In [2]:
```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import librosa.display
import os

from sklearn.model_selection import train_test_split
```

In [3]:
```
path = '../input/cremad/AudioWAV/'
audio_path = []
audio_emotion = []
```

In [4]:
```
# collects all the audio filename in the variable 'path'
directory_path = os.listdir(path)
```

In [5]:
```
for audio in directory_path:
    audio_path.append(path + audio)
    emotion = audio.split('_')
    if emotion[2] == 'SAD':
        audio_emotion.append("sad")
    elif emotion[2] == 'ANG':
        audio_emotion.append("angry")
    elif emotion[2] == 'DIS':
        audio_emotion.append("disgust")
    elif emotion[2] == 'NEU':
        audio_emotion.append("neutral")
    elif emotion[2] == 'HAP':
        audio_emotion.append("happy")
    elif emotion[2] == 'FEA':
        audio_emotion.append("fear")
    else:
        audio_emotion.append("unknown")
```

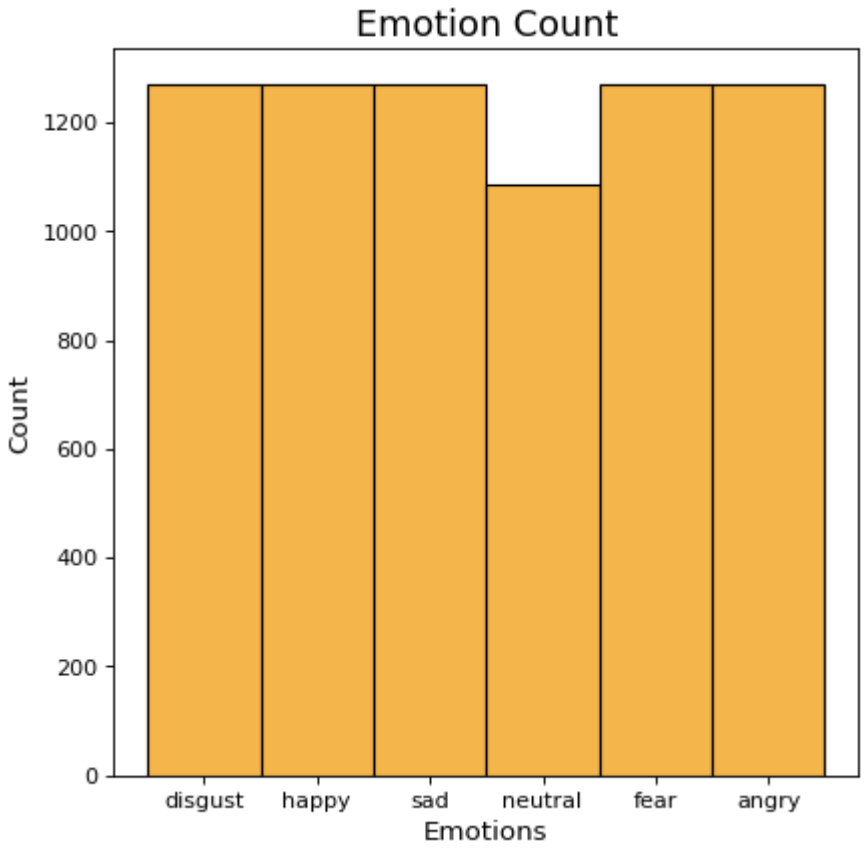
In [6]:
```
emotion_dataset = pd.DataFrame(audio_emotion, columns=['Emotions'])
audio_path_dataset = pd.DataFrame(audio_path, columns=['Path'])
dataset = pd.concat([audio_path_dataset, emotion_dataset], axis= 1)
#print(len(dataset))
print(dataset.head())
# print(dataset['File Path'][55])
```

```
                                        Path Emotions
0  ../input/cremad/AudioWAV/1028_TSI_DIS_XX.wav  disgust
1  ../input/cremad/AudioWAV/1075_IEO_HAP_LO.wav    happy
2  ../input/cremad/AudioWAV/1084_ITS_HAP_XX.wav    happy
3  ../input/cremad/AudioWAV/1067_IWW_DIS_XX.wav  disgust
4  ../input/cremad/AudioWAV/1066_TIE_DIS_XX.wav  disgust
```

