Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click

the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks

- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colaboratory
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

Featured examples

- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.

- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
import pandas as pad
import numpy as nup
import glob
import soundfile
import os
import sys


import librosa
import librosa.display
import seaborn as sbn
import matplotlib.pyplot as mplt
from sklearn import metrics


from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import glob
import os
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split


from IPython.display import Audio


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

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

```
Mounted at /content/drive
```

```
DataSet="/content/drive/MyDrive/speech-emotion-recognition-ravdess-data/"
```

```
DataSetDirectoryList=os.listdir(DataSet)
fileEmotion=[]
filePath=[]
for dir in DataSetDirectoryList:
  actor=os.listdir(DataSet+dir)
  for file in actor:
    part=file.split('.')[0]
    part=part.split('-')
    fileEmotion.append(int(part[2]))
    filePath.append(DataSet+dir+'/'+file)
emotion_df=pad.DataFrame(fileEmotion,columns=['Emotions'])
path_df=pad.DataFrame(filePath,columns=['Path'])
DataSet_df=pad.concat([emotion_df,path_df],axis=1)
```

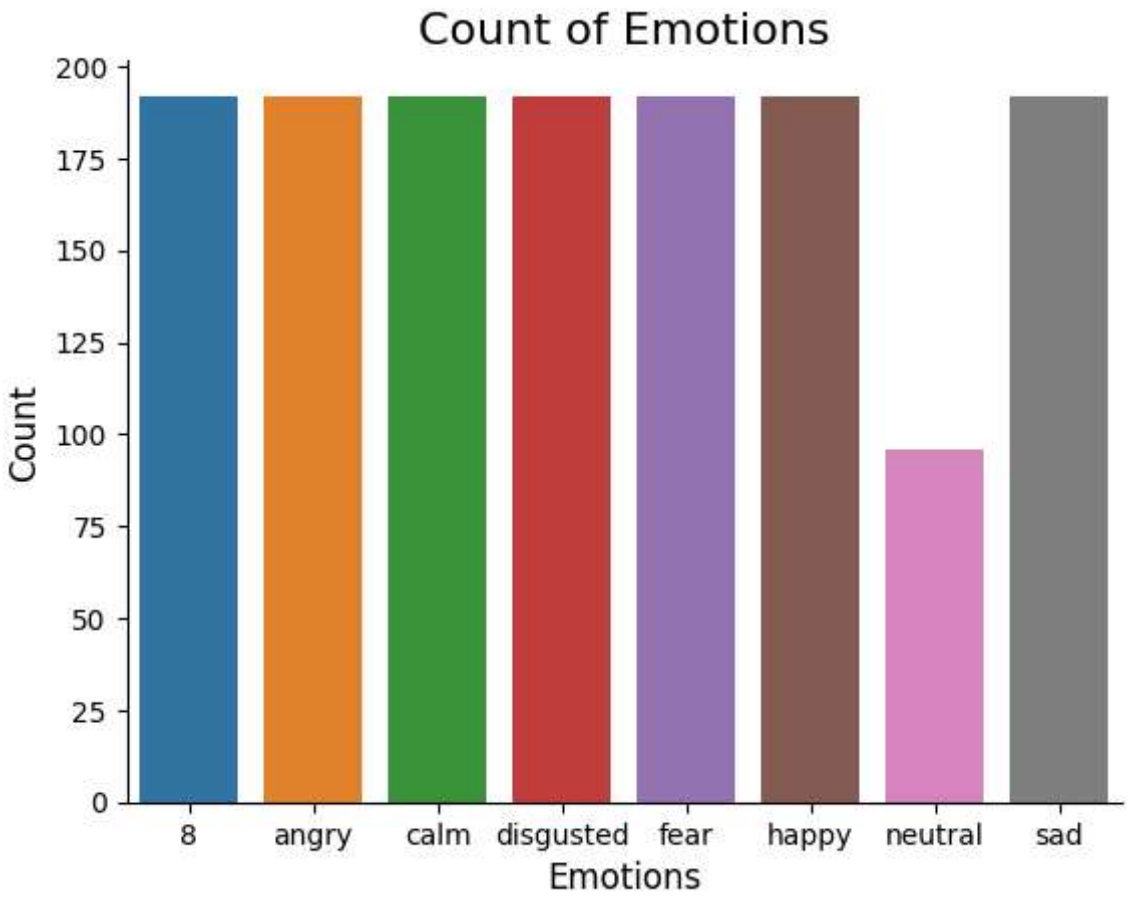
```
DataSet_df.Emotions.replace({1:'neutral',2:'calm',3:'happy',4:'sad',5:'angry',6:'fear',7:'di
DataSet_df.head(6)
```

