In [1]:
```
import pandas as pd
import numpy as np

import os
import sys

# librosa is a Python library for analyzing audio and music. It can be used to extract the data from the audio files we will see it later.
import librosa
import librosa.display
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# to play the audio files
from IPython.display import Audio

import keras
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint

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

In [2]:
```
# Paths for data.
Ravdess = "C:\\Users\\bella\\OneDrive\\Desktop\\speech-emotion-recognition-ravdess-data\\"
```

In [3]:
```python
Ravdess_directory_list = os.listdir(Ravdess)

file_emotion = []
file_path = []
for dir in Ravdess_directory_list:
    # as their are 20 different actors in our previous directory we need to extract files for each actor.
    actor = os.listdir(Ravdess + dir)
    for file in actor:
        part = file.split('.')[0]
        part = part.split('-')
        # third part in each file represents the emotion associated to that file.
        file_emotion.append(int(part[2]))
        file_path.append(Ravdess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Ravdess_df = pd.concat([emotion_df, path_df], axis=1)

# changing integers to actual emotions.
Ravdess_df.Emotions.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5:'angry', 6:'fear', 7:'disgust', 8:'surprise'}, inplace=True)
Ravdess_df.head()
```

Out[3]:

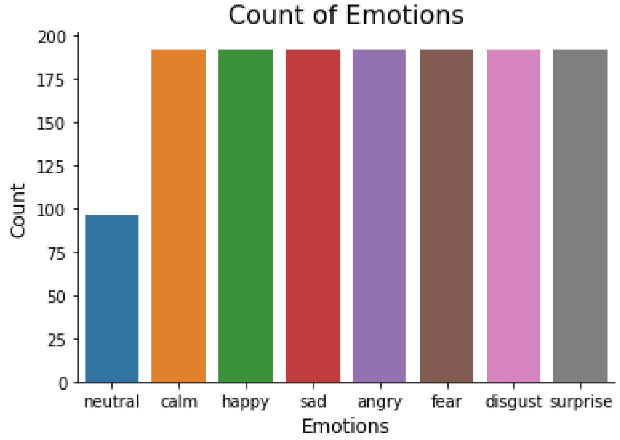
	Emotions	Path
0	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
1	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
2	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
3	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
4	calm	C:\Users\bella\OneDrive\Desktop\speech-emotion...

In [4]:
```python
# creating Dataframe using all the 4 dataframes we created so far.
data_path = pd.concat([Ravdess_df], axis = 0)
data_path.to_csv("data_path.csv",index=False)
data_path.head()
```

Out[4]:

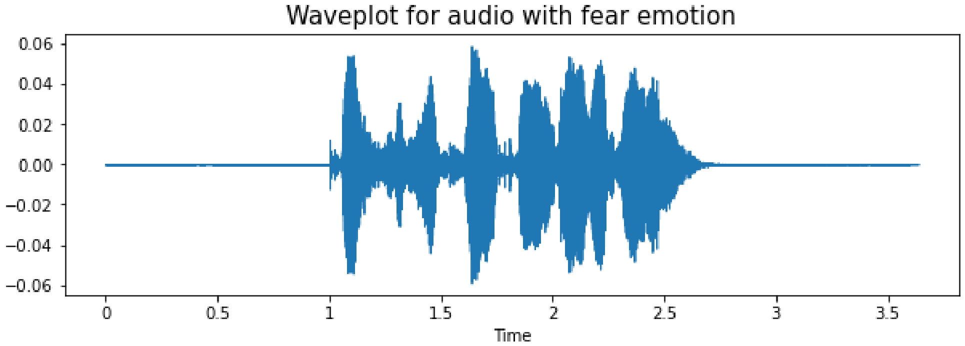
	Emotions	Path
0	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
1	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
2	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
3	neutral	C:\Users\bella\OneDrive\Desktop\speech-emotion...
4	calm	C:\Users\bella\OneDrive\Desktop\speech-emotion...

In [5]:
```python
plt.title('Count of Emotions', size=16)
sns.countplot(data_path.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```



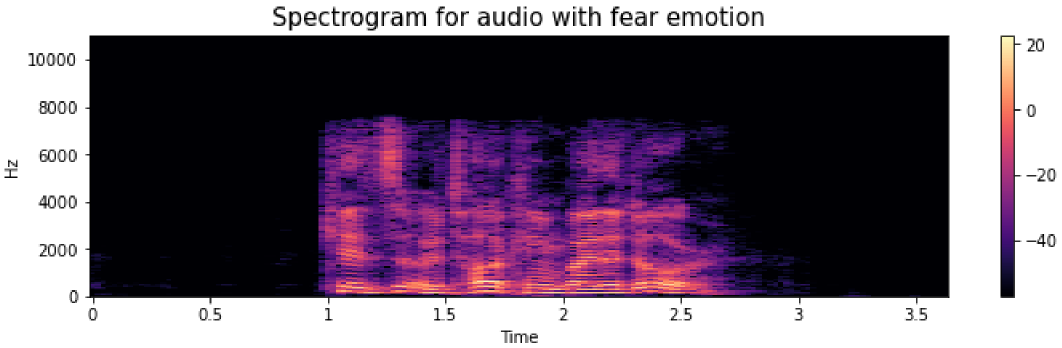
In [6]:
```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveshow(data,sr=sr)
    plt.show()

def create_spectrogram(data, sr, e):
    # stft function converts the data into short term fourier transform
    X = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(12, 3))
    plt.title('Spectrogram for audio with {} emotion'.format(e), size=15)
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
    #librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar()
```

