```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'ravdess-emotional-speech-audio:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F107620%2F256618%2Fbundle%2Farchive.zip%3FX-↵

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading ravdess-emotional-speech-audio, 450102890 bytes compressed
[==================================================] 450102890 bytes downloaded
Downloaded and uncompressed: ravdess-emotional-speech-audio
Downloading ravdess-emotional-song-audio, 477641670 bytes compressed
[==================================================] 477641670 bytes downloaded
Downloaded and uncompressed: ravdess-emotional-song-audio
Downloading toronto-emotional-speech-set-tess, 448572034 bytes compressed
[==================================================] 448572034 bytes downloaded
Downloaded and uncompressed: toronto-emotional-speech-set-tess
Downloading cremad, 473324524 bytes compressed
[==================================================] 473324524 bytes downloaded
Downloaded and uncompressed: cremad
Downloading surrey-audiovisual-expressed-emotion-savee, 112690765 bytes compressed
[==================================================] 112690765 bytes downloaded
Downloaded and uncompressed: surrey-audiovisual-expressed-emotion-savee
Downloading features, 38593913 bytes compressed
[==================================================] 38593913 bytes downloaded
Downloaded and uncompressed: features
Data source import complete.
```

```
import os

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

# 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

# to play the audio files
from IPython.display import Audio
plt.style.use('seaborn-white')
from scipy.signal import resample
```

```
<ipython-input-2-2f393a031fbb>:15: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn-white')
```

```
DATA_FRAMES = True
fem_path = '../input/features/Female_features.csv'
mal_path = '../input/features/Male_features.csv'
#TESS - 2,800
TESS = "../input/toronto-emotional-speech-set-tess/tess toronto emotional speech set data/TESS Toronto emotional speech set data/"
#RAVDESS - 2,076
RAV = "../input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
#SAVEE - 480
SAVEE = "../input/surrey-audiovisual-expressed-emotion-savee/ALL/"
#CREMA-D - 7,442
CREMA = "../input/cremad/AudioWAV/"
```

```
# Get the data location for SAVEE
dir_list = os.listdir(SAVEE)

# parse the filename to get the emotions
emotion=[]
path = []
for i in dir_list:
    if i[-8:-6]=='_a':
        emotion.append('angry')
#     elif i[-8:-6]=='_d':
#         emotion.append('disgust')
#     elif i[-8:-6]=='_f':
#         emotion.append('fear')
    elif i[-8:-6]=='_h':
        emotion.append('happy')
    elif i[-8:-6]=='_n':
        emotion.append('neutral')
    elif i[-8:-6]=='sa':
        emotion.append('sad')
#     elif i[-8:-6]=='su':
#         emotion.append('surprise')
    else:
        emotion.append('unknown')
    path.append(SAVEE + i)

# # Now check out the label count distribution
# SAVEE_df = pd.DataFrame(emotion, columns = ['labels'])
# SAVEE_df = pd.concat([SAVEE_df, pd.DataFrame(path, columns = ['path'])], axis = 1)
# print('SAVEE dataset')
# SAVEE_df.head()
# Create DataFrame
SAVEE_df = pd.DataFrame({'labels': emotion, 'path': path})

# Filter out rows with labels not equal to 'unknown'
SAVEE_df = SAVEE_df[SAVEE_df['labels'] != 'unknown']

print('SAVEE dataset')
SAVEE_df.head()
```

SAVEE dataset

	labels	path
2	sad	../input/surrey-audiovisual-expressed-emotion-...
4	neutral	../input/surrey-audiovisual-expressed-emotion-...
6	neutral	../input/surrey-audiovisual-expressed-emotion-...
7	sad	../input/surrey-audiovisual-expressed-emotion-...
9	angry	../input/surrey-audiovisual-expressed-emotion-...

Next steps: Generate code with SAVEE_df View recommended plots

```
# Get the data location for TESS
path = []
emotion = []
dir_list = os.listdir(TESS)

for i in dir_list:
    fname = os.listdir(TESS + i)
    for f in fname:
        if i == 'OAF_angry' or i == 'YAF_angry':
            emotion.append('angry')
#         elif i == 'OAF_disgust' or i == 'YAF_disgust':
#             emotion.append('disgust')
#         elif i == 'OAF_Fear' or i == 'YAF_fear':
#             emotion.append('fear')
        elif i == 'OAF_happy' or i == 'YAF_happy':
            emotion.append('happy')
        elif i == 'OAF_neutral' or i == 'YAF_neutral':
            emotion.append('neutral')
#         elif i == 'OAF_Pleasant_surprise' or i == 'YAF_pleasant_surprised':
#             emotion.append('surprise')
        elif i == 'OAF_Sad' or i == 'YAF_sad':
            emotion.append('sad')
        else:
            emotion.append('Unknown')
        path.append(TESS + i + "/" + f)
TESS_df = pd.DataFrame({'labels': emotion, 'path': path})

# Filter out rows with labels not equal to 'Unknown'
TESS_df = TESS_df[TESS_df['labels'] != 'Unknown']

#TESS_df = pd.DataFrame(emotion, columns = ['labels'])
#TESS_df['source'] = 'TESS'
#TESS_df = pd.concat([TESS_df,pd.DataFrame(path, columns = ['path'])],axis=1)
print('TESS dataset')
TESS_df.head()
```

TESS dataset

	labels	path
0	happy	../input/toronto-emotional-speech-set-tess/tes...
1	happy	../input/toronto-emotional-speech-set-tess/tes...
2	happy	../input/toronto-emotional-speech-set-tess/tes...
3	happy	../input/toronto-emotional-speech-set-tess/tes...
4	happy	../input/toronto-emotional-speech-set-tess/tes...

Next steps: Generate code with TESS_df View recommended plots

```
# Importing datas from RAVDESS
dir = os.listdir(RAV)

males = []
females = []

for actor in dir:

    files = os.listdir(RAV + actor)

    for file in files:
        part = file.split('.')[0]
        part = part.split("-")

        temp = int(part[6])

        if part[2] == '01':
            emotion = 'neutral'
        elif part[2] == '03':
            emotion = 'happy'
        elif part[2] == '04':
            emotion = 'sad'
        elif part[2] == '05':
            emotion = 'angry'
        else:
            emotion = 'unknown'

        if temp%2 == 0:
            path = (RAV + actor + '/' + file)
            #emotion = 'female_'+emotion
            females.append([emotion, path])
        else:
            path = (RAV + actor + '/' + file)
             #emotion = 'male_'+emotion
            males.append([emotion, path])


# Create DataFrames
RavFemales_df = pd.DataFrame(females, columns=['labels', 'path'])
RavMales_df = pd.DataFrame(males, columns=['labels', 'path'])

# Filter out rows with labels not equal to 'unknown'
RavFemales_df = RavFemales_df[RavFemales_df['labels'] != 'unknown']
RavMales_df = RavMales_df[RavMales_df['labels'] != 'unknown']

print('RAVDESS datasets - Females')
RavFemales_df.head()
```

RAVDESS datasets - Females

	labels	path
0	happy	../input/ravdess-emotional-speech-audio/audio_...
1	happy	../input/ravdess-emotional-speech-audio/audio_...
2	sad	../input/ravdess-emotional-speech-audio/audio_...
4	angry	../input/ravdess-emotional-speech-audio/audio_...
6	neutral	../input/ravdess-emotional-speech-audio/audio_...

Next steps: Generate code with RavFemales_df View recommended plots

```
print('\nRAVDESS datasets - Males')
RavMales_df.head()
```

RAVDESS datasets - Males

	labels	path
0	happy	../input/ravdess-emotional-speech-audio/audio_...
3	sad	../input/ravdess-emotional-speech-audio/audio_...
4	happy	../input/ravdess-emotional-speech-audio/audio_...
6	sad	../input/ravdess-emotional-speech-audio/audio_...
7	happy	../input/ravdess-emotional-speech-audio/audio_...

