In [1]:

```
#           ML Facial recognition to detect mood and suggest songs accordingly
```

In [2]:

```
import numpy as np
import pandas as pd
```

```
C:\Users\tridi\anaconda3\lib\site-packages\pandas\core\computation\expressions.py:21: UserWarning: Pandas requires vers
ion '2.7.3' or newer of 'numexpr' (version '2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:

```
import cv2
import os
import random
from skimage.io import imread
```

In [5]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, BatchNormalization, Conv2D,MaxPooling2D,Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.models import load_model
```

In [6]:

```
from IPython.display import Audio
```

In [7]:

```
# read dataset

from skimage.io import imread
train_dir = "dataset/train/"
test_dir = "dataset/test/"
```

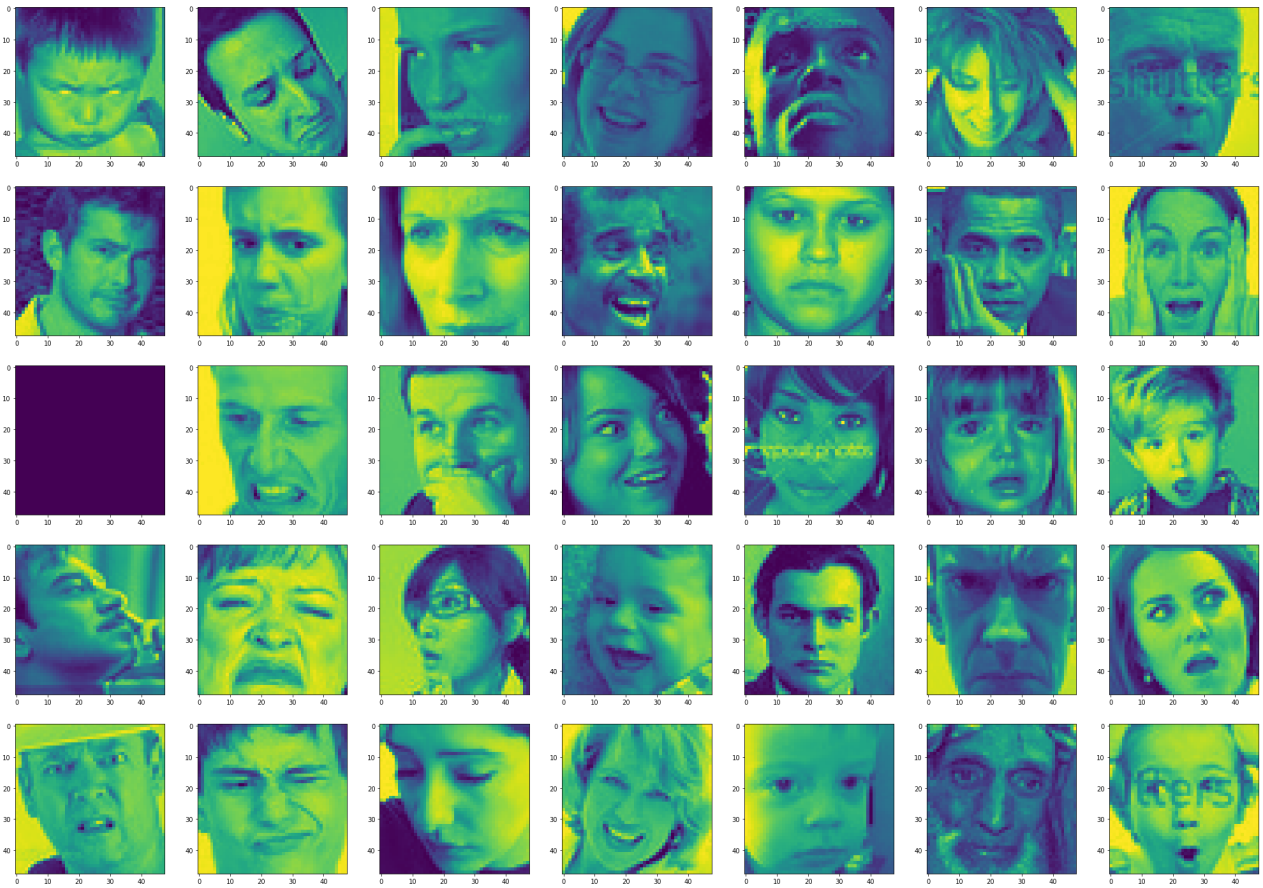
In [8]:

```
total_labels = len(os.listdir(train_dir))
```