In [7]:
```
emotion='fear'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



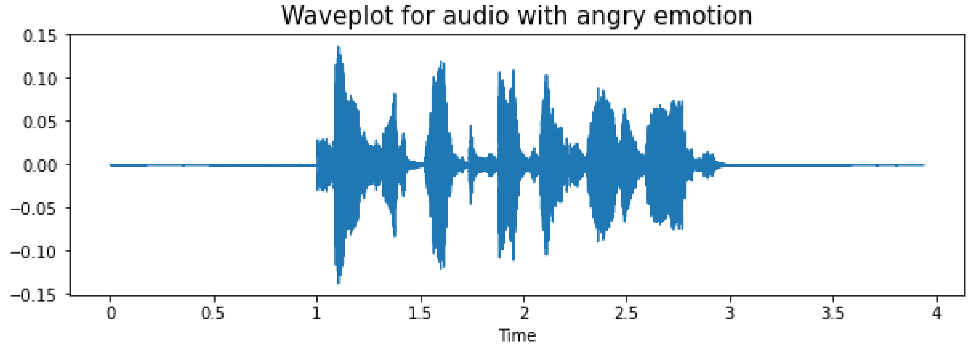
Out[7]:

0:00 / 0:03



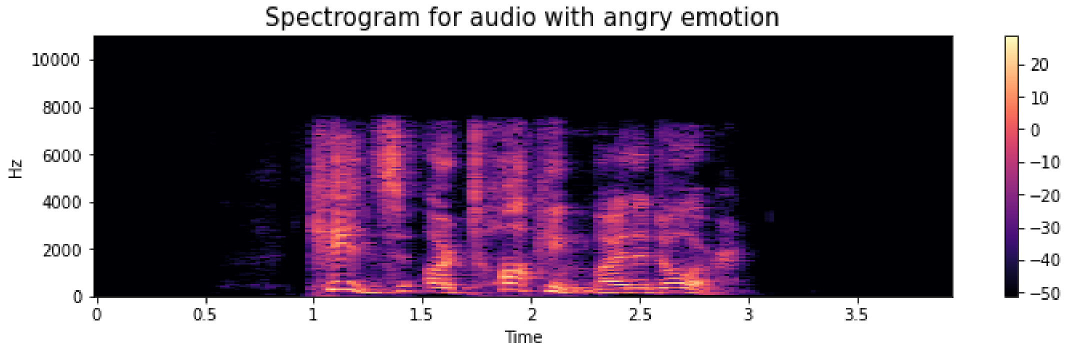
In [8]:

```
emotion='angry'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

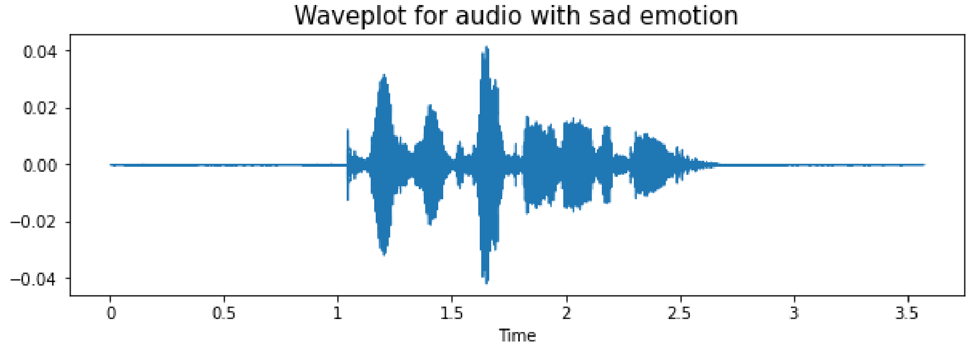


Out[8]:

0:00 / 0:03

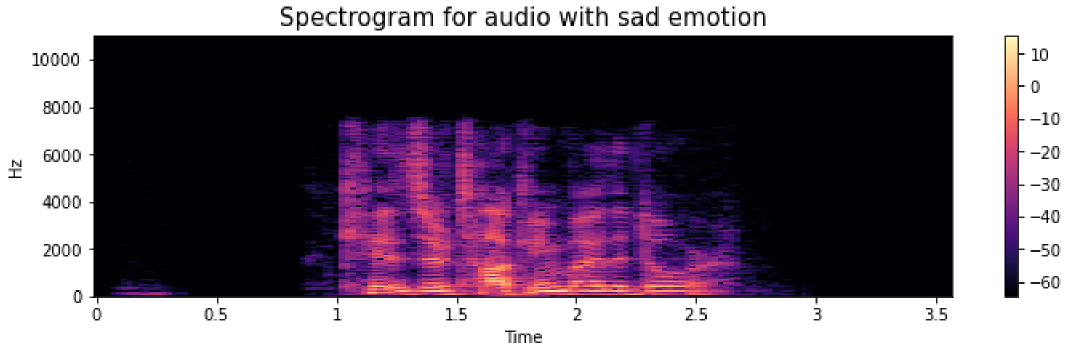


In [9]:
```
emotion='sad'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



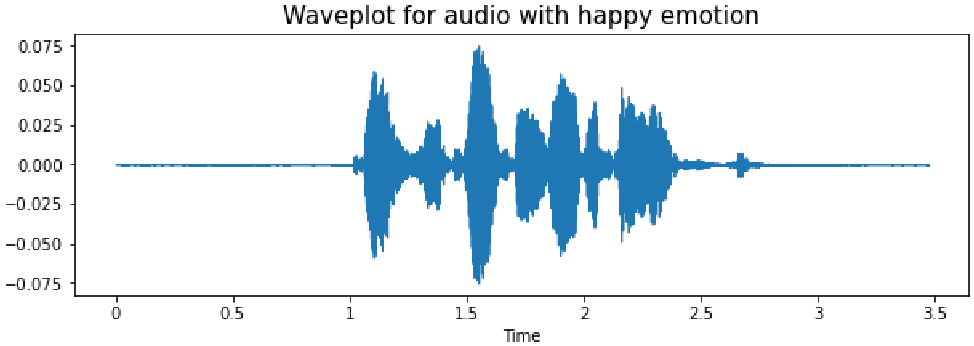
Out[9]:

0:00 / 0:03



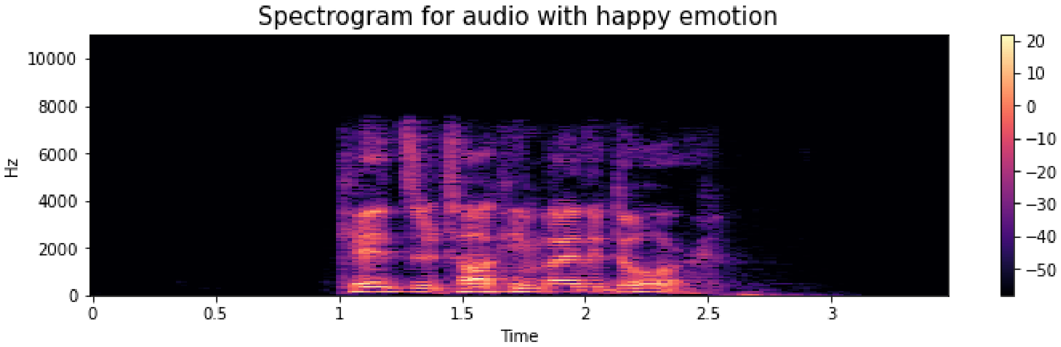
In [10]:

```
emotion='happy'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



Out[10]:

0:00 / 0:03

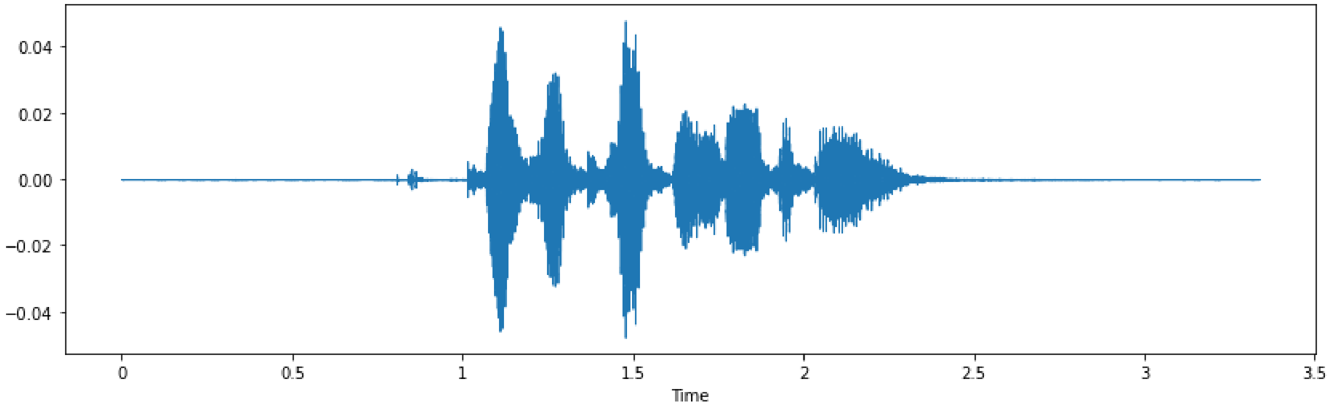


In [11]:
```python
def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate)

def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sampling_rate, pitch_factor)

# taking any example and checking for techniques.
path = np.array(data_path.Path)[1]
data, sample_rate = librosa.load(path)
```