Next steps: Generate code with RavMales_df View recommended plots

```
# Importing datas from RAVDESS
dir = os.listdir(RAV)

males = []
females = []

for actor in dir:

    files = os.listdir(RAV + actor)

    for file in files:
        part = file.split('.')[0]
        part = part.split("-")

        temp = int(part[6])

        if part[2] == '01':
            emotion = 'neutral'
#         elif part[2] == '02':
#             emotion = 'calm'
        elif part[2] == '03':
            emotion = 'happy'
        elif part[2] == '04':
            emotion = 'sad'
        elif part[2] == '05':
            emotion = 'angry'
#         elif part[2] == '06':
#             emotion = 'fear'
#         elif part[2] == '07':
#             emotion = 'disgust'
#         elif part[2] == '08':
#             emotion = 'surprise'
        else:
            emotion = 'unknown'

        if temp%2 == 0:
            path = (RAV + actor + '/' + file)
            #emotion = 'female_'+emotion
            females.append([emotion, path])
        else:
            path = (RAV + actor + '/' + file)
             #emotion = 'male_'+emotion
            males.append([emotion, path])


# RavFemales_df = pd.DataFrame(females)
# RavFemales_df.columns = ['labels', 'path']

# RavMales_df = pd.DataFrame(males)
# RavMales_df.columns = ['labels', 'path']

# print('RAVDESS datasets')
# RavFemales_df.head()
# Create DataFrames
RavFemales_df = pd.DataFrame(females, columns=['labels', 'path'])
RavMales_df = pd.DataFrame(males, columns=['labels', 'path'])

# Filter out rows with labels not equal to 'unknown'
RavFemales_df = RavFemales_df[RavFemales_df['labels'] != 'unknown']
RavMales_df = RavMales_df[RavMales_df['labels'] != 'unknown']

print('RAVDESS datasets - Females')
RavFemales_df.head()
```

RAVDESS datasets - Females

	labels	path
0	happy	../input/ravdess-emotional-speech-audio/audio_...
1	happy	../input/ravdess-emotional-speech-audio/audio_...
2	sad	../input/ravdess-emotional-speech-audio/audio_...
4	angry	../input/ravdess-emotional-speech-audio/audio_...
6	neutral	../input/ravdess-emotional-speech-audio/audio_...

Next steps: Generate code with RavFemales_df View recommended plots

```
print('\nRAVDESS datasets - Males')
RavMales_df.head()
```

RAVDESS datasets - Males

	labels	path
0	happy	../input/ravdess-emotional-speech-audio/audio_...
3	sad	../input/ravdess-emotional-speech-audio/audio_...
4	happy	../input/ravdess-emotional-speech-audio/audio_...
6	sad	../input/ravdess-emotional-speech-audio/audio_...
7	happy	../input/ravdess-emotional-speech-audio/audio_...

Next steps: Generate code with RavMales_df View recommended plots

```
files = os.listdir(CREMA)

female = [1002,1003,1004,1006,1007,1008,1009,1010,1012,1013,1018,1020,1021,1024,1025,1028,1029,1030,1037,1043,1046,1047,1049,
          1052,1053,1054,1055,1056,1058,1060,1061,1063,1072,1073,1074,1075,1076,1078,1079,1082,1084,1089,1091]
males = []
females = []

for file in files:
    part = file.split('_')

    if part[2] == 'SAD':
        emotion = 'sad'
    elif part[2] == 'ANG':
        emotion = 'angry'
#     elif part[2] == 'DIS':
#         emotion = 'disgust'
#     elif part[2] == 'FEA':
#         emotion = 'fear'
    elif part[2] == 'HAP':
        emotion = 'happy'
    elif part[2] == 'NEU':
        emotion = 'neutral'
    else:
        emotion = 'unknown'

    if int(part[0]) in female:
        path = (CREMA + '/' + file)
        #emotion = 'female_'+emotion
        females.append([emotion, path])
    else:
        path = (CREMA + '/' + file)
        #emotion = 'male_'+emotion
        males.append([emotion, path])

# CremaFemales_df = pd.DataFrame(females)
# CremaFemales_df.columns = ['labels', 'path']

# CremaMales_df = pd.DataFrame(males)
# CremaMales_df.columns = ['labels', 'path']

# print('CREMA datasets')
# CremaFemales_df.head()
# Create DataFrames
CremaFemales_df = pd.DataFrame(females, columns=['labels', 'path'])
CremaMales_df = pd.DataFrame(males, columns=['labels', 'path'])

# Filter out rows with labels not equal to 'unknown'
CremaFemales_df = CremaFemales_df[CremaFemales_df['labels'] != 'unknown']
CremaMales_df = CremaMales_df[CremaMales_df['labels'] != 'unknown']

print('CREMA datasets - Females')
CremaFemales_df.head()
```

CREMA datasets - Females

	labels	path
0	happy	../input/cremad/AudioWAV//1056_DFA_HAP_XX.wav
2	happy	../input/cremad/AudioWAV//1010_TIE_HAP_XX.wav
4	happy	../input/cremad/AudioWAV//1030_TAI_HAP_XX.wav
5	neutral	../input/cremad/AudioWAV//1037_IEO_NEU_XX.wav
6	sad	../input/cremad/AudioWAV//1078_ITS_SAD_XX.wav

Next steps: Generate code with CremaFemales_df View recommended plots

```
print('\nCREMA datasets - Males')
CremaMales_df.head()
```

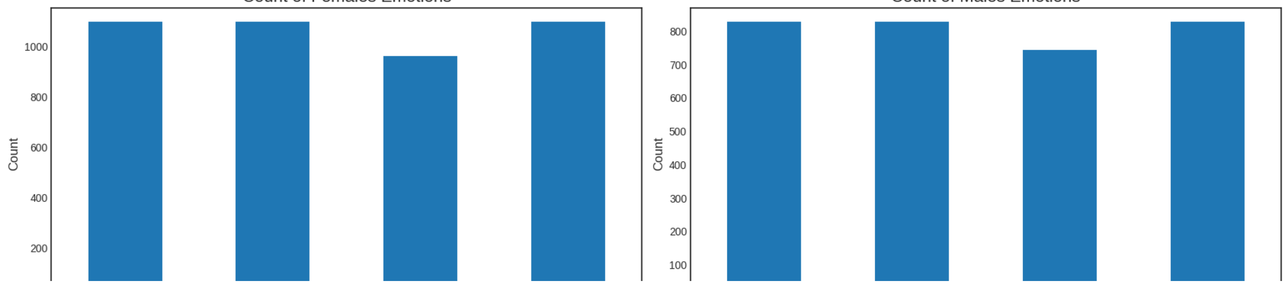
```
CREMA datasets - Males
```

	labels	path
1	angry	../input/cremad/AudioWAV//1064_MTI_ANG_XX.wav
4	sad	../input/cremad/AudioWAV//1015_IEO_SAD_MD.wav
7	neutral	../input/cremad/AudioWAV//1001_ITH_NEU_XX.wav
9	happy	../input/cremad/AudioWAV//1042_ITS_HAP_XX.wav
10	sad	../input/cremad/AudioWAV//1019_IEO_SAD_MD.wav