	Emotions	Path
0	calm	/content/drive/MyDrive/speech-emotion-recognit...
1	neutral	/content/drive/MyDrive/speech-emotion-recognit...
2	neutral	/content/drive/MyDrive/speech-emotion-recognit...
3	neutral	/content/drive/MyDrive/speech-emotion-recognit...
4	calm	/content/drive/MyDrive/speech-emotion-recognit...
5	neutral	/content/drive/MyDrive/speech-emotion-recognit...

```
dataPath=pad.concat([DataSet_df],axis=0)
dataPath.to_csv("data_path.csv",index=False)
dataPath.Emotions.replace({1:'neutral',2:'calm',3:'happy',4:'sad',5:'angry',6:'fear',7:'disg
dataPath.head()
```

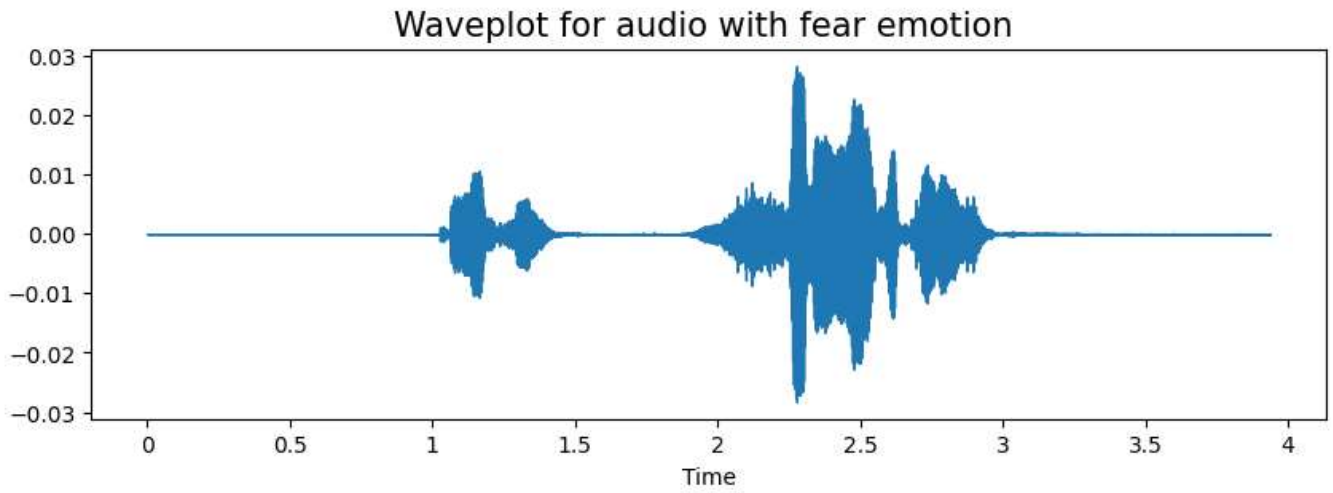
	Emotions	Path
0	calm	/content/drive/MyDrive/speech-emotion-recognit...
1	neutral	/content/drive/MyDrive/speech-emotion-recognit...

```
dataPath['Emotions'] = dataPath['Emotions'].astype('category')


mplt.title('Count of Emotions',size=16)
sbn.countplot(data=dataPath, x='Emotions')
mplt.ylabel('Count',size=12)
mplt.xlabel('Emotions',size=12)
sbn.despine(top=True,right=True,left=False,bottom=False)
mplt.show()
```

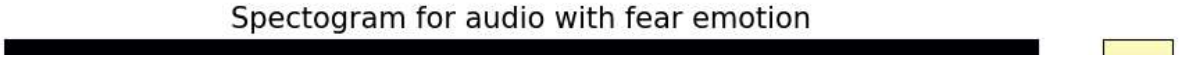



```
import librosa.display


def createWaveplot(data,sr,e):
  mplt.figure(figsize=(10,3))
  mplt.title('Waveplot for audio with {} emotion '.format(e),size=15)
  librosa.display.waveshow(data,sr=sr)
  mplt.show()
```