Visualization

In [7]:
```
# counting audio categorized by emotions
plt.figure(figsize=(6,6), dpi=80)
plt.title("Emotion Count", size=16)
plt.xlabel('Emotions', size = 12)
plt.ylabel('Count', size = 12)
sns.histplot(dataset.Emotions, color='#F19C0E')
#plt.show()
```

Out[7]: <AxesSubplot:title={'center':'Emotion Count'}, xlabel='Emotions', ylabel='Count'>



Showing spectogram and waveplot

In [8]:
```
emotion_sad = dataset[dataset['Emotions']=='sad']['Path']
print(type(emotion_sad))
```

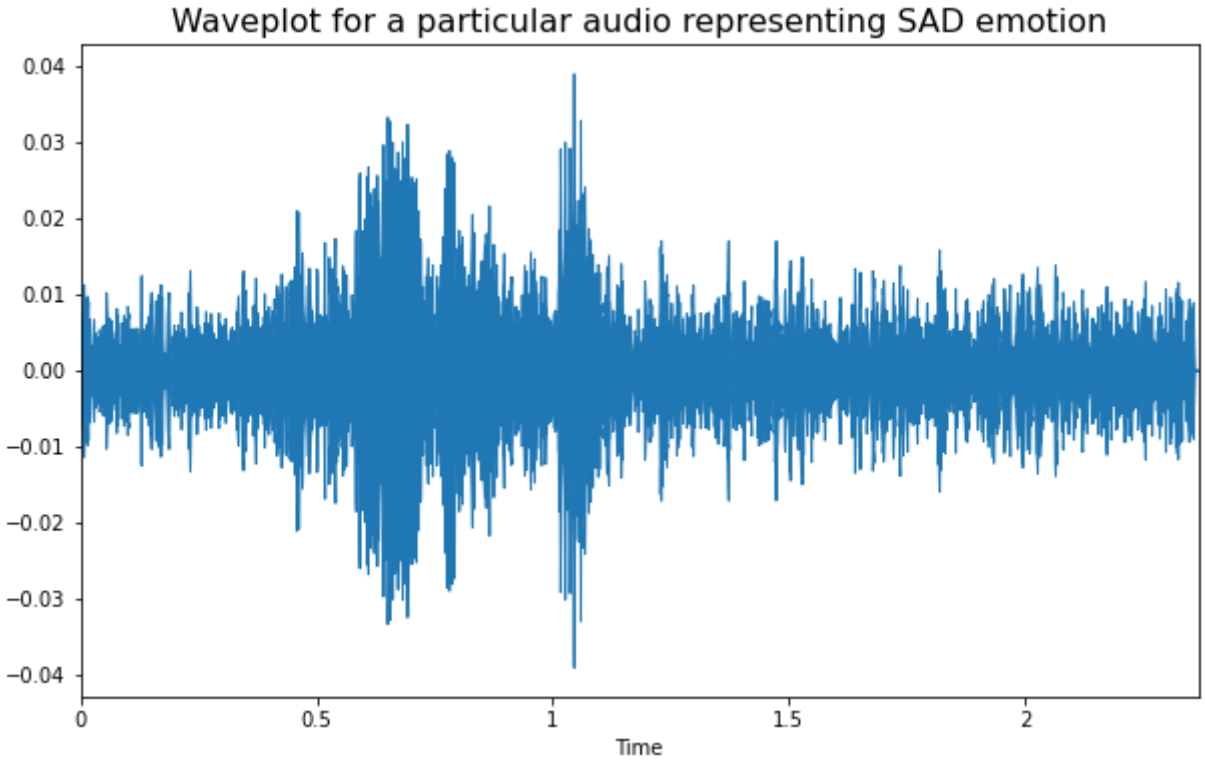
```
<class 'pandas.core.series.Series'>
```

In [9]:
```
#choosing a file to plot wave and spectogram
#print(emotion_sad.values[65])
data_path = emotion_sad.values[542]
data, sampling_rate = librosa.load(data_path)
```

Waveplot

In [10]:
```
plt.figure(figsize=(10,6))
plt.title("Waveplot for a particular audio representing SAD emotion", size=16)
librosa.display.waveplot(data, sr=sampling_rate)
#plt.show()
```