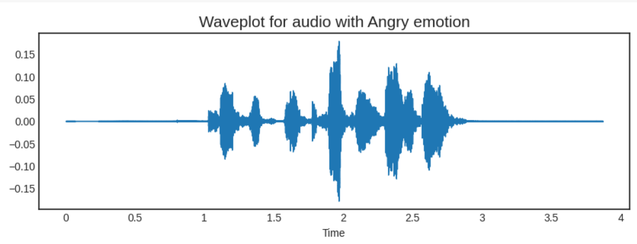
Next steps: Generate code with `CremaMales_df` View recommended plots

```
# Now lets merge all the dataframe
Males = pd.concat([SAVEE_df, RavMales_df, CremaMales_df], axis = 0)
Males.to_csv("males_emotions_df.csv", index = False)

Females = pd.concat([TESS_df, RavFemales_df, CremaFemales_df], axis = 0)
Females.to_csv("females_emotions_df.csv", index = False)
```

```
Females_labels_series = pd.Series(Females.labels)
Males_labels_series = pd.Series(Males.labels)

# Define the order of emotions
#order = ['angry', 'calm', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
order = ['angry', 'happy', 'neutral', 'sad']

# Create subplots
fig = plt.figure(figsize=(17, 5))

fig.add_subplot(121)
plt.title('Count of Females Emotions', size=16)
Females_labels_series.value_counts().loc[order].plot(kind='bar')
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility

fig.add_subplot(122)
plt.title('Count of Males Emotions', size=16)
Males_labels_series.value_counts().loc[order].plot(kind='bar')
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility

plt.tight_layout()
plt.show()
```

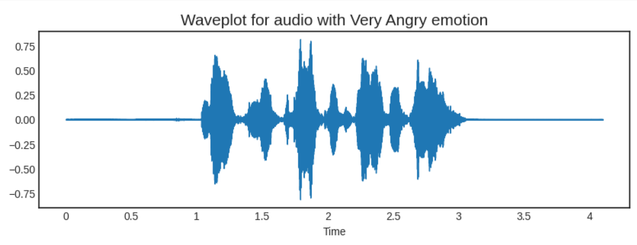



```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title(f'Waveplot for audio with {e} emotion', size=15)
    librosa.display.waveshow(data, sr=sr)
    plt.show()
emotion = 'Angry'
path = '../input/ravdess-emotional-speech-audio/Actor_01/03-01-05-01-01-01-01.wav'
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
Audio(path)
```



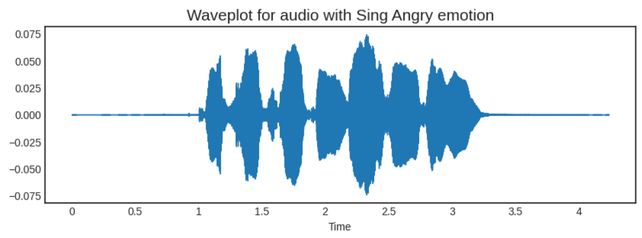
0:03 / 0:03

```
emotion='Very Angry'
path = '../input/ravdess-emotional-speech-audio/Actor_01/03-01-05-02-01-01-01.wav'
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
Audio(path)
```



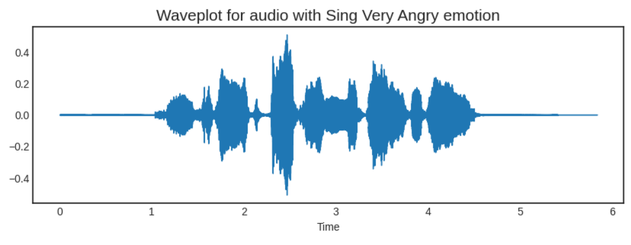
0:04 / 0:04

```
emotion='Sing Angry'
path = '../input/ravdess-emotional-song-audio/Actor_01/03-02-01-01-01-01-01.wav'
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
Audio(path)
```



0:04 / 0:04

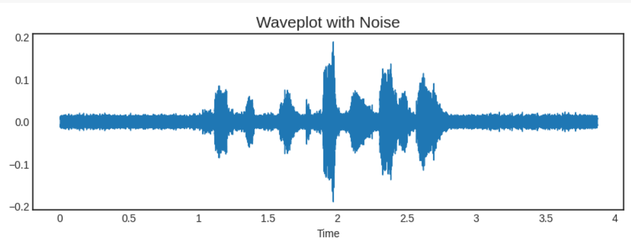
```
emotion='Sing Very Angry'
path = '../input/ravdess-emotional-song-audio/Actor_01/03-02-05-02-01-01-01.wav'
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
Audio(path)
```



0:05 / 0:05

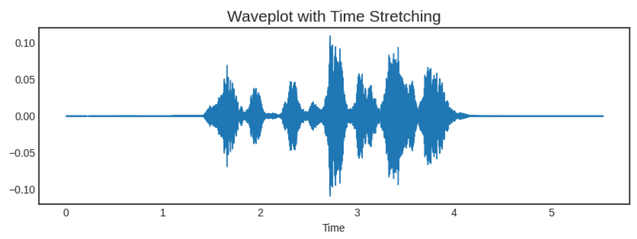
```
def noise(data):
    noise_amp = 0.04*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

def stretch(data, rate=0.70):
    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=0.8):
    return librosa.effects.pitch_shift(data, sampling_rate, pitch_factor)

def higher_speed(data, speed_factor = 1.25):
    return librosa.effects.time_stretch(data, speed_factor)

def lower_speed(data, speed_factor = 0.75):
    return librosa.effects.time_stretch(data, speed_factor)

# taking any example and checking for techniques.
path = path = '../input/ravdess-emotional-speech-audio/Actor_01/03-01-05-01-01-01-01.wav'
data, sample_rate = librosa.load(path)
```

```
plt.figure(figsize=(10, 3))
x = noise(data)  # Applying noise to the data
y = resample(x, len(data))  # Resampling the noisy data to match original length
librosa.display.waveshow(y, sr=sample_rate)
plt.title('Waveplot with Noise', size=15)
plt.show()

# Playing the audio with added noise
Audio(y, rate=sample_rate)
```



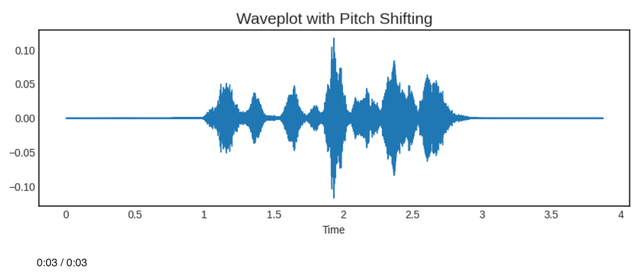
0:03 / 0:03

```
def stretch(data, rate=0.70):
    return librosa.effects.time_stretch(data, rate=rate)

plt.figure(figsize=(10, 3))
x = stretch(data)  # Assuming data is loaded or defined elsewhere
librosa.display.waveshow(y=x, sr=sample_rate)
plt.title('Waveplot with Time Stretching', size=15)
plt.show()

# Playing the audio after time stretching
Audio(x, rate=sample_rate)
```



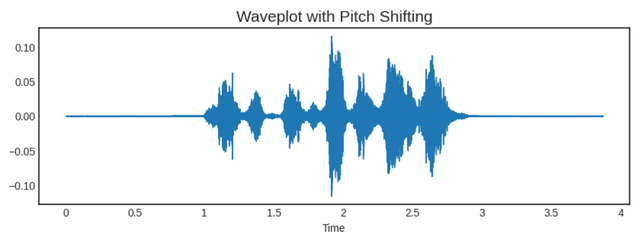
0:05 / 0:05

```
def shift(data, n_steps=2):
    return librosa.effects.pitch_shift(data, sr=sample_rate, n_steps=n_steps)

plt.figure(figsize=(10, 3))
x = shift(data)  # Assuming data is loaded or defined elsewhere
librosa.display.waveshow(y=x, sr=sample_rate)
plt.title('Waveplot with Pitch Shifting', size=15)
plt.show()

# Playing the audio after pitch shifting
Audio(x, rate=sample_rate)
```