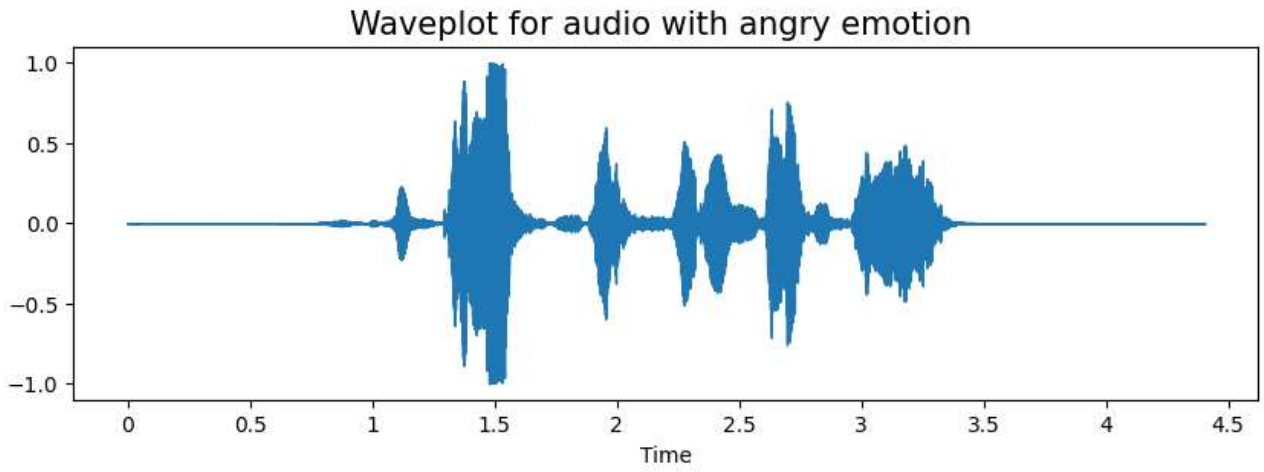
```
def createSpectrogram(data,sr,e):
  X=librosa.stft(data)
  Xdb=librosa.amplitude_to_db(abs(X))
  mplt.figure(figsize=(12,13))
  mplt.title('Spectogram for audio with {} emotion'.format(e),size=15)
  librosa.display.specshow(Xdb,sr=sr,x_axis='time',y_axis='hz')
  mplt.colorbar()



emotion='fear'
path=nup.array(dataPath.Path[dataPath.Emotions==emotion])[1]
data,samplingRate=librosa.load(path)
createWaveplot(data,samplingRate,emotion)
createSpectrogram(data,samplingRate,emotion)
Audio(path)
```