In [9]:

```
#data visulizatibn
fig, ax = plt.subplots(nrows=5, ncols=total_labels, figsize=(35, 25))
for x in range(5):
    for y,v in zip(range(total_labels),os.listdir(train_dir)):
        ax[x][y].imshow(imread(train_dir+v+'/'+os.listdir(train_dir+v)[x]))

plt.show()
```



In [10]:

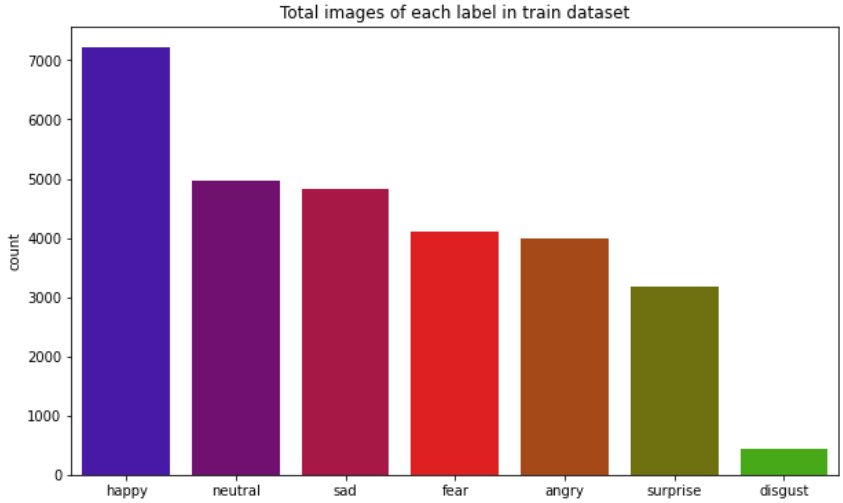
```
#gray images for cnn
fig, ax = plt.subplots(nrows=5, ncols=total_labels, figsize=(35, 25))
for x in range(5):
    for y,v in zip(range(total_labels),os.listdir(train_dir)):
        ax[x][y].imshow(imread(train_dir+v+'/'+os.listdir(train_dir+v)[x]), cmap='gray')

plt.show()
```

In [11]:

```
df = {}
for i in os.listdir(train_dir):
    directory = train_dir + i
    df[i] = len(os.listdir(directory))
df = pd.DataFrame(df, index=["total"]).transpose().sort_values("total", ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x=df.index, y="total", palette="brg", data=df)
plt.ylabel("count")
plt.title("Total images of each label in train dataset")
plt.show()
```



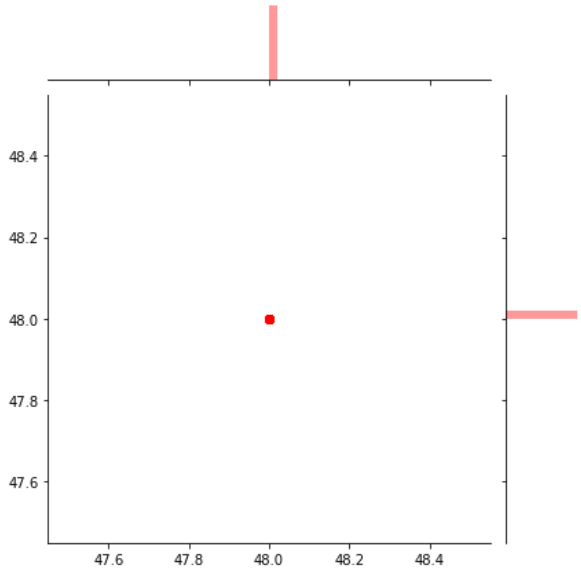
In [12]:

```
happy = os.listdir(train_dir+'happy/')
dim1, dim2 = [], []

for img_filename in happy:
    img = imread(train_dir+'happy/'+img_filename)
    d1, d2 = img.shape
    dim1.append(d1)
    dim2.append(d2)
```

In [13]:

```
img_shape = (int(np.mean(dim1)), int(np.mean(dim2)), 1)
sns.jointplot(dim1, dim2,color='red')
plt.show()
```