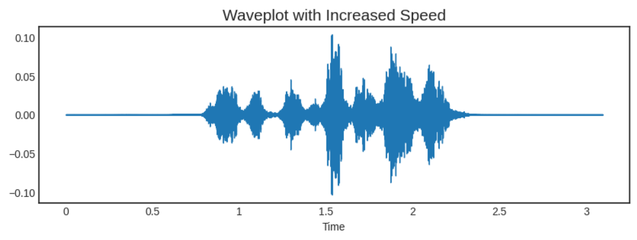
0:03 / 0:03

```
def pitch(data, sampling_rate, pitch_factor=0.8):
    return librosa.effects.pitch_shift(data, n_steps=int(pitch_factor * 12), sr=sampling_rate)

plt.figure(figsize=(10, 3))
x = pitch(data, sample_rate)
librosa.display.waveshow(y=x, sr=sample_rate)
plt.title('Waveplot with Pitch Shifting', size=15)
plt.show()

# Playing the audio after pitch shifting
Audio(x, rate=sample_rate)
```



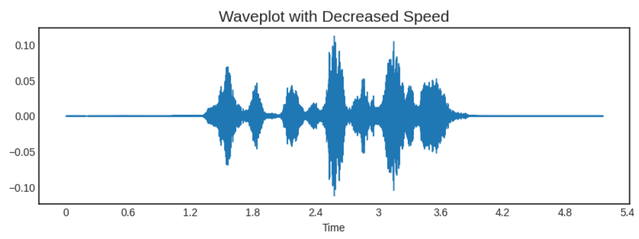
0:03 / 0:03

```
def higher_speed(data, sample_rate, speed_factor=1.25):
    return librosa.effects.time_stretch(y=data, rate=speed_factor)
plt.figure(figsize=(10, 3))
x = higher_speed(data, sample_rate)  # Assuming data and sample_rate are defined elsewhere
librosa.display.waveshow(y=x, sr=sample_rate)
plt.title('Waveplot with Increased Speed', size=15)
plt.show()

# Playing the audio after increasing the speed
Audio(x, rate=sample_rate)
```



0:03 / 0:03

```
def lower_speed(data, speed_factor=0.75):
    return librosa.effects.time_stretch(data, rate=speed_factor)

plt.figure(figsize=(10, 3))
x = lower_speed(data)  # Assuming data and sample_rate are defined elsewhere
librosa.display.waveshow(y=x, sr=sample_rate)
plt.title('Waveplot with Decreased Speed', size=15)
plt.show()

# Playing the audio after decreasing the speed
Audio(x, rate=sample_rate)
```



0:05 / 0:05

```
def extract_features(data):

    result = np.array([])

    #mfccs = librosa.feature.mfcc(y=data, sr=22050, n_mfcc=42) #42 mfcc so we get frames of ~60 ms
    mfccs = librosa.feature.mfcc(y=data, sr=22050, n_mfcc=58)
    mfccs_processed = np.mean(mfccs.T,axis=0)
    result = np.array(mfccs_processed)

    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=3, offset=0.5, res_type='kaiser_fast')

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

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

    #stretched
    stretch_data = stretch(data)
    res3 = extract_features(stretch_data)
    result = np.vstack((result, res3))

    #shifted
    shift_data = shift(data)
    res4 = extract_features(shift_data)
    result = np.vstack((result, res4))

    #pitched
    pitch_data = pitch(data, sample_rate)
    res5 = extract_features(pitch_data)
    result = np.vstack((result, res5))

    #speed up
    higher_speed_data = higher_speed(data)
    res6 = extract_features(higher_speed_data)
    result = np.vstack((result, res6))

    #speed down
    lower_speed_data = higher_speed(data)
    res7 = extract_features(lower_speed_data)
    result = np.vstack((result, res7))

    return result
```

```
if not DATA_FRAMES:

    valid_emotions = ['angry', 'happy', 'neutral', 'sad']

    female_X, female_Y = [], []
    for path, emotion in zip(Females.path, Females.labels):
        if emotion in valid_emotions:
            features = get_features(path)
            # adding augmentation, get_features return a multi-dimensional array (for each augmentation),
            # so we have to use a loop to fill the df
            for elem in features:
                female_X.append(elem)
                female_Y.append(emotion)

    male_X, male_Y = [], []
    for path, emotion in zip(Males.path, Males.labels):
        if emotion in valid_emotions:
            features = get_features(path)
            for elem in features:
                male_X.append(elem)
                male_Y.append(emotion)

    print(f'Check shapes:\nFemale features: {len(female_X)}, labels: {len(female_Y)}\nMale features:   {len(male_X)}, labels: {len(male_Y)}')
```

#

```
def setup_dataframe(gender, features, labels):
    df = pd.DataFrame(features)
    df['labels'] = labels

    # Filter only the desired labels
    valid_labels = ['angry', 'happy', 'neutral', 'sad']
    df = df[df['labels'].isin(valid_labels)]

    df.to_csv(f'{gender}_features.csv', index=False)

    print(f'{gender} dataframe')
    df.sample(frac=1).head()

    return df
```

```
if not DATA_FRAMES:
    Females_Features = setup_dataframe('Female', female_X, female_Y)
else:
    Females_Features = pd.read_csv(fem_path)
```

```
if not DATA_FRAMES:
    Males_Features = setup_dataframe('Male', male_X, male_Y)
else:
    Males_Features = pd.read_csv(mal_path)
```

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

```
valid_labels = ['angry', 'happy', 'neutral', 'sad']

# For Females
female_X = Females_Features[Females_Features['labels'].isin(valid_labels)].iloc[:, :-1].values
female_Y = Females_Features[Females_Features['labels'].isin(valid_labels)]['labels'].values

# For Males
male_X = Males_Features[Males_Features['labels'].isin(valid_labels)].iloc[:, :-1].values
male_Y = Males_Features[Males_Features['labels'].isin(valid_labels)]['labels'].values
```

```
# As this is a multiclass classification problem onehotencoding our Y.
encoder = OneHotEncoder()

female_Y = encoder.fit_transform(np.array(female_Y).reshape(-1,1)).toarray()
male_Y = encoder.fit_transform(np.array(male_Y).reshape(-1,1)).toarray()
```

```
nogender_X = np.concatenate((female_X, male_X))
nogender_Y = np.concatenate((female_Y, male_Y))

x_train, x_test, y_train, y_test = train_test_split(nogender_X, nogender_Y, random_state=0, test_size=0.20, shuffle=True)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((41843, 58), (41843, 4), (10461, 58), (10461, 4))
```

```
x_trainF, x_testF, y_trainF, y_testF = train_test_split(female_X, female_Y, random_state=0, test_size=0.20, shuffle=True)
x_trainF.shape, y_trainF.shape, x_testF.shape, y_testF.shape
```

```
((23788, 58), (23788, 4), (5948, 58), (5948, 4))
```

```
x_trainM, x_testM, y_trainM, y_testM = train_test_split(male_X, male_Y, random_state=0, test_size=0.20, shuffle=True)
x_trainM.shape, y_trainM.shape, x_testM.shape, y_testM.shape
```

```
((18054, 58), (18054, 4), (4514, 58), (4514, 4))
```

```
scaler = StandardScaler()

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

x_trainF = scaler.fit_transform(x_trainF)
x_testF = scaler.transform(x_testF)

x_trainM = scaler.fit_transform(x_trainM)
x_testM = scaler.transform(x_testM)
```

```
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
```

```
((41843, 58, 1), (41843, 4), (10461, 58, 1), (10461, 4))
```

```
x_trainF = np.expand_dims(x_trainF, axis=2)
x_testF = np.expand_dims(x_testF, axis=2)
x_trainF.shape, y_trainF.shape, x_testF.shape, y_testF.shape
```

```
((23788, 58, 1), (23788, 4), (5948, 58, 1), (5948, 4))
```

```
x_trainM = np.expand_dims(x_trainM, axis=2)
x_testM = np.expand_dims(x_testM, axis=2)
x_trainM.shape, y_trainM.shape, x_testM.shape, y_testM.shape
```

```
((18054, 58, 1), (18054, 4), (4514, 58, 1), (4514, 4))
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization, AveragePooling1D
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint
```

```
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