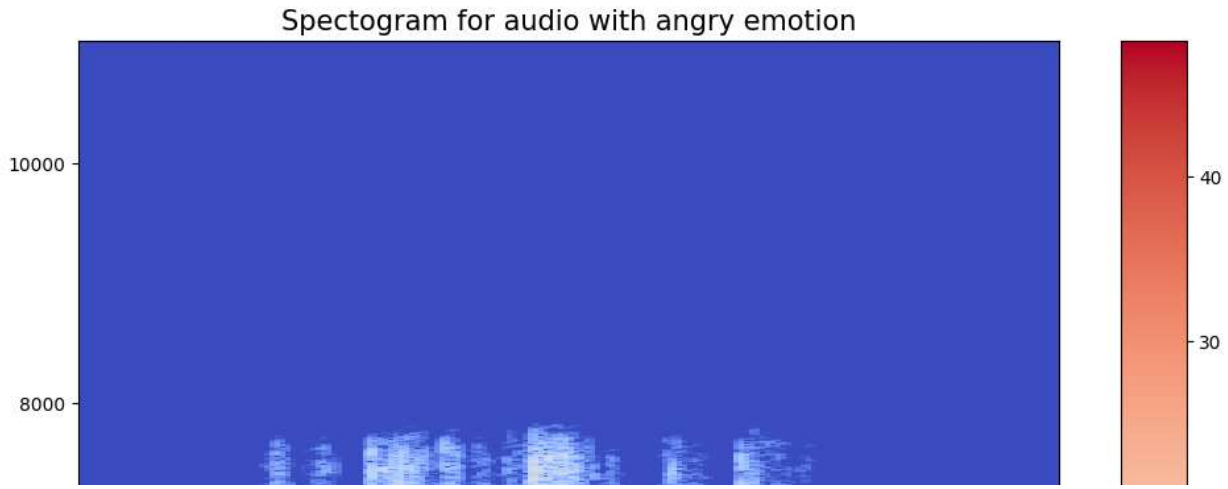
0:00 / 0:03



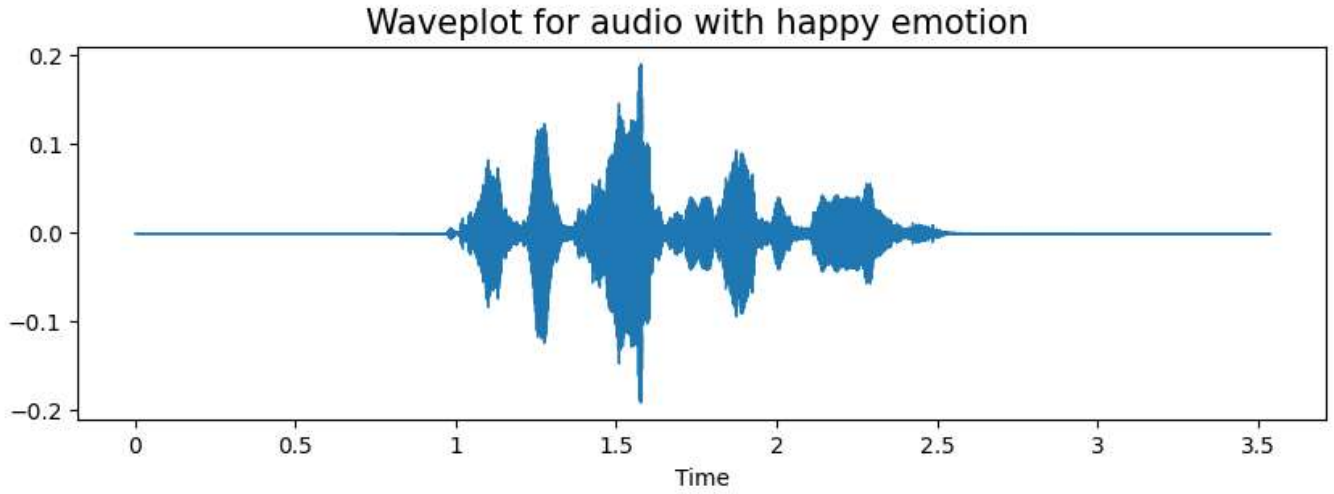

```
emotion='angry'
path=nup.array(dataPath.Path[dataPath.Emotions==emotion])[1]
data,samplingRate=librosa.load(path)
createWaveplot(data,samplingRate,emotion)
createSpectrogram(data,samplingRate,emotion)
Audio(path)
```



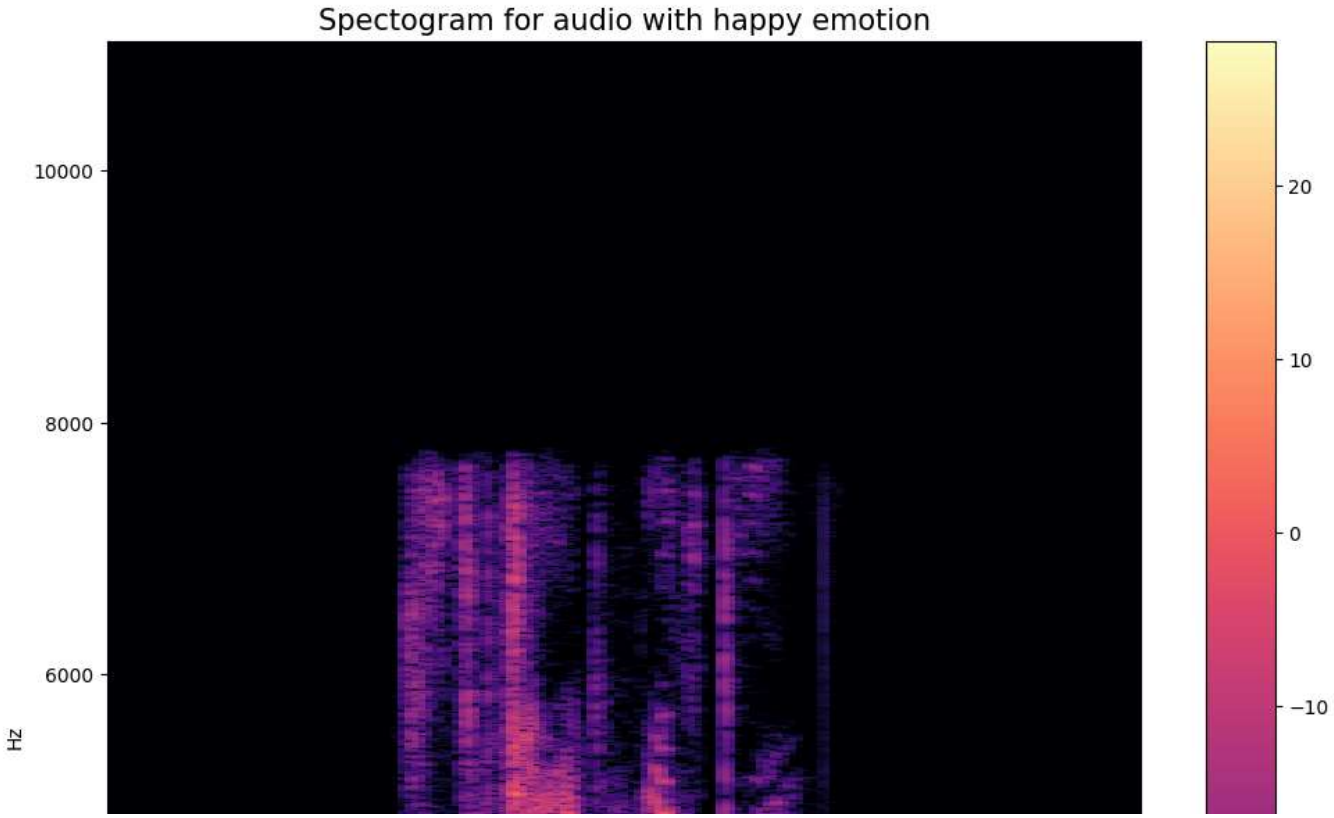
0:00 / 0:04




```
emotion='happy'
path=nup.array(dataPath.Path[dataPath.Emotions==emotion])[1]
data,samplingRate=librosa.load(path)
createWaveplot(data,samplingRate,emotion)
createSpectrogram(data,samplingRate,emotion)
Audio(path)
```