Out[10]: <matplotlib.collections.PolyCollection at 0x7f581b1dc1d0>

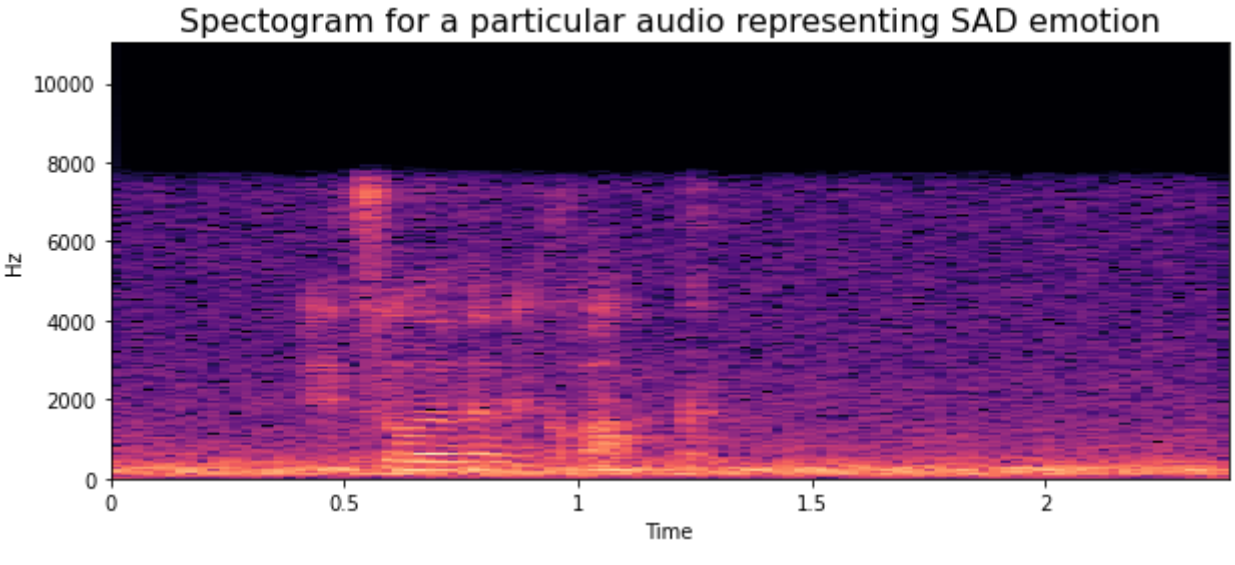


Spectogram

In [11]:
```

plt.figure(figsize=(10,4))
plt.title("Spectogram for a particular audio representing SAD emotion", size=16)
D = librosa.stft(data)
S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)
librosa.display.specshow(S_db, sr = sampling_rate, x_axis='time', y_axis='hz')
#plt.show()
```