In [12]:
```python
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=data, sr=sample_rate)
Audio(path)
```

Out[12]:

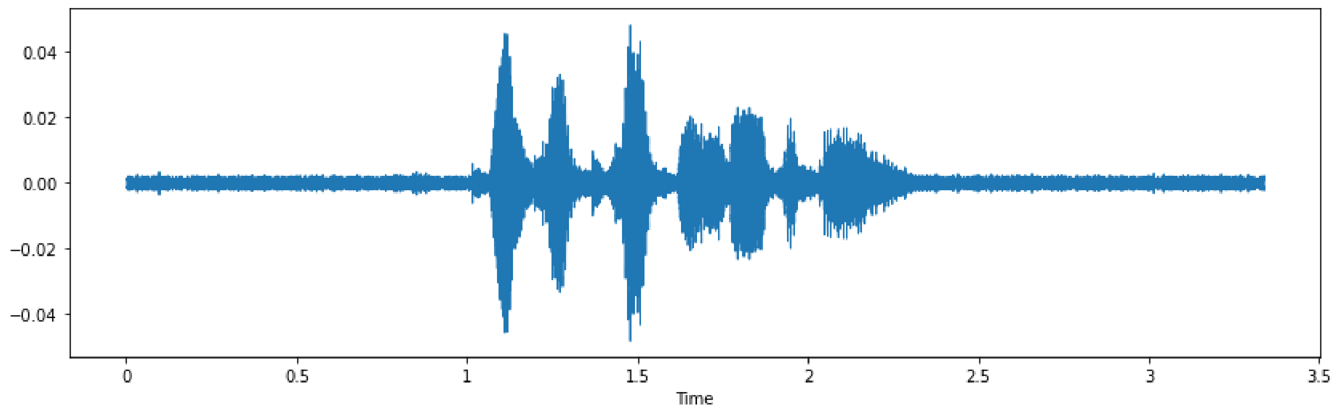
0:00 / 0:03



In [13]:
```
x = noise(data)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[13]:

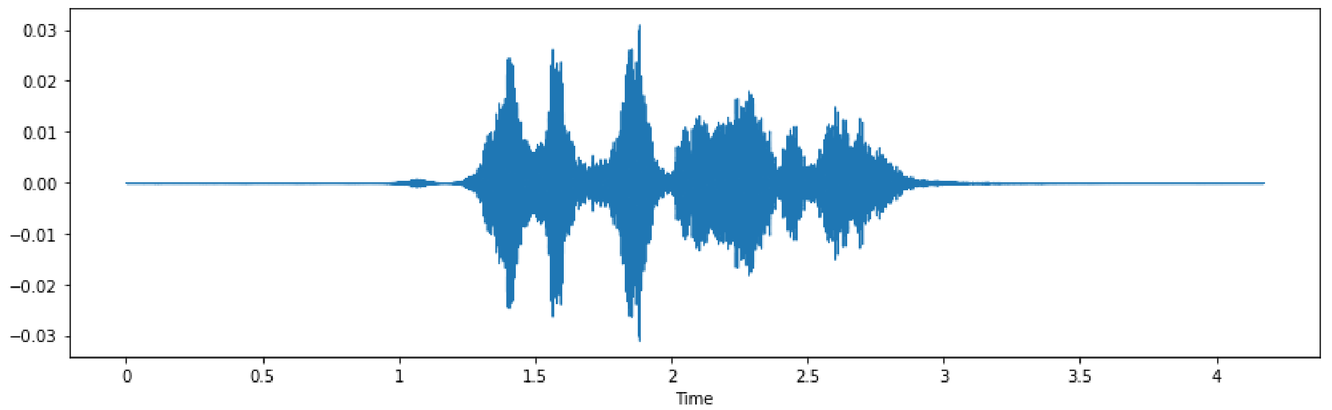
0:00 / 0:03



In [14]:
```
x = stretch(data)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[14]:

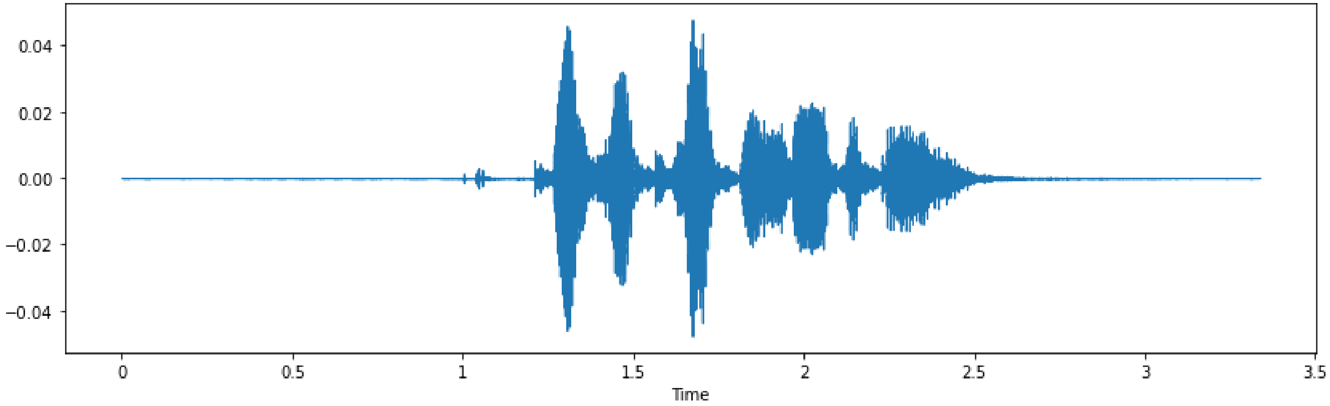
0:00 / 0:04



In [15]:
```
x = shift(data)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[15]:

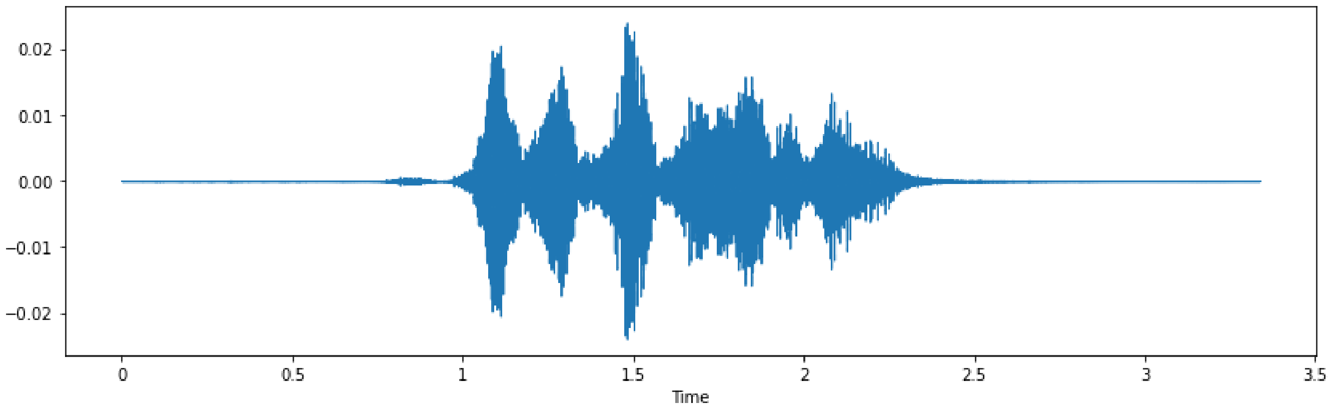
0:00 / 0:03



In [16]:
```
x = pitch(data, sample_rate)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[16]:

0:03 / 0:03



In [17]:
```python
def extract_features(data):
    # ZCR
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result=np.hstack((result, zcr)) # stacking horizontally

    # Chroma_stft
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft)) # stacking horizontally

    # MFCC
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mfcc)) # stacking horizontally

    # Root Mean Square Value
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms)) # stacking horizontally

    # MelSpectogram
    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel)) # stacking horizontally

    return result

def get_features(path):
    # duration and offset are used to take care of the no audio in start and the ending of each audio files as seen above.
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)

    # without augmentation
    res1 = extract_features(data)
    result = np.array(res1)

    # data with noise
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2)) # stacking vertically

    # data with stretching and pitching
    new_data = stretch(data)
    data_stretch_pitch = pitch(new_data, sample_rate)
    res3 = extract_features(data_stretch_pitch)
    result = np.vstack((result, res3)) # stacking vertically

    return result
```

In [18]:
```
X, Y = [], []
for path, emotion in zip(data_path.Path, data_path.Emotions):
    feature = get_features(path)
    for ele in feature:
        X.append(ele)
        # appending emotion 3 times as we have made 3 augmentation techniques on each audio file.
        Y.append(emotion)
```

