{predicted_emotion}")
                print("\n")
    print("--- KẾT THÚC DỰ ĐOÁN ---")
```

```
--- BẮT ĐẦU DỰ ĐOÁN ---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
File: angry1.mp3 -> Dự đoán: disgust


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: angry2.mp3 -> Dự đoán: disgust


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
File: disgust1.mp3 -> Dự đoán: fear


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
File: disgust2.mp3 -> Dự đoán: happy


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: disgusted1.wav -> Dự đoán: happy


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: disgusted2.wav -> Dự đoán: disgust


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: fear1.wav -> Dự đoán: happy


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: fear2.wav -> Dự đoán: fear


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: happy1.mp3 -> Dự đoán: disgust
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
File: happy2.mp3 -> Dự đoán: fear

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
File: neutral1.mp3 -> Dự đoán: fear

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
File: sad1.mp3 -> Dự đoán: disgust

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
File: sad2.wav -> Dự đoán: angry

--- KẾT THÚC DỰ ĐOÁN ---
```