0:00 / 0:03

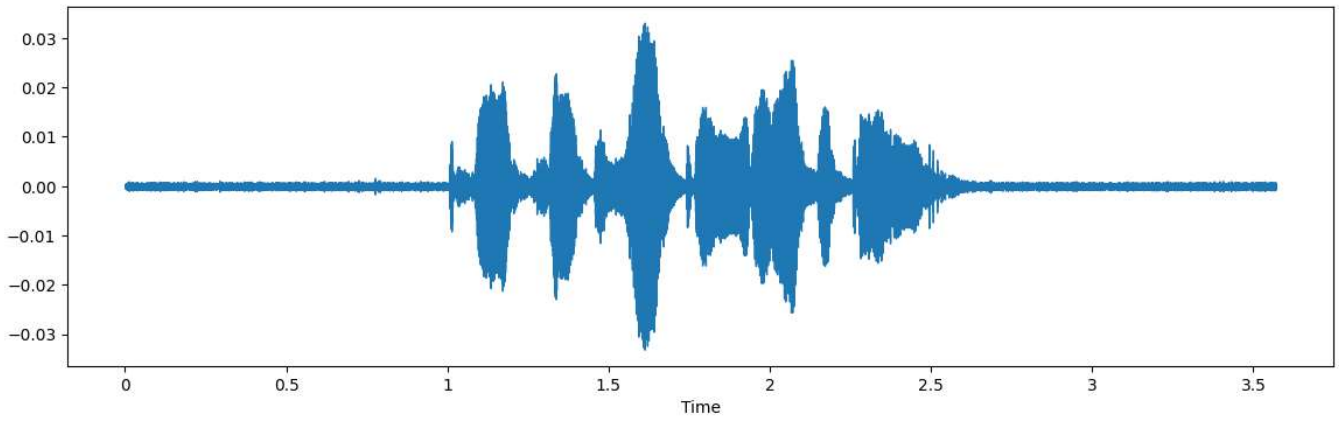



```
def noise(data):
  noiseAmp=0.035*nup.random.uniform()*nup.amax(data)
  data=data+noiseAmp*nup.random.normal(size=data.shape[0])
  return data;
```

```
def stretch(data,rate=0.8):

  return librosa.effects.time_stretch(data,rate=0.8)
```

```
def shift(data):
  shiftRange=int(nup.random.uniform(low=5,high=5)*1000)
  return nup.roll(data,shiftRange)


def pitch(data,samplingRate,pitchFactor=0.7):
  return librosa.effects.pitch_shift(data,sr=samplingRate,n_steps=pitchFactor)


path=nup.array(dataPath.Path)[1]
data,sampleRate=librosa.load(path)



x=noise(data)
mplt.figure(figsize=(14,4))
librosa.display.waveshow(y=x,sr=sampleRate)
Audio(x,rate=sampleRate)
```