In [19]:
```
len(X), len(Y), data_path.Path.shape
```

Out[19]:
```
(4320, 4320, (1440,))
```

In [20]:
```
Features = pd.DataFrame(X)
Features['labels'] = Y
Features.to_csv('features.csv', index=False)
Features.head()
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	...	153	154	155	156	157	158	159
0	0.224306	0.664190	0.693037	0.665390	0.673116	0.696442	0.684787	0.706183	0.748844	0.782297	...	6.888786e-17	6.993841e-17	7.601734e-17	7.305700e-17	6.833248e-17	7.254374e-17	7.962738e-17
1	0.307301	0.769032	0.818461	0.806665	0.797195	0.815296	0.749935	0.672598	0.709985	0.753011	...	3.386356e-05	3.487547e-05	3.451716e-05	3.443656e-05	3.228268e-05	3.271177e-05	3.358197e-05
2	0.169383	0.579495	0.662042	0.674849	0.631609	0.623019	0.687452	0.671907	0.692694	0.712601	...	1.885160e-15	1.743947e-15	1.590055e-15	1.477827e-15	1.528373e-15	1.689925e-15	1.679742e-15
3	0.196533	0.652948	0.692924	0.664361	0.648762	0.686783	0.688136	0.683010	0.735986	0.759067	...	7.385022e-17	6.953342e-17	7.445123e-17	7.997667e-17	7.754530e-17	8.110750e-17	7.647288e-17
4	0.338257	0.799537	0.836667	0.822812	0.794836	0.811926	0.724140	0.671130	0.715562	0.752416	...	1.729198e-04	1.675139e-04	1.730260e-04	1.729619e-04	1.737910e-04	1.722833e-04	1.784420e-04

5 rows × 163 columns

In [21]:
```
X = Features.iloc[: ,:-1].values
Y = Features['labels'].values
```

In [22]:
```
# As this is a multiclass classification problem onehotencoding our Y.
encoder = OneHotEncoder()
Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()
```

In [23]:
```
# splitting data
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0, shuffle=True)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[23]:
```
((3240, 162), (3240, 8), (1080, 162), (1080, 8))
```

In [24]:
```
# scaling our data with sklearn's Standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[24]:
```
((3240, 162), (3240, 8), (1080, 162), (1080, 8))
```

In [25]:
```
# making our data compatible to model.
x_train = np.expand_dims(x_train, axis=2)
x_test = np.expand_dims(x_test, axis=2)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[25]:
```
((3240, 162, 1), (3240, 8), (1080, 162, 1), (1080, 8))
```

In [26]:
```
model=Sequential()
model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu', input_shape=(x_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(128, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))
model.add(Dropout(0.2))

model.add(Conv1D(64, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Flatten())
model.add(Dense(units=32, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(units=8, activation='softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])

model.summary()
```

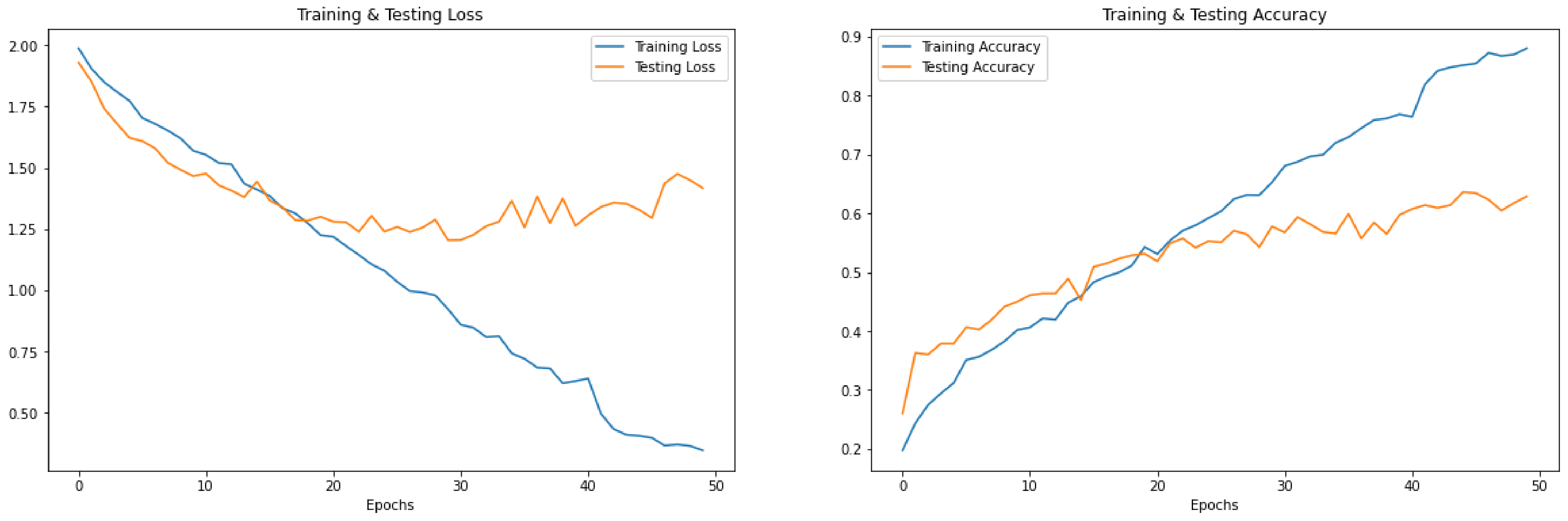
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 162, 256)          1536