In [14]:

```
#Data Preprocessing
```

In [15]:

```
#to gnerate batches of tensor images

train_gen = ImageDataGenerator(rescale=1/255,
                               rotation_range=40,
                               width_shift_range=0.2,
                               height_shift_range=0.2,
                               shear_range=0.2,
                               zoom_range=0.2,
                               horizontal_flip=True,
                               fill_mode='nearest')

test_gen = ImageDataGenerator(rescale=1/255)
```

In [16]:

```
img_shape = (int(np.mean(dim1)), int(np.mean(dim2)), 1)
```

In [17]:

```
#splitting into train& test data

train_generator = train_gen.flow_from_directory(directory=train_dir,
                                                target_size=(img_shape[0], img_shape[1]),
                                                color_mode='grayscale',
                                                batch_size=64,
                                                class_mode='categorical',
                                                shuffle=True)

test_generator = test_gen.flow_from_directory(directory=test_dir,
                                                target_size=(img_shape[0], img_shape[1]),
                                                color_mode='grayscale',
                                                batch_size=64,
                                                class_mode='categorical',
                                                shuffle=False)
```

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

In [18]:

```
# Creating the Model
```

In [19]:

```
model=Sequential()
```

In [20]:

```
# with 64 filter
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', input_shape=img_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
```

In [21]:

```
#with 128 filters
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
```

In [22]:

```
# with 512
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
```

In [23]:

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
```

In [24]:

```
model.add(Flatten())
```

In [25]:

```
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2))
```

In [26]:

```
model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.2))
```

In [27]:

```
model.add(Dense(units=len(os.listdir(train_dir)),activation='softmax'))
```

In [28]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 64)        640

 batch_normalization (BatchN  (None, 48, 48, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 24, 24, 64)       0
 )

 dropout (Dropout)           (None, 24, 24, 64)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 128)       73856

 batch_normalization_1 (Batc  (None, 24, 24, 128)      512
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 128)      0
 2D)

 dropout_1 (Dropout)         (None, 12, 12, 128)       0

 conv2d_2 (Conv2D)           (None, 12, 12, 512)       590336

 batch_normalization_2 (Batc  (None, 12, 12, 512)      2048
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 512)        0
 2D)

 dropout_2 (Dropout)         (None, 6, 6, 512)         0

 conv2d_3 (Conv2D)           (None, 6, 6, 512)         2359808

 batch_normalization_3 (Batc  (None, 6, 6, 512)        2048
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 512)        0
 2D)

 dropout_3 (Dropout)         (None, 3, 3, 512)         0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 512)               2359808

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 1024)              525312

 dropout_5 (Dropout)         (None, 1024)              0

 dense_2 (Dense)             (None, 7)                 7175

=================================================================
Total params: 5,921,799
Trainable params: 5,919,367
Non-trainable params: 2,432
_________________________________________________________________
```

In [29]:

```
# Train the model
```

In [30]:

```
model.compile(optimizer=Adam(learning_rate=0.0001,
                             decay=1e-6),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [31]:

```
steps_per_epoch = train_generator.n // train_generator.batch_size
validation_steps = test_generator.n // test_generator.batch_size
num_epochs = 13

history = model.fit(train_generator,
                    epochs=num_epochs,
                    verbose=1,
                    #callbacks=callbacks,
                    validation_data=test_generator,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps)
```

```
Epoch 1/13
448/448 [==============================] - 921s 2s/step - loss: 1.8810 - accuracy: 0.2267 - val_loss: 2.3236 - val_accu
racy: 0.1731
Epoch 2/13
448/448 [==============================] - 779s 2s/step - loss: 1.8036 - accuracy: 0.2445 - val_loss: 1.7933 - val_accu
racy: 0.2662
Epoch 3/13
448/448 [==============================] - 797s 2s/step - loss: 1.7865 - accuracy: 0.2563 - val_loss: 1.7478 - val_accu
racy: 0.2857
Epoch 4/13
448/448 [==============================] - 830s 2s/step - loss: 1.7730 - accuracy: 0.2681 - val_loss: 1.7122 - val_accu
racy: 0.3086
Epoch 5/13
448/448 [==============================] - 649s 1s/step - loss: 1.7537 - accuracy: 0.2805 - val_loss: 1.7661 - val_accu
racy: 0.3104
Epoch 6/13
448/448 [==============================] - 616s 1s/step - loss: 1.7314 - accuracy: 0.2934 - val_loss: 1.7054 - val_accu
racy: 0.3340
Epoch 7/13
448/448 [==============================] - 778s 2s/step - loss: 1.7114 - accuracy: 0.3016 - val_loss: 1.7847 - val_accu
racy: 0.3165
Epoch 8/13
448/448 [==============================] - 855s 2s/step - loss: 1.6775 - accuracy: 0.3259 - val_loss: 1.5881 - val_accu
racy: 0.3850
Epoch 9/13
448/448 [==============================] - 549s 1s/step - loss: 1.6370 - accuracy: 0.3444 - val_loss: 1.7637 - val_accu
racy: 0.3676
Epoch 10/13
448/448 [==============================] - 552s 1s/step - loss: 1.6062 - accuracy: 0.3642 - val_loss: 1.4751 - val_accu
racy: 0.4067
Epoch 11/13
448/448 [==============================] - 589s 1s/step - loss: 1.5656 - accuracy: 0.3857 - val_loss: 1.4236 - val_accu
racy: 0.4634
Epoch 12/13
448/448 [==============================] - 589s 1s/step - loss: 1.5330 - accuracy: 0.3983 - val_loss: 1.3380 - val_accu
racy: 0.4895
Epoch 13/13
448/448 [==============================] - 563s 1s/step - loss: 1.5031 - accuracy: 0.4165 - val_loss: 1.3777 - val_accu
racy: 0.4756
```

In [32]:

```
model.save('model.h5')
```

In [33]:

```
# Evaluate the model
test_loss, test_acc = model.evaluate(test_generator)
print("validation accuracy :", str(test_acc*100)+"%")
print("validation loss :", test_loss)
```

```
113/113 [==============================] - 30s 268ms/step - loss: 1.3765 - accuracy: 0.4762
validation accuracy : 47.61772155761719%
validation loss : 1.3765020370483398
```

In [34]:

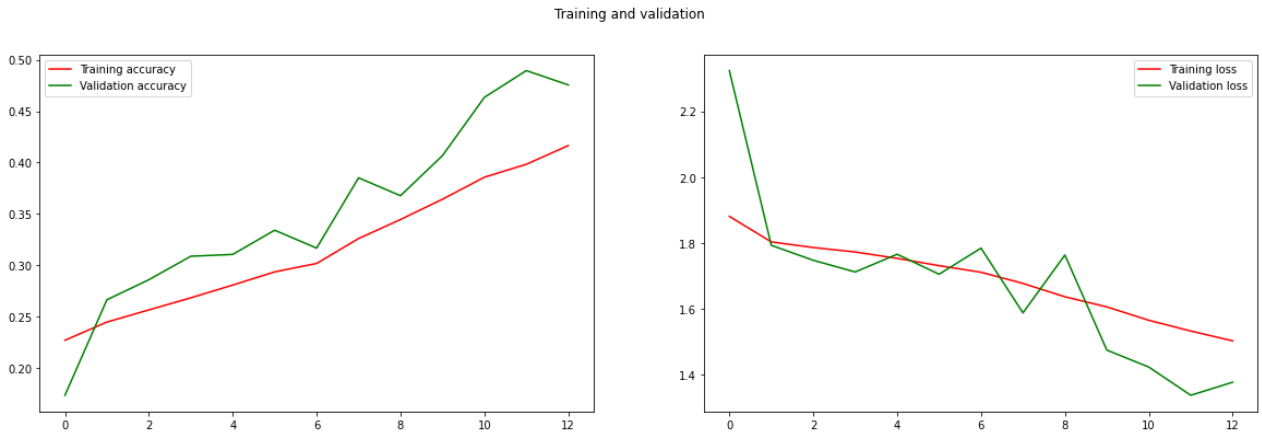
```
# Plotting Training and Validation plot
```

In [36]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))
ax[0].plot(epochs, acc, 'red', label='Training accuracy')
ax[0].plot(epochs, val_acc, 'green', label='Validation accuracy')
ax[0].legend(loc=0)
ax[1].plot(epochs, loss, 'red', label='Training loss')
ax[1].plot(epochs, val_loss, 'green', label='Validation loss')
ax[1].legend(loc=0)

plt.suptitle('Training and validation')
plt.show()
```



In [41]:

```
# Testing our model with new image
```

In [ ]:

```
#1
```

In [38]:

```
image = cv2.imread("images/sad.jpg")
from IPython.display import Image
Image(filename='images/sad.jpg')
```

Out[38]:

In [40]:

```
# Model Prediction 1

model = load_model("model.h5")
# A list of emoticon categories
EMOTIONS = ['Angry', 'Disgust', 'Happy', 'Sad', 'Surprise', 'Neutral']
# Load image
img = image

# Trim the image to 48 x 48, and turn the grayscale image, normalization
frame = cv2.resize(img,(48,48),interpolation=cv2.INTER_BITS2)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) / 255.0

# Reinvent the image dimension
gray = gray.reshape(1,48,48,1)

# Output the prediction
predicts = model.predict(gray)[0]
label = EMOTIONS[predicts.argmax()]
for (i,j) in zip(range(7),EMOTIONS):
    predictss = predicts[i]
    print("{:^10s}".format(j)+"prediction rate is   {0:.2f}%".format(predictss))
print( "\n\n The system considers this expression to be:",label)
```

```
1/1 [==============================] - 0s 141ms/step
  Angry   prediction rate is   0.22%
 Disgust  prediction rate is   0.02%
  Happy   prediction rate is   0.11%
   Sad    prediction rate is   0.32%
 Surprise prediction rate is   0.10%
 Neutral  prediction rate is   0.18%


 The system considers this expression to be: Sad
```

In [43]:

```
# Song Recommdation 1


if (label=='Angry'):
    path="song\\Angry\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Angry\\'+ d,autoplay=True)
    display(audio)

elif (label=='Disgust'):
    path="song\\Disgust\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Disgust\\'+ d,autoplay=True)
    display(audio)

elif (label=="Happy"):
    path="song\\Happy\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Happy\\'+ d,autoplay=True)
    display(audio)

elif (label=='Sad'):
    path="song\\Sad\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Sad\\'+ d,autoplay=True)
    display(audio)

elif (label=='Surprise'):
    path="song\\Surprise\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Surprise\\'+ d,autoplay=True)
    display(audio)

elif (label=='Neutral'):
    path="song\\Neutral\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Neutral\\'+ d,autoplay=True)
    display(audio)
```

```
Now Playing: Srotoshini - Encore.mp3
```

0:17 / 4:01

In [44]:

```
#2
```

In [48]:

```
image = cv2.imread("images/angry.jpeg")
from IPython.display import Image
Image(filename='images/angry.jpeg')
```

Out[48]:

In [49]:

```
# Model Prediction 2

model = load_model("model.h5")
# A list of emoticon categories
EMOTIONS = ['Angry', 'Disgust', 'Happy', 'Sad', 'Surprise', 'Neutral']
# Load image
img = image

# Trim the image to 48 x 48, and turn the grayscale image, normalization
frame = cv2.resize(img,(48,48),interpolation=cv2.INTER_BITS2)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) / 255.0

# Reinvent the image dimension
gray = gray.reshape(1,48,48,1)

# Output the prediction
predicts = model.predict(gray)[0]
label = EMOTIONS[predicts.argmax()]
for (i,j) in zip(range(7),EMOTIONS):
    predictss = predicts[i]
    print("{:^10s}".format(j)+"prediction rate is   {0:.2f}%".format(predictss))
print( "\n\n The system considers this expression to be:",label)
```

```
1/1 [==============================] - 0s 109ms/step
  Angry   prediction rate is   0.24%
 Disgust  prediction rate is   0.01%
  Happy   prediction rate is   0.20%
   Sad    prediction rate is   0.16%
 Surprise prediction rate is   0.14%
 Neutral  prediction rate is   0.17%


 The system considers this expression to be: Angry
```

In [50]:

```
# Song Recommdation 2


if (label=='Angry'):
    path="song\\Angry\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Angry\\'+ d,autoplay=True)
    display(audio)

elif (label=='Disgust'):
    path="song\\Disgust\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Disgust\\'+ d,autoplay=True)
    display(audio)

elif (label=="Happy"):
    path="song\\Happy\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Happy\\'+ d,autoplay=True)
    display(audio)

elif (label=='Sad'):
    path="song\\Sad\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Sad\\'+ d,autoplay=True)
    display(audio)

elif (label=='Surprise'):
    path="song\\Surprise\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Surprise\\'+ d,autoplay=True)
    display(audio)

elif (label=='Neutral'):
    path="song\\Neutral\\"
    files=os.listdir(path)
    d=random.choice(files)
    print("Now Playing:",d)
    audio = Audio(filename='song\\Neutral\\'+ d,autoplay=True)
    display(audio)
```

```
Now Playing: Sadda Haq-rockstar.mp3
```

0:10 / 6:48

In [ ]:

```
# Thank You
```