Out[11]: <matplotlib.collections.QuadMesh at 0x7f581bd23910>

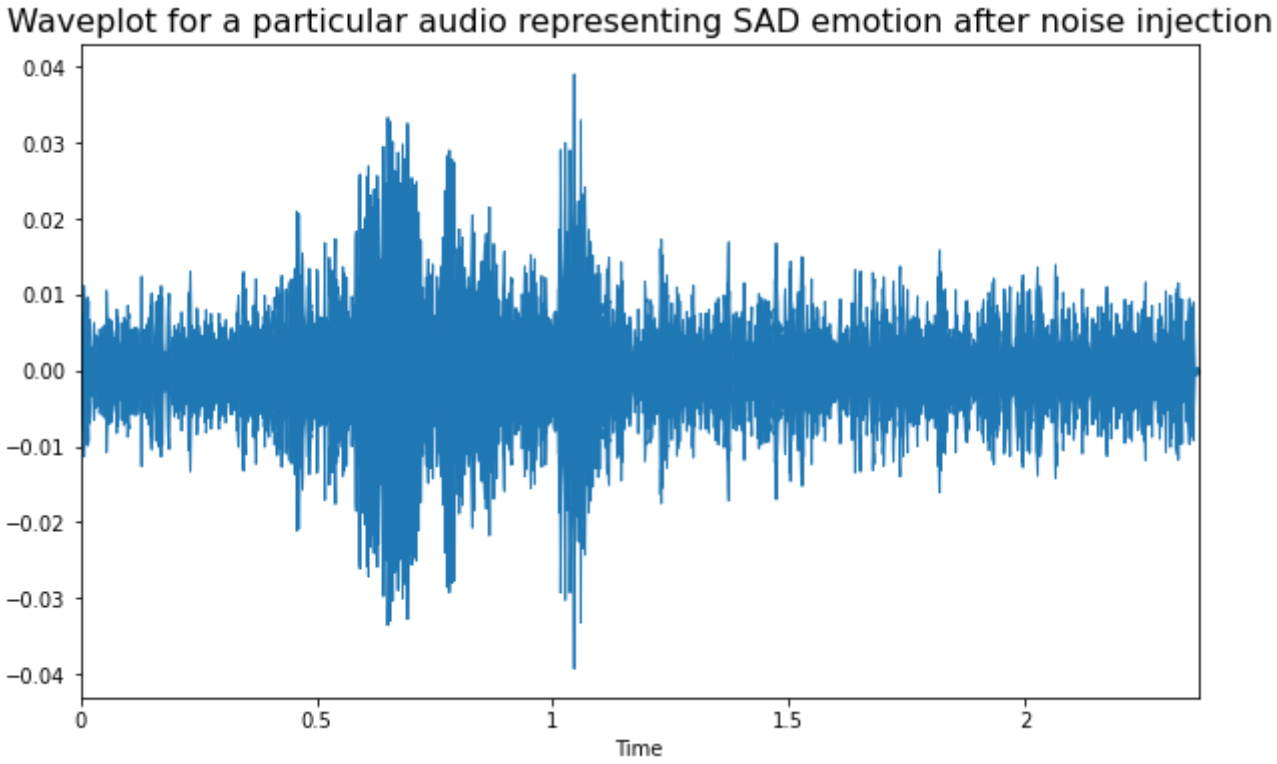


Augmentation (Noise Injection)

In [12]:
```
# for audio processing accuracy
# add noise to audio and check how the waveplot changes
# also the observing the change in audio quality

## Augmentation (Noise Injection)
noise_amp = 0.035*np.random.uniform()*np.amax(data)
audio_injected_data = data + noise_amp*np.random.normal(size=data.shape[0])

# waveplot view after noise injection:
plt.figure(figsize=(10,6))
plt.title("Waveplot for a particular audio representing SAD emotion after noise inj
librosa.display.waveplot(audio_injected_data, sr=sampling_rate)
plt.show()
```



Fearure Extraction

Creating a DF with extracted Feautures

In [13]:

```
X, Y = [], []
print("Feature processing...")

for path, emo, index in zip(dataset.Path, dataset.Emotions, range(len(dataset))):
    value, sample = librosa.load(path)
    # noise injection
    noise_amp = 0.035 * np.random.uniform() * np.amax(value)
    value = value + noise_amp * np.random.normal(size=value.shape[0])
    # mfcc
    mfcc = librosa.feature.mfcc(y=value, sr= sample, n_mfcc=13, n_fft=200, hop_leng
    mfcc = np.ravel(mfcc.T)
    # mel
    mel = librosa.feature.melspectrogram(y=value, sr=sample, hop_length = 256, n_ff
    mel = librosa.power_to_db(mel ** 2)
    mel = np.ravel(mel).T
    result = np.array([])
    result = np.hstack((result, mfcc, mel))
    #print(result)
    result = np.array(result)
    X.append(result)
    Y.append(emo)
```

```
Feature processing...
```

In [14]:

```
# print(X)
# print(Y)
extracted_audio_df = pd.DataFrame(X)
extracted_audio_df["emotion_of_audio"] = Y
print(extracted_audio_df.shape)
print(extracted_audio_df.tail(10))
extracted_audio_df = extracted_audio_df.fillna(0)
#print(extracted_audio_df.isna().any())
```

```
(7442, 30457)
                0          1          2          3          4          5  \
7432 -559.200843 -73.280942  -9.686211  -7.283784  18.869909  -8.773296
7433 -512.106808 -58.911652 -10.667038  16.432381  20.050270  -6.125210
7434 -512.564702 -78.250346   0.999713  10.071348  21.047835   1.798151
7435 -687.661885 -91.652409   4.695701  28.066455  10.075825  -3.847991
7436 -534.066959 -76.568546  -8.396436  -6.301705   1.900523   0.690286
7437 -603.322967 -92.506487  10.069625  34.567998  -0.123639  -0.325672
7438 -501.625926 -97.038078 -23.161478   9.334335   7.236941   9.445802
7439 -544.170771 -91.829970   3.046557  12.402814  12.947438  13.671182
7440 -792.501669 -46.832100   8.801988  33.063864  12.468825 -16.028599
7441 -745.219608 -49.169030  12.641756  29.362595  -6.108275  -4.888654

               6          7          8          9  ...  30447  30448  30449  \
7432 -26.851384   0.918844   4.587700 -13.149765  ...    NaN    NaN    NaN
7433 -12.220169  -2.864142  -3.623588   3.218214  ...    NaN    NaN    NaN
7434 -30.792923 -18.055478   6.103335   3.545619  ...    NaN    NaN    NaN
7435   2.544490  -0.391296  -9.519397  -0.084294  ...    NaN    NaN    NaN
7436 -16.713967  -2.741473   4.235537   3.132909  ...    NaN    NaN    NaN
7437  14.165165  -9.085516 -13.073067  -3.160416  ...    NaN    NaN    NaN
7438  -8.530354   2.094583  17.751684 -14.345202  ...    NaN    NaN    NaN
7439   6.281240 -24.027286 -17.749574  13.954773  ...    NaN    NaN    NaN
7440   5.045405   2.159563   3.349647  -4.139884  ...    NaN    NaN    NaN
7441   3.836113  10.753297   6.675437 -12.665519  ...    NaN    NaN    NaN

      30450  30451  30452  30453  30454  30455  emotion_of_audio
7432    NaN    NaN    NaN    NaN    NaN    NaN             angry
7433    NaN    NaN    NaN    NaN    NaN    NaN             happy
7434    NaN    NaN    NaN    NaN    NaN    NaN             angry
7435    NaN    NaN    NaN    NaN    NaN    NaN               sad
7436    NaN    NaN    NaN    NaN    NaN    NaN             angry
7437    NaN    NaN    NaN    NaN    NaN    NaN             angry
7438    NaN    NaN    NaN    NaN    NaN    NaN             angry
7439    NaN    NaN    NaN    NaN    NaN    NaN             angry
7440    NaN    NaN    NaN    NaN    NaN    NaN               sad
7441    NaN    NaN    NaN    NaN    NaN    NaN               sad

[10 rows x 30457 columns]
```

Training

In [15]:

```
# preparing to train
X = extracted_audio_df.drop(labels='emotion_of_audio', axis= 1)
Y = extracted_audio_df['emotion_of_audio']

x_train, x_test, y_train, y_test = train_test_split(np.array(X), Y, test_size=0.2)
```

In [16]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score


scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Model Creation and Fitting

In [17]:
```
mlp_model = MLPClassifier(activation='relu',
                          solver='sgd',
                          hidden_layer_sizes=100,
                          alpha=0.839903176695813,
                          batch_size=150,
                          learning_rate='adaptive',
                          max_iter=100000)
# Fit mlp model
mlp_model.fit(x_train,y_train)
```

Out[17]:
```
MLPClassifier(alpha=0.839903176695813, batch_size=150, hidden_layer_sizes=100,
              learning_rate='adaptive', max_iter=100000, solver='sgd')
```

Accuracy Calculation

In [24]:
```
y_pred = mlp_model.predict(x_test)
accuracy=accuracy_score(y_true=y_test, y_pred=y_pred)

# the accuracy didn't turn out to be that good :(
print("\nModel:{}    Accuracy: {:.2f}%".
          format(type(mlp_model).__name__ , accuracy*100))
```

```
Model:MLPClassifier    Accuracy: 39.29%
```

Prediction Verification

In [25]:
```
# the prediction made by the model:
print("The Prediction Made By Model: ")
print("<<<=========================================>>>")
df = pd.DataFrame({'Actual': y_test, 'Predict': y_pred})
print(df.head())
```

```
The Prediction Made By Model:
<<<=========================================>>>
       Actual  Predict
7163  disgust    angry
5450    happy  neutral
3992    happy    happy
2441     fear    angry
2390  neutral  neutral
```