 max_pooling1d (MaxPooling1D  (None, 81, 256)          0
 )

 conv1d_1 (Conv1D)           (None, 81, 256)           327936

 max_pooling1d_1 (MaxPooling  (None, 41, 256)          0
 1D)

 conv1d_2 (Conv1D)           (None, 41, 128)           163968

 max_pooling1d_2 (MaxPooling  (None, 21, 128)          0
 1D)

 dropout (Dropout)           (None, 21, 128)           0

 conv1d_3 (Conv1D)           (None, 21, 64)            41024

 max_pooling1d_3 (MaxPooling  (None, 11, 64)           0
 1D)

 flatten (Flatten)           (None, 704)               0

 dense (Dense)               (None, 32)                22560

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 8)                 264

=================================================================
Total params: 557,288
Trainable params: 557,288
Non-trainable params: 0
_________________________________________________________________
```

In [27]:
```python
rlrp = ReduceLROnPlateau(monitor='loss', factor=0.4, verbose=0, patience=2, min_lr=0.0000001)
history=model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), callbacks=[rlrp])
```

```
Epoch 1/50
51/51 [==============================] - 19s 329ms/step - loss: 1.9880 - accuracy: 0.1975 - val_loss: 1.9296 - val_accuracy: 0.2602 - lr: 0.0010
Epoch 2/50
51/51 [==============================] - 16s 320ms/step - loss: 1.9054 - accuracy: 0.2435 - val_loss: 1.8525 - val_accuracy: 0.3630 - lr: 0.0010
Epoch 3/50
51/51 [==============================] - 18s 361ms/step - loss: 1.8496 - accuracy: 0.2747 - val_loss: 1.7420 - val_accuracy: 0.3602 - lr: 0.0010
Epoch 4/50
51/51 [==============================] - 17s 330ms/step - loss: 1.8101 - accuracy: 0.2941 - val_loss: 1.6805 - val_accuracy: 0.3787 - lr: 0.0010
Epoch 5/50
51/51 [==============================] - 16s 321ms/step - loss: 1.7731 - accuracy: 0.3117 - val_loss: 1.6219 - val_accuracy: 0.3787 - lr: 0.0010
Epoch 6/50
51/51 [==============================] - 17s 325ms/step - loss: 1.7032 - accuracy: 0.3512 - val_loss: 1.6086 - val_accuracy: 0.4065 - lr: 0.0010
Epoch 7/50
51/51 [==============================] - 17s 325ms/step - loss: 1.6794 - accuracy: 0.3565 - val_loss: 1.5794 - val_accuracy: 0.4028 - lr: 0.0010
Epoch 8/50
51/51 [==============================] - 16s 321ms/step - loss: 1.6519 - accuracy: 0.3682 - val_loss: 1.5204 - val_accuracy: 0.4194 - lr: 0.0010
Epoch 9/50
51/51 [==============================] - 16s 322ms/step - loss: 1.6204 - accuracy: 0.3827 - val_loss: 1.4920 - val_accuracy: 0.4417 - lr: 0.0010
Epoch 10/50
```

In [28]:
```python
print("Accuracy of our model on test data : " , model.evaluate(x_test,y_test)[1]*100 , "%")

epochs = [i for i in range(50)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
test_acc = history.history['val_accuracy']
test_loss = history.history['val_loss']

fig.set_size_inches(20,6)
ax[0].plot(epochs , train_loss , label = 'Training Loss')
ax[0].plot(epochs , test_loss , label = 'Testing Loss')
ax[0].set_title('Training & Testing Loss')
ax[0].legend()
ax[0].set_xlabel("Epochs")

ax[1].plot(epochs , train_acc , label = 'Training Accuracy')
ax[1].plot(epochs , test_acc , label = 'Testing Accuracy')
ax[1].set_title('Training & Testing Accuracy')
ax[1].legend()
ax[1].set_xlabel("Epochs")
plt.show()
```

```
34/34 [==============================] - 1s 34ms/step - loss: 1.4168 - accuracy: 0.6287
Accuracy of our model on test data :  62.87037134170532 %
```



In [29]:
```
# predicting on test data.
pred_test = model.predict(x_test)
y_pred = encoder.inverse_transform(pred_test)