```
Num GPUs Available:  0
```

```
# Create a MirroredStrategy.
strategy = tf.distribute.MirroredStrategy()
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))
```

```
Number of devices: 1
```

```
with strategy.scope():

    def build_model(in_shape):

        model=Sequential()
        model.add(Conv1D(256, kernel_size=6, strides=1, padding='same', activation='relu', input_shape=(in_shape, 1)))
        model.add(AveragePooling1D(pool_size=4, strides = 2, padding = 'same'))

        model.add(Conv1D(128, kernel_size=6, strides=1, padding='same', activation='relu'))
        model.add(AveragePooling1D(pool_size=4, strides = 2, padding = 'same'))

        model.add(Conv1D(128, kernel_size=6, strides=1, padding='same', activation='relu'))
        model.add(AveragePooling1D(pool_size=4, strides = 2, padding = 'same'))
        model.add(Dropout(0.2))

        model.add(Conv1D(64, kernel_size=6, strides=1, padding='same', activation='relu'))
        model.add(MaxPooling1D(pool_size=4, strides = 2, padding = 'same'))

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

        model.add(Dense(units=4, activation='softmax'))
        model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])


        return model
```

```
def model_build_summary(mod_dim, tr_features, val_features, val_labels):
    model = build_model(mod_dim)
    model.summary()

    score = model.evaluate(val_features, val_labels, verbose = 1)
    accuracy = 100*score[1]

    return model
```

```
rlrp = ReduceLROnPlateau(monitor='loss', factor=0.4, verbose=0, patience=4, min_lr=0.000001)

batch_size = 32
n_epochs = 50
```

```
def show_graphs(history):
    epochs = [i for i in range(n_epochs)]
    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(30,12)
    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()
```

```
total_model = model_build_summary(x_train.shape[1], x_train, x_test, y_test)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 58, 256)           1792

 average_pooling1d (Average  (None, 29, 256)           0
 Pooling1D)

 conv1d_1 (Conv1D)           (None, 29, 128)           196736

 average_pooling1d_1 (Avera  (None, 15, 128)           0
 gePooling1D)

 conv1d_2 (Conv1D)           (None, 15, 128)           98432

 average_pooling1d_2 (Avera  (None, 8, 128)            0
 gePooling1D)

 dropout (Dropout)           (None, 8, 128)            0

 conv1d_3 (Conv1D)           (None, 8, 64)             49216

 max_pooling1d (MaxPooling1  (None, 4, 64)             0
 D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 32)                8224

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 4)                 132

=================================================================
Total params: 354532 (1.35 MB)
Trainable params: 354532 (1.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
327/327 [==============================] - 5s 14ms/step - loss: 1.3865 - accuracy: 0.2437
```

```
female_model = model_build_summary(x_trainF.shape[1], x_trainF, x_testF, y_testF)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_4 (Conv1D)           (None, 58, 256)           1792

 average_pooling1d_3 (Avera  (None, 29, 256)           0
 gePooling1D)

 conv1d_5 (Conv1D)           (None, 29, 128)           196736

 average_pooling1d_4 (Avera  (None, 15, 128)           0
 gePooling1D)

 conv1d_6 (Conv1D)           (None, 15, 128)           98432

 average_pooling1d_5 (Avera  (None, 8, 128)            0
 gePooling1D)
```

```
 dropout_2 (Dropout)         (None, 8, 128)            0

 conv1d_7 (Conv1D)           (None, 8, 64)             49216

 max_pooling1d_1 (MaxPoolin  (None, 4, 64)             0
 g1D)

 flatten_1 (Flatten)         (None, 256)               0

 dense_2 (Dense)             (None, 32)                8224

 dropout_3 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 4)                 132

=================================================================
Total params: 354532 (1.35 MB)
Trainable params: 354532 (1.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
186/186 [==============================] - 3s 14ms/step - loss: 1.3860 - accuracy: 0.2892
```

```
male_model = model_build_summary(x_trainM.shape[1], x_trainM, x_testM, y_testM)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_8 (Conv1D)           (None, 58, 256)           1792

 average_pooling1d_6 (Avera  (None, 29, 256)           0
 gePooling1D)

 conv1d_9 (Conv1D)           (None, 29, 128)           196736

 average_pooling1d_7 (Avera  (None, 15, 128)           0
 gePooling1D)

 conv1d_10 (Conv1D)          (None, 15, 128)           98432

 average_pooling1d_8 (Avera  (None, 8, 128)            0
 gePooling1D)

 dropout_4 (Dropout)         (None, 8, 128)            0

 conv1d_11 (Conv1D)          (None, 8, 64)             49216

 max_pooling1d_2 (MaxPoolin  (None, 4, 64)             0
 g1D)

 flatten_2 (Flatten)         (None, 256)               0

 dense_4 (Dense)             (None, 32)                8224

 dropout_5 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 4)                 132

=================================================================
Total params: 354532 (1.35 MB)
Trainable params: 354532 (1.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
142/142 [==============================] - 3s 18ms/step - loss: 1.3874 - accuracy: 0.2288
```

```
history = total_model.fit(x_train, y_train, batch_size=batch_size, epochs=n_epochs, validation_data=(x_test, y_test), callbacks=[rlrp])
```

```
Epoch 1/50
1308/1308 [==============================] - 74s 56ms/step - loss: 0.8815 - accuracy: 0.6015 - val_loss: 0.7257 - val_accuracy: 0.6780 - lr: 0.0010
Epoch 2/50
1308/1308 [==============================] - 75s 58ms/step - loss: 0.7224 - accuracy: 0.6866 - val_loss: 0.6599 - val_accuracy: 0.7154 - lr: 0.0010
Epoch 3/50
1308/1308 [==============================] - 75s 57ms/step - loss: 0.6634 - accuracy: 0.7162 - val_loss: 0.6102 - val_accuracy: 0.7342 - lr: 0.0010
Epoch 4/50
1308/1308 [==============================] - 74s 57ms/step - loss: 0.6209 - accuracy: 0.7365 - val_loss: 0.5796 - val_accuracy: 0.7550 - lr: 0.0010
Epoch 5/50
1308/1308 [==============================] - 74s 57ms/step - loss: 0.5770 - accuracy: 0.7557 - val_loss: 0.5632 - val_accuracy: 0.7611 - lr: 0.0010
Epoch 6/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.5434 - accuracy: 0.7717 - val_loss: 0.5260 - val_accuracy: 0.7791 - lr: 0.0010
Epoch 7/50
1308/1308 [==============================] - 76s 58ms/step - loss: 0.5089 - accuracy: 0.7887 - val_loss: 0.4898 - val_accuracy: 0.7985 - lr: 0.0010
Epoch 8/50
1308/1308 [==============================] - 76s 58ms/step - loss: 0.4694 - accuracy: 0.8083 - val_loss: 0.4637 - val_accuracy: 0.8068 - lr: 0.0010
Epoch 9/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.4349 - accuracy: 0.8217 - val_loss: 0.4544 - val_accuracy: 0.8135 - lr: 0.0010
Epoch 10/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.4048 - accuracy: 0.8359 - val_loss: 0.4214 - val_accuracy: 0.8348 - lr: 0.0010
Epoch 11/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.3749 - accuracy: 0.8500 - val_loss: 0.3663 - val_accuracy: 0.8573 - lr: 0.0010
Epoch 12/50
1308/1308 [==============================] - 74s 57ms/step - loss: 0.3492 - accuracy: 0.8622 - val_loss: 0.3731 - val_accuracy: 0.8537 - lr: 0.0010
Epoch 13/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.3221 - accuracy: 0.8741 - val_loss: 0.3515 - val_accuracy: 0.8662 - lr: 0.0010
Epoch 14/50
1308/1308 [==============================] - 72s 55ms/step - loss: 0.3046 - accuracy: 0.8787 - val_loss: 0.3324 - val_accuracy: 0.8755 - lr: 0.0010
Epoch 15/50
1308/1308 [==============================] - 79s 61ms/step - loss: 0.2835 - accuracy: 0.8906 - val_loss: 0.3195 - val_accuracy: 0.8806 - lr: 0.0010
Epoch 16/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.2698 - accuracy: 0.8958 - val_loss: 0.2939 - val_accuracy: 0.8932 - lr: 0.0010
Epoch 17/50
1308/1308 [==============================] - 72s 55ms/step - loss: 0.2531 - accuracy: 0.9047 - val_loss: 0.2821 - val_accuracy: 0.9024 - lr: 0.0010
Epoch 18/50
1308/1308 [==============================] - 75s 58ms/step - loss: 0.2448 - accuracy: 0.9085 - val_loss: 0.2957 - val_accuracy: 0.8926 - lr: 0.0010
Epoch 19/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.2287 - accuracy: 0.9151 - val_loss: 0.2789 - val_accuracy: 0.8981 - lr: 0.0010
Epoch 20/50
1308/1308 [==============================] - 72s 55ms/step - loss: 0.2170 - accuracy: 0.9182 - val_loss: 0.2676 - val_accuracy: 0.9028 - lr: 0.0010
Epoch 21/50
1308/1308 [==============================] - 74s 56ms/step - loss: 0.2101 - accuracy: 0.9224 - val_loss: 0.2604 - val_accuracy: 0.9083 - lr: 0.0010
Epoch 22/50
1308/1308 [==============================] - 74s 57ms/step - loss: 0.1960 - accuracy: 0.9264 - val_loss: 0.2745 - val_accuracy: 0.9058 - lr: 0.0010
Epoch 23/50
1308/1308 [==============================] - 74s 57ms/step - loss: 0.1961 - accuracy: 0.9280 - val_loss: 0.2653 - val_accuracy: 0.9038 - lr: 0.0010
Epoch 24/50
1308/1308 [==============================] - 74s 56ms/step - loss: 0.1836 - accuracy: 0.9319 - val_loss: 0.2599 - val_accuracy: 0.9130 - lr: 0.0010
Epoch 25/50
1308/1308 [==============================] - 75s 57ms/step - loss: 0.1817 - accuracy: 0.9338 - val_loss: 0.2444 - val_accuracy: 0.9199 - lr: 0.0010
Epoch 26/50
1308/1308 [==============================] - 72s 55ms/step - loss: 0.1612 - accuracy: 0.9421 - val_loss: 0.2498 - val_accuracy: 0.9175 - lr: 0.0010
Epoch 27/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.1714 - accuracy: 0.9372 - val_loss: 0.2651 - val_accuracy: 0.9187 - lr: 0.0010
Epoch 28/50
1308/1308 [==============================] - 74s 57ms/step - loss: 0.1602 - accuracy: 0.9414 - val_loss: 0.2431 - val_accuracy: 0.9216 - lr: 0.0010
Epoch 29/50
1308/1308 [==============================] - 73s 56ms/step - loss: 0.1515 - accuracy: 0.9451 - val_loss: 0.2379 - val_accuracy: 0.9270 - lr: 0.0010
```

```
female_history = female_model.fit(x_trainF, y_trainF, batch_size=batch_size, epochs=n_epochs, validation_data=(x_testF, y_testF), callbacks=[rlrp])
```

```
Epoch 22/50
744/744 [==============================] - 45s 60ms/step - loss: 0.0056 - accuracy: 0.9982 - val_loss: 0.1753 - val_accuracy: 0.9709 - lr: 1.6000e-04
Epoch 23/50
744/744 [==============================] - 44s 59ms/step - loss: 0.0055 - accuracy: 0.9982 - val_loss: 0.1722 - val_accuracy: 0.9707 - lr: 1.6000e-04
Epoch 24/50
744/744 [==============================] - 45s 61ms/step - loss: 0.0045 - accuracy: 0.9984 - val_loss: 0.1789 - val_accuracy: 0.9682 - lr: 1.6000e-04
Epoch 25/50
744/744 [==============================] - 43s 58ms/step - loss: 0.0062 - accuracy: 0.9980 - val_loss: 0.1785 - val_accuracy: 0.9691 - lr: 1.6000e-04
Epoch 26/50
744/744 [==============================] - 41s 56ms/step - loss: 0.0055 - accuracy: 0.9982 - val_loss: 0.1766 - val_accuracy: 0.9711 - lr: 1.6000e-04
Epoch 27/50
744/744 [==============================] - 40s 53ms/step - loss: 0.0066 - accuracy: 0.9982 - val_loss: 0.1804 - val_accuracy: 0.9696 - lr: 1.6000e-04
Epoch 28/50
744/744 [==============================] - 40s 54ms/step - loss: 0.0055 - accuracy: 0.9981 - val_loss: 0.2050 - val_accuracy: 0.9669 - lr: 1.6000e-04
Epoch 29/50
744/744 [==============================] - 41s 56ms/step - loss: 0.0055 - accuracy: 0.9981 - val_loss: 0.1849 - val_accuracy: 0.9694 - lr: 6.4000e-05
Epoch 30/50
744/744 [==============================] - 44s 59ms/step - loss: 0.0034 - accuracy: 0.9988 - val_loss: 0.1841 - val_accuracy: 0.9704 - lr: 6.4000e-05
Epoch 31/50
744/744 [==============================] - 41s 56ms/step - loss: 0.0045 - accuracy: 0.9985 - val_loss: 0.1799 - val_accuracy: 0.9704 - lr: 6.4000e-05
Epoch 32/50
744/744 [==============================] - 42s 56ms/step - loss: 0.0037 - accuracy: 0.9987 - val_loss: 0.1822 - val_accuracy: 0.9711 - lr: 6.4000e-05
Epoch 33/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0037 - accuracy: 0.9987 - val_loss: 0.1774 - val_accuracy: 0.9707 - lr: 6.4000e-05
Epoch 34/50
744/744 [==============================] - 41s 56ms/step - loss: 0.0049 - accuracy: 0.9984 - val_loss: 0.1813 - val_accuracy: 0.9713 - lr: 6.4000e-05
Epoch 35/50
744/744 [==============================] - 42s 56ms/step - loss: 0.0026 - accuracy: 0.9991 - val_loss: 0.1822 - val_accuracy: 0.9704 - lr: 2.5600e-05
Epoch 36/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0033 - accuracy: 0.9989 - val_loss: 0.1836 - val_accuracy: 0.9706 - lr: 2.5600e-05
Epoch 37/50
744/744 [==============================] - 42s 56ms/step - loss: 0.0028 - accuracy: 0.9989 - val_loss: 0.1854 - val_accuracy: 0.9704 - lr: 2.5600e-05
Epoch 38/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0031 - accuracy: 0.9989 - val_loss: 0.1830 - val_accuracy: 0.9694 - lr: 2.5600e-05
Epoch 39/50
744/744 [==============================] - 42s 56ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 0.1875 - val_accuracy: 0.9691 - lr: 2.5600e-05
Epoch 40/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0032 - accuracy: 0.9987 - val_loss: 0.1860 - val_accuracy: 0.9692 - lr: 1.0240e-05
Epoch 41/50
744/744 [==============================] - 45s 61ms/step - loss: 0.0026 - accuracy: 0.9991 - val_loss: 0.1829 - val_accuracy: 0.9691 - lr: 1.0240e-05
Epoch 42/50
744/744 [==============================] - 44s 59ms/step - loss: 0.0028 - accuracy: 0.9990 - val_loss: 0.1818 - val_accuracy: 0.9701 - lr: 1.0240e-05
Epoch 43/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0026 - accuracy: 0.9990 - val_loss: 0.1850 - val_accuracy: 0.9696 - lr: 1.0240e-05
Epoch 44/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0027 - accuracy: 0.9994 - val_loss: 0.1841 - val_accuracy: 0.9701 - lr: 4.0960e-06
Epoch 45/50
744/744 [==============================] - 44s 59ms/step - loss: 0.0035 - accuracy: 0.9989 - val_loss: 0.1833 - val_accuracy: 0.9701 - lr: 4.0960e-06
Epoch 46/50
744/744 [==============================] - 41s 56ms/step - loss: 0.0022 - accuracy: 0.9993 - val_loss: 0.1839 - val_accuracy: 0.9701 - lr: 4.0960e-06
Epoch 47/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0038 - accuracy: 0.9989 - val_loss: 0.1830 - val_accuracy: 0.9694 - lr: 4.0960e-06
Epoch 48/50
744/744 [==============================] - 44s 59ms/step - loss: 0.0027 - accuracy: 0.9992 - val_loss: 0.1828 - val_accuracy: 0.9697 - lr: 4.0960e-06
Epoch 49/50
744/744 [==============================] - 42s 56ms/step - loss: 0.0023 - accuracy: 0.9993 - val_loss: 0.1829 - val_accuracy: 0.9697 - lr: 4.0960e-06
Epoch 50/50
744/744 [==============================] - 41s 55ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 0.1826 - val_accuracy: 0.9702 - lr: 4.0960e-06
```

```
male_history = male_model.fit(x_trainM, y_trainM, batch_size=batch_size, epochs=n_epochs, validation_data=(x_testM, y_testM), callbacks=[rlrp])
```

```
Epoch 22/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0083 - accuracy: 0.9976 - val_loss: 0.3702 - val_accuracy: 0.9457 - lr: 6.4000e-05
Epoch 23/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0077 - accuracy: 0.9977 - val_loss: 0.3690 - val_accuracy: 0.9451 - lr: 2.5600e-05
Epoch 24/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0060 - accuracy: 0.9981 - val_loss: 0.3703 - val_accuracy: 0.9451 - lr: 2.5600e-05
Epoch 25/50
565/565 [==============================] - 31s 56ms/step - loss: 0.0075 - accuracy: 0.9976 - val_loss: 0.3738 - val_accuracy: 0.9457 - lr: 2.5600e-05
Epoch 26/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0062 - accuracy: 0.9982 - val_loss: 0.3737 - val_accuracy: 0.9444 - lr: 2.5600e-05
Epoch 27/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0081 - accuracy: 0.9977 - val_loss: 0.3846 - val_accuracy: 0.9442 - lr: 2.5600e-05
Epoch 28/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0075 - accuracy: 0.9976 - val_loss: 0.3839 - val_accuracy: 0.9442 - lr: 2.5600e-05
Epoch 29/50
565/565 [==============================] - 31s 55ms/step - loss: 0.0073 - accuracy: 0.9982 - val_loss: 0.3832 - val_accuracy: 0.9453 - lr: 1.0240e-05
Epoch 30/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0072 - accuracy: 0.9982 - val_loss: 0.3821 - val_accuracy: 0.9455 - lr: 1.0240e-05
Epoch 31/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0067 - accuracy: 0.9978 - val_loss: 0.3821 - val_accuracy: 0.9455 - lr: 1.0240e-05
Epoch 32/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0068 - accuracy: 0.9980 - val_loss: 0.3796 - val_accuracy: 0.9459 - lr: 1.0240e-05
Epoch 33/50
565/565 [==============================] - 31s 56ms/step - loss: 0.0063 - accuracy: 0.9986 - val_loss: 0.3791 - val_accuracy: 0.9459 - lr: 4.0960e-06
Epoch 34/50
565/565 [==============================] - 31s 56ms/step - loss: 0.0065 - accuracy: 0.9978 - val_loss: 0.3799 - val_accuracy: 0.9459 - lr: 4.0960e-06
Epoch 35/50
565/565 [==============================] - 31s 55ms/step - loss: 0.0058 - accuracy: 0.9988 - val_loss: 0.3815 - val_accuracy: 0.9459 - lr: 4.0960e-06
Epoch 36/50
565/565 [==============================] - 31s 56ms/step - loss: 0.0055 - accuracy: 0.9986 - val_loss: 0.3798 - val_accuracy: 0.9457 - lr: 4.0960e-06
Epoch 37/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0081 - accuracy: 0.9975 - val_loss: 0.3808 - val_accuracy: 0.9455 - lr: 4.0960e-06
Epoch 38/50
565/565 [==============================] - 35s 62ms/step - loss: 0.0073 - accuracy: 0.9977 - val_loss: 0.3804 - val_accuracy: 0.9462 - lr: 4.0960e-06
Epoch 39/50
565/565 [==============================] - 32s 57ms/step - loss: 0.0057 - accuracy: 0.9978 - val_loss: 0.3803 - val_accuracy: 0.9462 - lr: 4.0960e-06
Epoch 40/50
565/565 [==============================] - 34s 60ms/step - loss: 0.0075 - accuracy: 0.9979 - val_loss: 0.3790 - val_accuracy: 0.9462 - lr: 4.0960e-06
Epoch 41/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0065 - accuracy: 0.9982 - val_loss: 0.3791 - val_accuracy: 0.9462 - lr: 1.6384e-06
Epoch 42/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0065 - accuracy: 0.9978 - val_loss: 0.3796 - val_accuracy: 0.9464 - lr: 1.6384e-06
Epoch 43/50
565/565 [==============================] - 34s 61ms/step - loss: 0.0062 - accuracy: 0.9982 - val_loss: 0.3793 - val_accuracy: 0.9466 - lr: 1.6384e-06
Epoch 44/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0067 - accuracy: 0.9982 - val_loss: 0.3799 - val_accuracy: 0.9468 - lr: 1.6384e-06
Epoch 45/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0056 - accuracy: 0.9982 - val_loss: 0.3800 - val_accuracy: 0.9466 - lr: 1.0000e-06
Epoch 46/50
565/565 [==============================] - 32s 57ms/step - loss: 0.0064 - accuracy: 0.9981 - val_loss: 0.3799 - val_accuracy: 0.9468 - lr: 1.0000e-06
Epoch 47/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0059 - accuracy: 0.9982 - val_loss: 0.3800 - val_accuracy: 0.9468 - lr: 1.0000e-06
Epoch 48/50
565/565 [==============================] - 35s 63ms/step - loss: 0.0066 - accuracy: 0.9980 - val_loss: 0.3803 - val_accuracy: 0.9468 - lr: 1.0000e-06
Epoch 49/50
565/565 [==============================] - 32s 56ms/step - loss: 0.0059 - accuracy: 0.9982 - val_loss: 0.3805 - val_accuracy: 0.9466 - lr: 1.0000e-06
Epoch 50/50
565/565 [==============================] - 33s 58ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.3803 - val_accuracy: 0.9466 - lr: 1.0000e-06
```

```
# genderless
score = total_model.evaluate(x_train,y_train, verbose = 0)
print("Mixed-gender emotions training Accuracy: {0:.2%}".format(score[1]))

score = total_model.evaluate(x_test, y_test, verbose=0)
print("Mixed-gender emotions testing Accuracy: {0:.2%}".format(score[1]))
```

```
Mixed-gender emotions training Accuracy: 98.84%
Mixed-gender emotions testing Accuracy: 92.95%
```

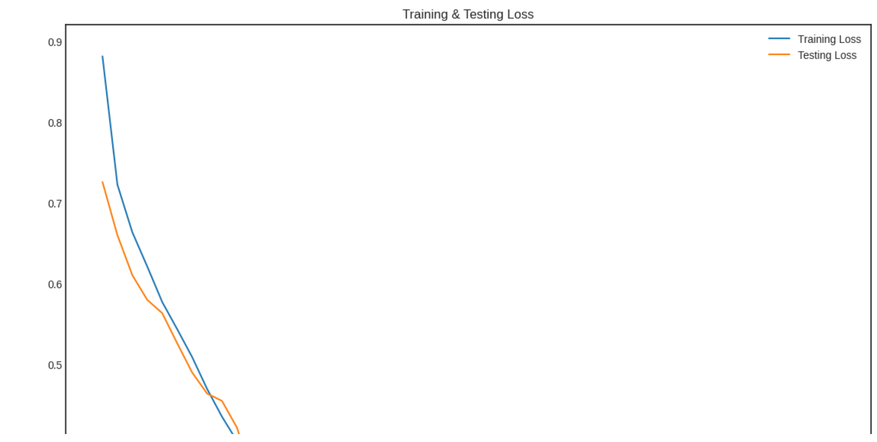
```
score = female_model.evaluate(x_trainF,y_trainF, verbose = 0)
print("Female emotions training Accuracy: {0:.2%}".format(score[1]))

score = female_model.evaluate(x_testF, y_testF, verbose=0)
print("Female emotions testing Accuracy: {0:.2%}".format(score[1]))
```

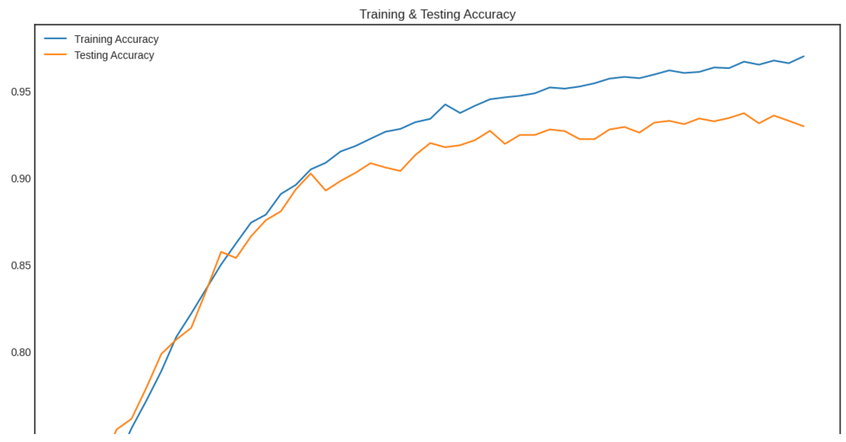
```
Female emotions training Accuracy: 99.98%
Female emotions testing Accuracy: 97.02%
```

```
score = male_model.evaluate(x_trainM,y_trainM, verbose = 0)
print("Male emotions training Accuracy: {0:.2%}".format(score[1]))

score = male_model.evaluate(x_testM, y_testM, verbose=0)
print("Male emotions testing Accuracy: {0:.2%}".format(score[1]))
```

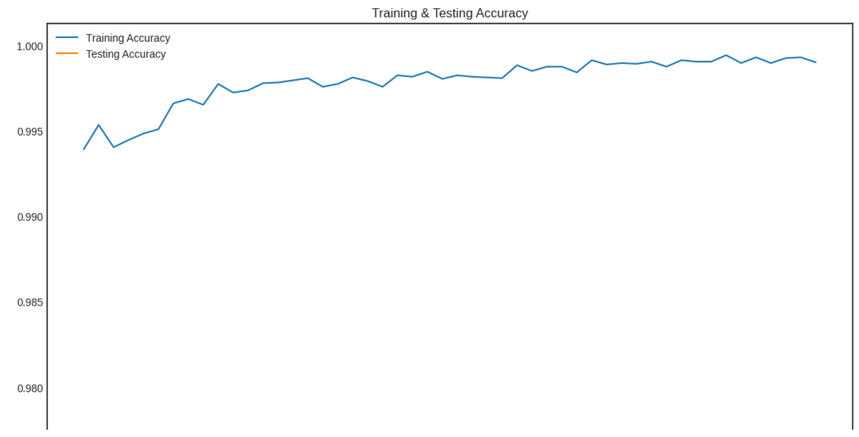
```
Male emotions training Accuracy: 100.00%
Male emotions testing Accuracy: 94.66%
```

```
show_graphs(history)
```

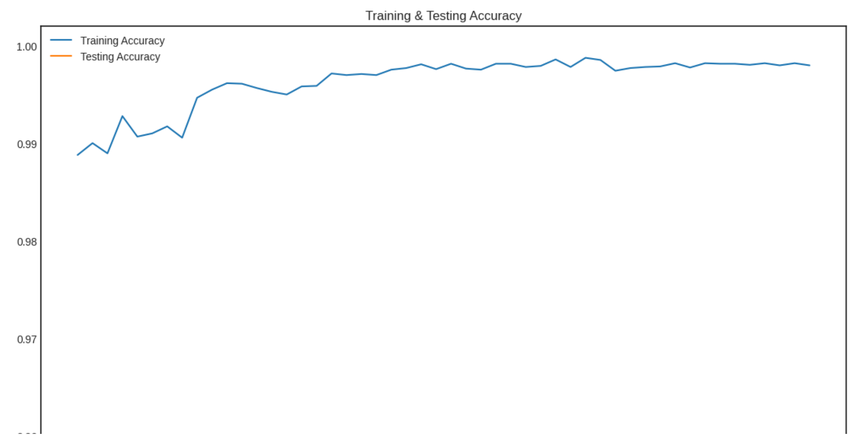





```
show_graphs(female_history)
```



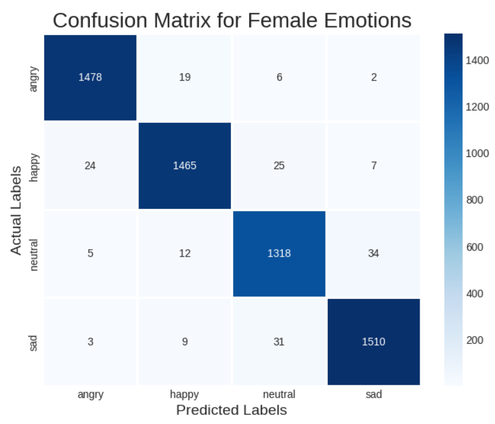

```
show_graphs(male_history)
```




```
# predicting on test data.
pred_test = female_model.predict(x_testF)
y_pred = encoder.inverse_transform(pred_test)
y_test_ = encoder.inverse_transform(y_testF)
```

```
186/186 [==============================] - 4s 19ms/step
```

```
cm = confusion_matrix(y_test_, y_pred)
plt.figure(figsize = (8, 6))
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 for Female Emotions', size=20)
plt.xlabel('Predicted Labels', size=14)
plt.ylabel('Actual Labels', size=14)
plt.show()
```




```
# predicting on test data.
pred_test = male_model.predict(x_testM)
y_pred = encoder.inverse_transform(pred_test)
y_test_ = encoder.inverse_transform(y_testM)
```

```
142/142 [==============================] - 3s 19ms/step
```

```
cm = confusion_matrix(y_test_, y_pred)
plt.figure(figsize = (8, 6))
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 for Male Emotions', size=20)
plt.xlabel('Predicted Labels', size=14)
plt.ylabel('Actual Labels', size=14)
plt.show()
```