0:00 / 0:03

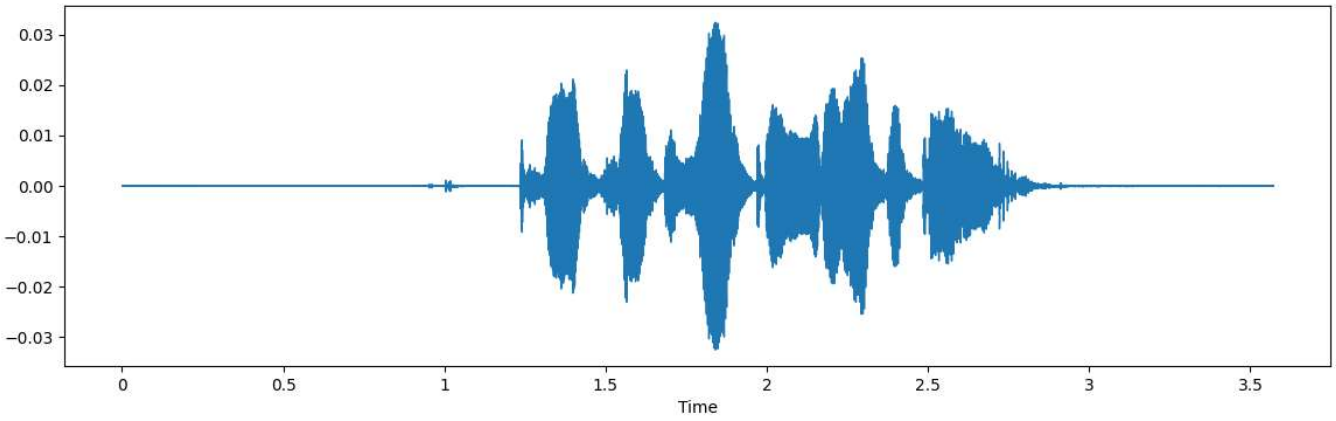



```
x=stretch(data)
mplt.figure(figsize=(14,4))
librosa.display.waveshow(y=x,sr=sampleRate)
Audio(x,rate=sampleRate)
```

0:00 / 0:04

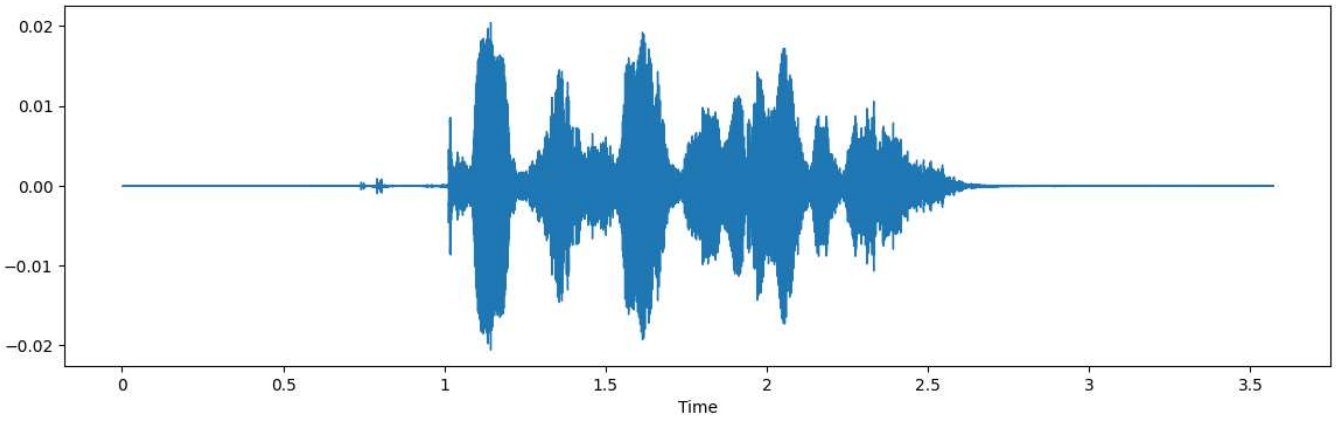
```
x=shift(data)
mplt.figure(figsize=(14,4))
librosa.display.waveshow(y=x,sr=sampleRate)
Audio(x,rate=sampleRate)
```

0:00 / 0:03




```
x = pitch(data, samplingRate, pitchFactor=0.7)
mplt.figure(figsize=(14, 4))
librosa.display.waveshow(y=x, sr=samplingRate)
Audio(x, rate=samplingRate)
```

0:00 / 0:03




```
def extractFeature (fileName,mfcc,chroma,mel):
  with soundfile.SoundFile(fileName) as soundFile:
    X=soundFile.read(dtype='float32')
    sampleRate=soundFile.samplerate
    if chroma:
     stft=nup.abs(librosa.stft(X))

    result=nup.array([])
```

```
    if mfcc:
      mfccs=nup.mean(librosa.feature.mfcc(y=X,sr=sampleRate,n_mfcc=40).T,axis=0)
      result=nup.hstack((result,mfccs))
    if chroma:
      chroma=nup.mean(librosa.feature.chroma_stft(S=stft,sr=sampleRate).T,axis=0)
      result=nup.hstack((result,chroma))
    if mel:
      mel=nup.mean(librosa.feature.melspectrogram(y=X,sr=sampleRate).T,axis=0)
      result=nup.hstack((result,mel))
    return result



emotions={
    '01':'neutral',
    '02':'calm',
    '03':'happy',
    '04':'sad',
    '05':'angry',
    '06':'fearful',
    '07':'disgust',
    '08':'surprised'
}
observedEmotions=['calm','happy','fearful','disgust']


def loadData(test_size=0.2):
  x,y=[],[]
  for file in glob.glob("/content/drive/MyDrive/speech-emotion-recognition-ravdes
    fileName=os.path.basename(file)
    emotion1=emotions[fileName.split("-")[2]]
    if emotion1 not in observedEmotions:
      continue
    feature =extractFeature(file,mfcc=True,chroma=True,mel=True)
    x.append(feature)
    y.append(emotion1)
  return train_test_split(nup.array(x),y,test_size=test_size,random_state=9)


xtrain,xtest,ytrain,ytest=loadData(test_size=0.23)


print((xtrain.shape[0],xtest.shape[0]))


     (591, 177)


print(f'Featires extracted:{xtrain.shape[1]}')

     Featires extracted:180
```

```
model=MLPClassifier(alpha=0.01,batch_size=256,epsilon=1e-08,hidden_layer_sizes=(300,),learni
```

```
model.fit(xtrain,ytrain)
```

```
MLPClassifier(alpha=0.01, batch_size=256, hidden_layer_sizes=(300,),
              learning_rate='adaptive', max_iter=500)
```

```
expected_Of_y=ytest
yPred=model.predict(xtest)
```

```
print(metrics.confusion_matrix(expected_Of_y,yPred))
```

```
[[32 10  3  2]
 [ 2 30  2 10]
 [ 0  2 30  5]
 [ 1  8  7 33]]
```

```
print(classification_report(ytest,yPred))
```

```
                  precision    recall  f1-score   support

            calm       0.91      0.68      0.78        47
         disgust       0.60      0.68      0.64        44
         fearful       0.71      0.81      0.76        37
           happy       0.66      0.67      0.67        49

        accuracy                           0.71       177
       macro avg       0.72      0.71      0.71       177
    weighted avg       0.72      0.71      0.71       177
```

```
accuracy=accuracy_score(y_true=ytest,y_pred=yPred)
```

```
print("Accuracy:{:.2f}%".format(accuracy*100))
```

```
Accuracy:70.62%
```

```
import pandas as pd
import numpy as np
import glob
import os
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

```
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest = scaler.transform(xtest)
```

```
svm_model = SVC(kernel='linear', C=1.0, gamma='auto', random_state=9)
```

```
svm_model.fit(xtrain, ytrain)
```

```
SVC
SVC(gamma='auto', kernel='linear', random_state=9)
```

```
y_pred = svm_model.predict(xtest)
```

```
print("Confusion Matrix:\n", confusion_matrix(ytest, y_pred))
print("\nClassification Report:\n", classification_report(ytest, y_pred))
```

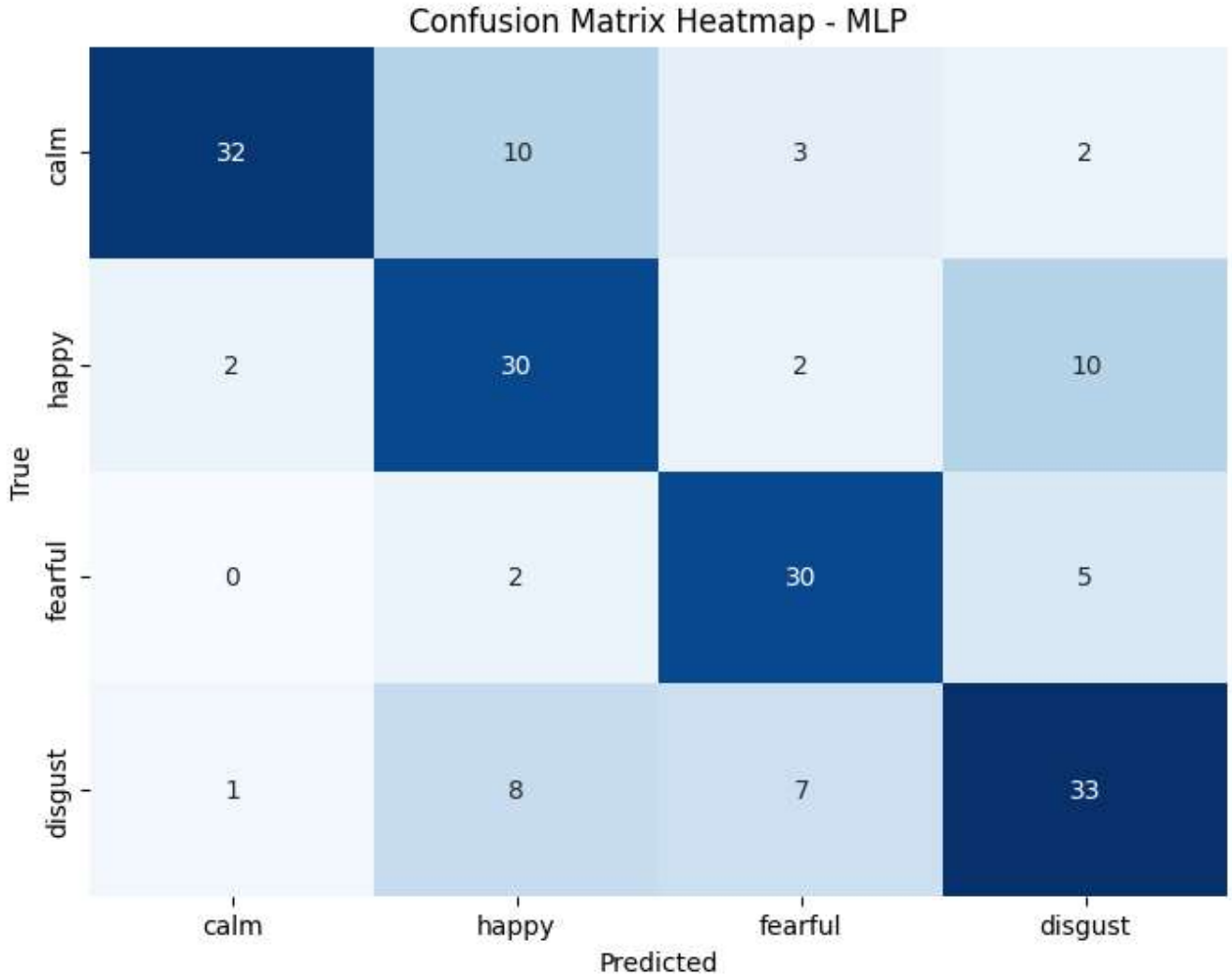
```
Confusion Matrix:
 [[42  4  1  0]
 [ 5 29  6  4]
 [ 3  5 25  4]
 [ 6 13  8 22]]

Classification Report:
              precision    recall  f1-score   support

        calm       0.75      0.89      0.82        47
     disgust       0.57      0.66      0.61        44
     fearful       0.62      0.68      0.65        37
       happy       0.73      0.45      0.56        49

    accuracy                           0.67       177
   macro avg       0.67      0.67      0.66       177
weighted avg       0.67      0.67      0.66       177
```

```
accuracy_svm = metrics.accuracy_score(y_true=ytest, y_pred=y_pred)
print("Accuracy: {:.2f}%".format(accuracy_svm * 100))
```

```
Accuracy: 66.67%
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Function to plot confusion matrix heatmap
def plot_confusion_matrix_heatmap(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=observedEmotions, columns=observedEmotions)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm_df, annot=True, fmt='g', cmap='Blues', cbar=False)
    plt.title(f'Confusion Matrix Heatmap - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# MLP model confusion matrix heatmap
plot_confusion_matrix_heatmap(ytest, yPred, 'MLP')

# SVM model confusion matrix heatmap
plot_confusion_matrix_heatmap(ytest, y_pred, 'SVM')
```




```
from sklearn.neighbors import KNeighborsClassifier
```

```
scaler = StandardScaler()
xtrain_scaled = scaler.fit_transform(xtrain)
xtest_scaled = scaler.transform(xtest)
```

```
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors as
knn_model.fit(xtrain_scaled, ytrain)
```

KNeighborsClassifier

KNeighborsClassifier()

```
y_pred_knn = knn_model.predict(xtest_scaled)
```

```
accuracy_knn = accuracy_score(y_true=ytest, y_pred=y_pred_knn)
```

Predicted

```
print(f"KNN Model Accuracy: {accuracy_knn * 100:.2f}%")
```

```
KNN Model Accuracy: 63.84%
```

```
import matplotlib.pyplot as plt


# Accuracy values
accuracies = [accuracy * 100 for accuracy in [accuracy_knn, accuracy, accuracy_svm]]

# Model names
models = ['KNN', 'MLP', 'SVM']

# Bar plot
plt.figure(figsize=(8, 5))
plt.bar(models, accuracies, color=['blue', 'orange', 'green'])
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies')
plt.ylim([0, 100])
plt.show()
```