y_test = encoder.inverse_transform(y_test)
```

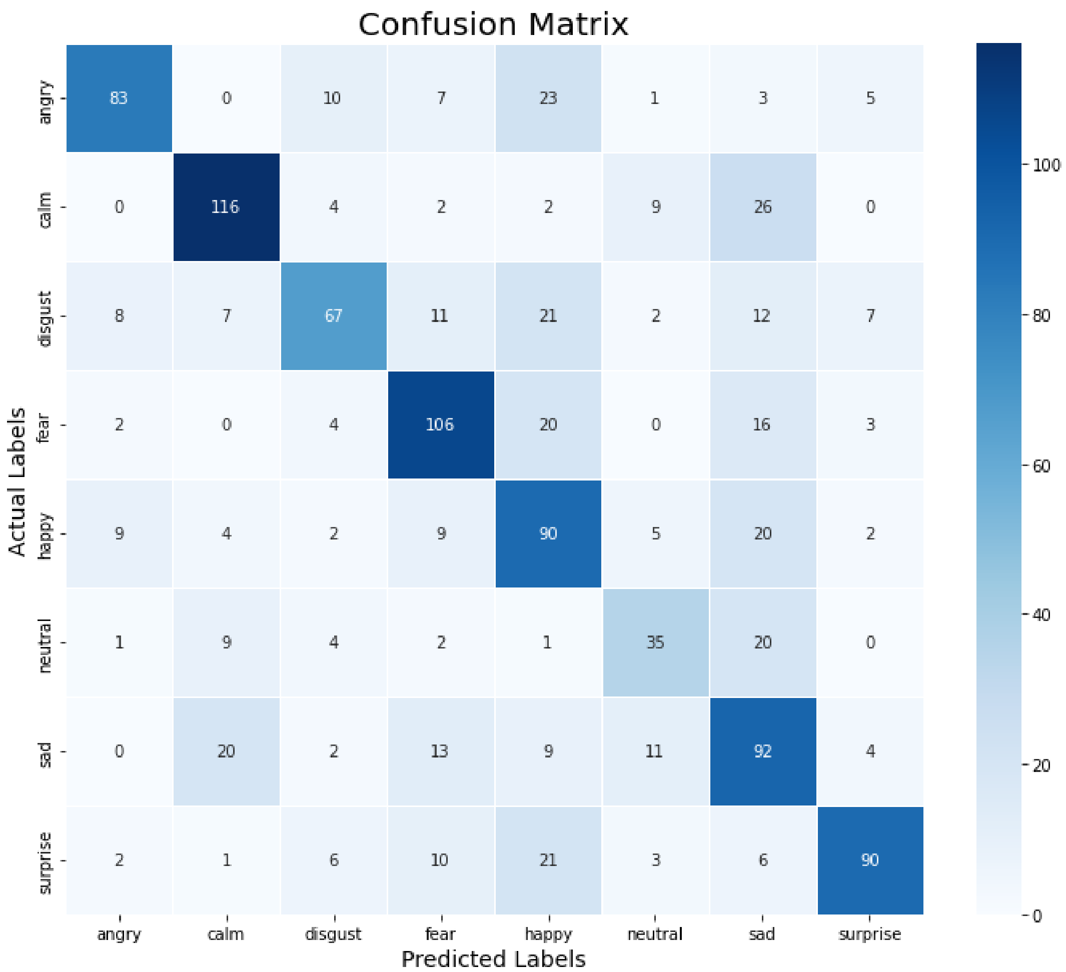
```
34/34 [==============================] - 1s 34ms/step
```

In [30]:
```
df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df['Predicted Labels'] = y_pred.flatten()
df['Actual Labels'] = y_test.flatten()

df.head(10)
```

Out[30]:

	Predicted Labels	Actual Labels
0	fear	fear
1	disgust	angry
2	happy	fear
3	neutral	calm
4	happy	angry
5	surprise	surprise
6	sad	fear
7	happy	happy
8	fear	fear
9	sad	sad

In [31]:
```python
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (12, 10))
cm = pd.DataFrame(cm , index = [i for i in encoder.categories_] , columns = [i for i in encoder.categories_])
sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True, fmt='')
plt.title('Confusion Matrix', size=20)
plt.xlabel('Predicted Labels', size=14)
plt.ylabel('Actual Labels', size=14)
plt.show()
```



Actual \ Predicted	angry	calm	disgust	fear	happy	neutral	sad	surprise
angry	83	0	10	7	23	1	3	5
calm	0	116	4	2	2	9	26	0
disgust	8	7	67	11	21	2	12	7
fear	2	0	4	106	20	0	16	3
happy	9	4	2	9	90	5	20	2
neutral	1	9	4	2	1	35	20	0
sad	0	20	2	13	9	11	92	4
surprise	2	1	6	10	21	3	6	90

In [32]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       angry       0.79      0.63      0.70       132
        calm       0.74      0.73      0.73       159
     disgust       0.68      0.50      0.57       135
        fear       0.66      0.70      0.68       151
       happy       0.48      0.64      0.55       141
     neutral       0.53      0.49      0.51        72
         sad       0.47      0.61      0.53       151
    surprise       0.81      0.65      0.72       139

    accuracy                           0.63      1080
   macro avg       0.65      0.62      0.62      1080
weighted avg       0.65      0.63      0.63      1080
```

In [ ]: