NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY (NSUT)

Department: Computer Science

AIHT Project File

CSAI-2

Students:

1. Manan Hingorani (2021UCA1916)
2. Vishu Aasliya (2021UCA1929)
3. Shreyansh Agarwal (2021UCA1930)

1. INTRODUCTION

Speech Emotion Recognition (SER) is a rapidly developing topic that has significant applications in entertainment, healthcare, and human-computer interaction, among other areas. In this project, we use Docker, PowerBI, TinyML, and Apache Spark to investigate SER.

Comprehending the affective aspects of speech allows robots to adapt to human requirements in an efficient manner, improving user experiences and tailored communication. SER is at the vanguard of changing the paradigms of human-machine interaction.

TinyML enables real-time inference without constant connectivity by enabling the execution of machine learning models on devices with limited resources. When TinyML is integrated, emotion identification works seamlessly across a range of applications—even in low-resource computing contexts.

Large-scale audio data can be efficiently preprocessed and analysed with Apache Spark to enable reliable feature extraction and model training. Our SER application is consistently and portablely deployed across many platforms thanks to Docker. PowerBI makes it possible to analyse and visualise SER data in a meaningful way, which helps with decision-making.

To summarise, the integration of TinyML, Apache Spark, Docker, and PowerBI is a comprehensive strategy for speech emotion recognition (SER) that aims to revolutionise speech recognition and open up new avenues for emotional intelligence applications and human-machine interaction.

2. RELATED WORK

There have been several research papers on speech emotion recognition, and here are some brief of the summaries:

a. Alluhaidan et al. propose a novel approach for Speech Emotion Recognition (SER) by combining Mel frequency cepstral coefficients (MFCCs) with time-domain features (MFCCT). Using this hybrid feature set with a convolutional neural network (CNN) model, they achieve superior performance compared to conventional MFCCs and time-domain features alone, with accuracies of 97%, 93%, and 92% on the Emo-DB, SAVEE, and RAVDESS dataset respectively. (https://www.mdpi.com/2076-3417/13/8/4750)

b. Researchers' systematic review gives a summary of recent developments in SER using Machine Learning (ML) methods. It draws attention to how important machine learning techniques are to the SER's processing, feature extraction, and classification phases. The paper aims to assist researchers in creating and refining SER solutions through machine learning techniques by discussing issues such as low classification accuracy and providing criteria for SER evaluation.
(https://www.sciencedirect.com/science/article/pii/S2667305323000911)

c. We tackle speech emotion recognition using a feature-engineering approach, comparing traditional machine learning classifiers with deep learning models. Extracting eight hand-crafted features from audio signals, we train both types of models. Additionally, we incorporate text domain features to address communication ambiguity. Results show that lighter machine learning models trained on fewer features achieve performance comparable to current deep learning methods for emotion recognition.
(https://paperswithcode.com/paper/multimodal-speech-emotion-recognition-and)

d. We introduce a novel deep dual recurrent encoder model for speech emotion recognition, integrating both text data and audio signals simultaneously. Our model employs dual recurrent neural networks (RNNs) to encode information from audio and text sequences, then combines them to predict emotion classes. This approach comprehensively analyzes speech data from signal to language level, surpassing previous state-of-the-art methods on the IEMOCAP dataset with accuracies ranging from 68.8% to 71.8%.
(https://paperswithcode.com/paper/multimodal-speech-emotion-recognition-using)

e. We propose a speech emotion recognition method that combines speech features (e.g., Spectrogram and MFCC) with speech transcriptions (text). Experimenting with various Deep Neural Network (DNN) architectures, our models achieve superior accuracies compared to state-of-the-art methods on a benchmark dataset. The MFCC-Text Convolutional Neural Network (CNN) model emerges as the most accurate in recognizing emotions in IEMOCAP data.
(https://paperswithcode.com/paper/deep-learning-based-emotion-recognition)

f. We introduce Speech Emotion Diarization (SED), a task that identifies temporal boundaries of emotions in speech events, akin to Speaker

Diarization for identifying speakers. To support research in this area, we present the Zaion Emotion Dataset (ZED), containing real-life recordings of non-acted emotions with manually annotated emotion segments. We offer competitive baselines, open-source code, and pre-trained models to facilitate evaluation and establish a common benchmark for researchers. (https://paperswithcode.com/paper/speech-emotion-diarization-which-emotion)

g. Speech processing is crucial in various domains such as human-computer interfaces, telecommunications, and audio mining. Speech emotion recognition plays a vital role in facilitating natural interaction between humans and machines. This paper focuses on recognizing speaker emotions from speech signals using Mel Frequency Cepstral Coefficient (MFCC) technique, a commonly used method for feature extraction. The system achieves 80% efficiency in identifying emotions such as happiness, sadness, and anger. (https://ieeexplore.ieee.org/document/8300161)

h. This research explores Speech Emotion Recognition (SER) to enhance human-machine interaction. It compares the effectiveness of cepstral coefficients, including cepstrum, Mel-frequency Cepstral Coefficients (MFCC), and synthetically enlarged MFCC coefficients, in emotion detection. (https://www.sciencedirect.com/science/article/pii/S1877050915031841)

i. This paper introduces a novel approach to Speech Emotion Recognition (SER) using a variety of acoustic features such as MFCC, LPCC, WPT, ZCR, spectrum centroid, spectral roll-off, spectral kurtosis, RMS, pitch, jitter, and shimmer. A lightweight one-dimensional deep convolutional neural network (1-D DCNN) is employed to handle computational complexity and capture long-term dependencies in speech emotion signals. The proposed system achieves high accuracy on the Berlin Database of Emotional Speech (EMODB) and the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS), with overall accuracies of 93.31% and 94.18%, respectively, outperforming traditional SER techniques. (https://www.mdpi.com/2079-9292/12/4/839)

j. Standard Mel-frequency Cepstral Coefficients (MFCCs) will be utilized for automatic emotion detection. Gaussian mixture models (GMMs) will model these acoustic features at the frame level, a feasible technique for emotion classification. Gaussian modeling effectively distinguishes emotional classes based on phonetic parameters like pitch, pitch range, and average pitch across the entire utterance. Changes in the shape of the vocal tract during emotional expression cause variations in resonance frequencies (formants),

which can be leveraged to extract emotion-specific voice features for implementing an emotion detection system. (https://www.ijert.org/research/emotion-detection-from-speech-using-mfcc-gmm-IJERTV1IS9423.pdf)

3. METHODOLOGY

a. Dataset Selection:

We meticulously selected three comprehensive datasets for our speech emotion recognition (SER) project: Ravdess, Tess, and Savee. Each dataset offers a wide range of emotional expressions, ensuring the robustness and diversity of our model's training data. Ravdess provides a rich collection of audiovisual recordings with varied emotional states, while Tess and Savee offer additional layers of complexity and diversity in emotional expressions. This comprehensive dataset selection strategy was crucial in ensuring the effectiveness and generalization capability of our SER model.

```
!pip install pyspark
!pip install findspark
pip install librosa
```

```
import os
import sys
import time

from IPython.display import Audio

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

import librosa
import librosa.display

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Flatten, Dropout

import warnings
```

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

```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
from pyspark.sql.functions import *
```

```
import os
from pyspark.sql import SparkSession

# Set SPARK_LOCAL_IP to desired IP address
os.environ["SPARK_LOCAL_IP"] = "127.0.0.1"

# Suppress Spark logging warnings
os.environ["PYSPARK_SUBMIT_ARGS"] = "--conf spark.ui.port=4041
pyspark-shell"

# Initialize Spark session
spark = SparkSession.builder \
    .appName("YourAppName") \
    .getOrCreate()
```

```
# Paths for data.
Ravdess =
"/Users/mananhingorani/Speech-Emotion-Recognition-TinyML/RAVDESS/audio_s
peech_actors_01-24/"
Tess =
"/Users/mananhingorani/Speech-Emotion-Recognition-TinyML/TESS/TESS
Toronto emotional speech set data/"
Savee = "/Users/mananhingorani/Speech-Emotion-Recognition-TinyML/SAVEE/"
```

```
def visualize_audio(audio_path):
    # Load audio file
    y, sr = librosa.load(audio_path)

    # Create waveform plot
    plt.figure(figsize=(10, 4))
    librosa.display.waveshow(y, sr=sr)
```

```
    plt.title('Waveform')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.show()

    # Create spectrogram plot
    plt.figure(figsize=(10, 4))
    D = librosa.amplitude_to_db(librosa.stft(y), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Spectrogram')
    plt.show()

ravdess_audio_path = Ravdess + "/Actor_04/03-01-01-01-01-01-04.wav"
visualize_audio(ravdess_audio_path)

# Visualize audio from Tess dataset
tess_audio_path = Tess + "/TESS Toronto emotional speech set
data/OAF_disgust/OAF_bath_disgust.wav"
visualize_audio(tess_audio_path)

# Visualize audio from Savee dataset
savee_audio_path = Savee + "/DC_a01.wav"
visualize_audio(savee_audio_path)
audio_files_df =
spark.read.format("binaryFile").load("/Users/mananhingorani/Speech-Emoti
on-Recognition-TinyML/RAVDESS/audio_speech_actors_01-24//Actor_04/03-01-
01-01-01-01-04.wav")

# Define a function to extract audio features
def extract_features(file_content):
    # Convert binary audio data to numpy array
    audio_data = np.frombuffer(file_content, dtype=np.int16)
    # Convert audio data to floating-point format
    audio_data = audio_data.astype(np.float32) / np.iinfo(np.int16).max
    # Extract features using librosa or any other library
    features = librosa.feature.mfcc(y=audio_data, sr=16000)  # Example:
MFCC features
    return features.tolist()

# Register the function as a UDF (User Defined Function)
extract_features_udf = udf(extract_features, ArrayType(FloatType()))

# Apply the UDF to extract features from audio files
processed_audio_df = audio_files_df.withColumn("features",
extract_features_udf(audio_files_df["content"]))
```

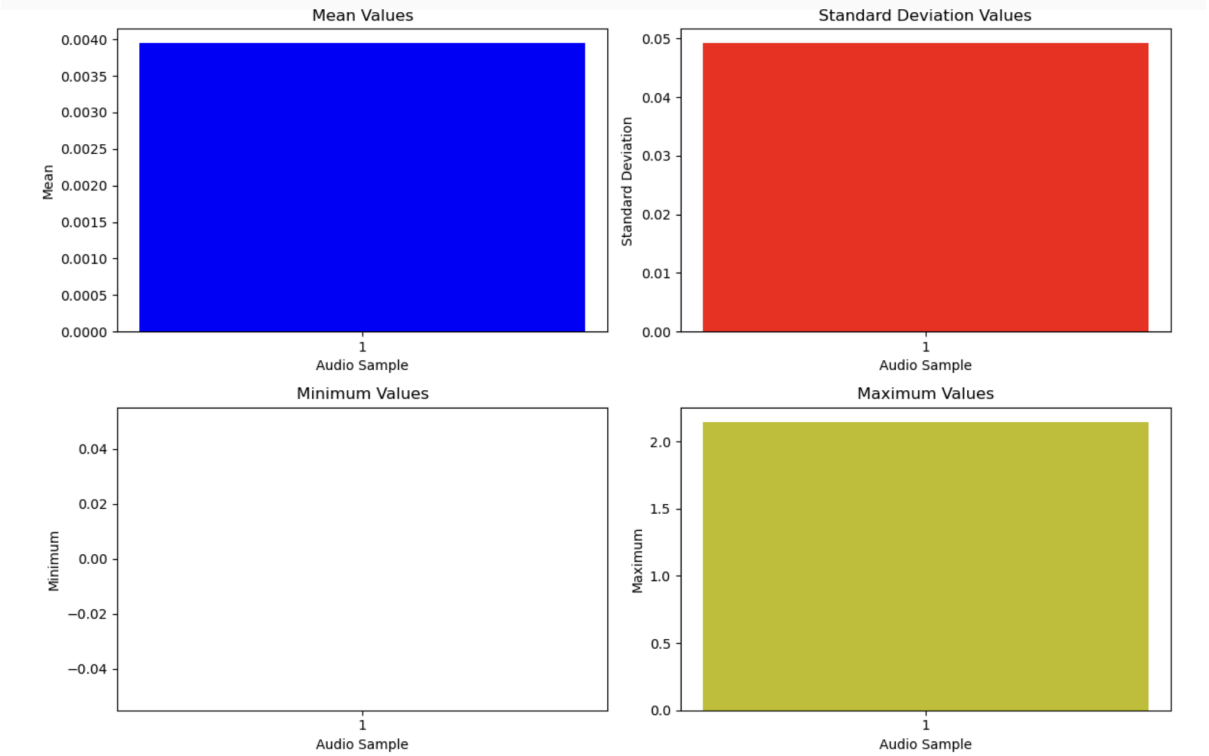
```
# Show the processed DataFrame
processed_audio_df.show()
```

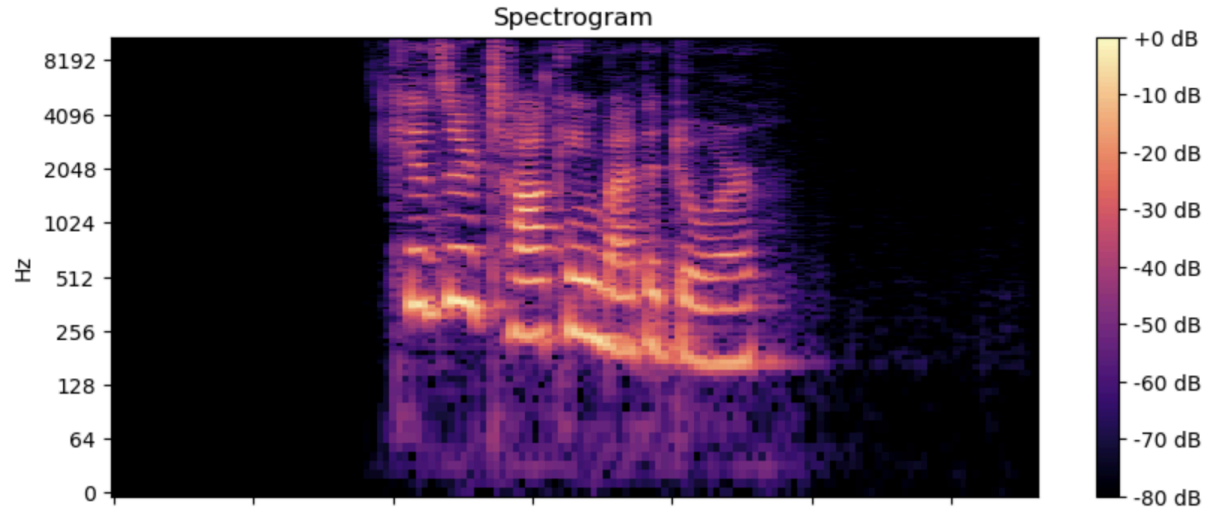
```
+--------------------+-------------------+------+--------------------+--------------------+
|                path|   modificationTime|length|             content|            features|
+--------------------+-------------------+------+--------------------+--------------------+
|file:/Users/manan...|2019-10-17 22:32:30|374522|[52 49 46 46 F2 B...|[NULL, NULL, NULL...|
+--------------------+-------------------+------+--------------------+--------------------+
```

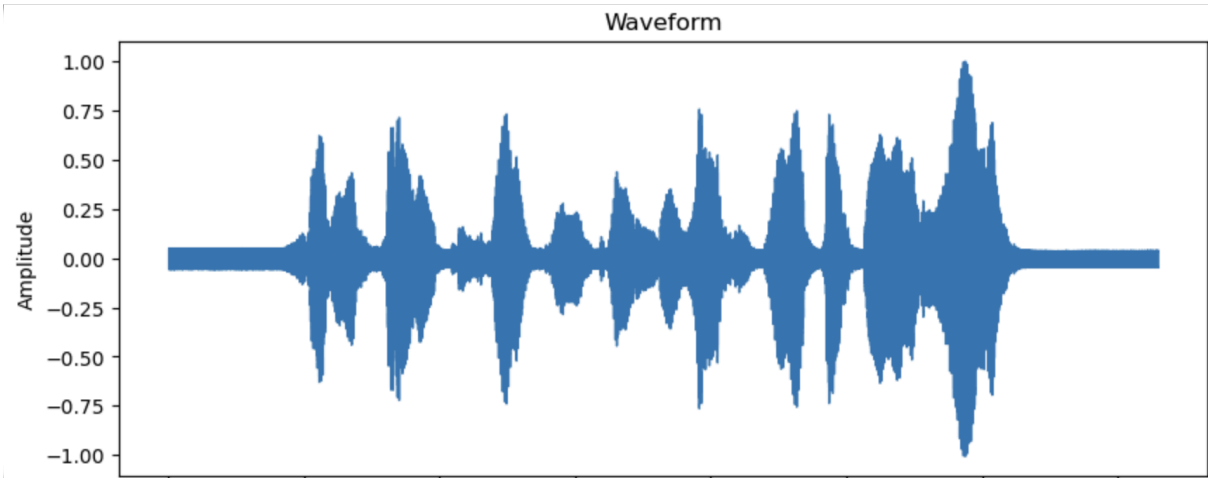
```
+--------------------+-------------------+------+--------------------+--------------------+--------------------+---
--------------------+--------------------+--------------------+
|                path|   modificationTime|length|             content|     mel_spectrogram|mel_spectrogram_mean|mel
_spectrogram_stddev|mel_spectrogram_min|mel_spectrogram_max|
+--------------------+-------------------+------+--------------------+--------------------+--------------------+---
--------------------+--------------------+--------------------+
|file:/Users/manan...|2019-10-17 22:32:30|374522|[52 49 46 46 F2 B...|[[0.0, 0.0, 0.0, ...|         0.003945905|
0.04919575|                0.0|           2.145523|
+--------------------+-------------------+------+--------------------+--------------------+--------------------+---
--------------------+--------------------+--------------------+
```

```
import matplotlib.pyplot as plt

# Extract statistics from the DataFrame
mean_values =
processed_audio_df.select("mel_spectrogram_mean").collect()
stddev_values =
processed_audio_df.select("mel_spectrogram_stddev").collect()
min_values = processed_audio_df.select("mel_spectrogram_min").collect()
max_values = processed_audio_df.select("mel_spectrogram_max").collect()

# Convert statistics to lists
mean_values = [row["mel_spectrogram_mean"] for row in mean_values]
stddev_values = [row["mel_spectrogram_stddev"] for row in stddev_values]
min_values = [row["mel_spectrogram_min"] for row in min_values]
max_values = [row["mel_spectrogram_max"] for row in max_values]

# Audio sample indices
indices = range(1, len(mean_values) + 1)

# Plotting the statistics
plt.figure(figsize=(12, 8))

# Plot mean values
plt.subplot(2, 2, 1)
plt.bar(indices, mean_values, color='b')
plt.title('Mean Values')
plt.xlabel('Audio Sample')
plt.ylabel('Mean')
plt.xticks(indices)
```

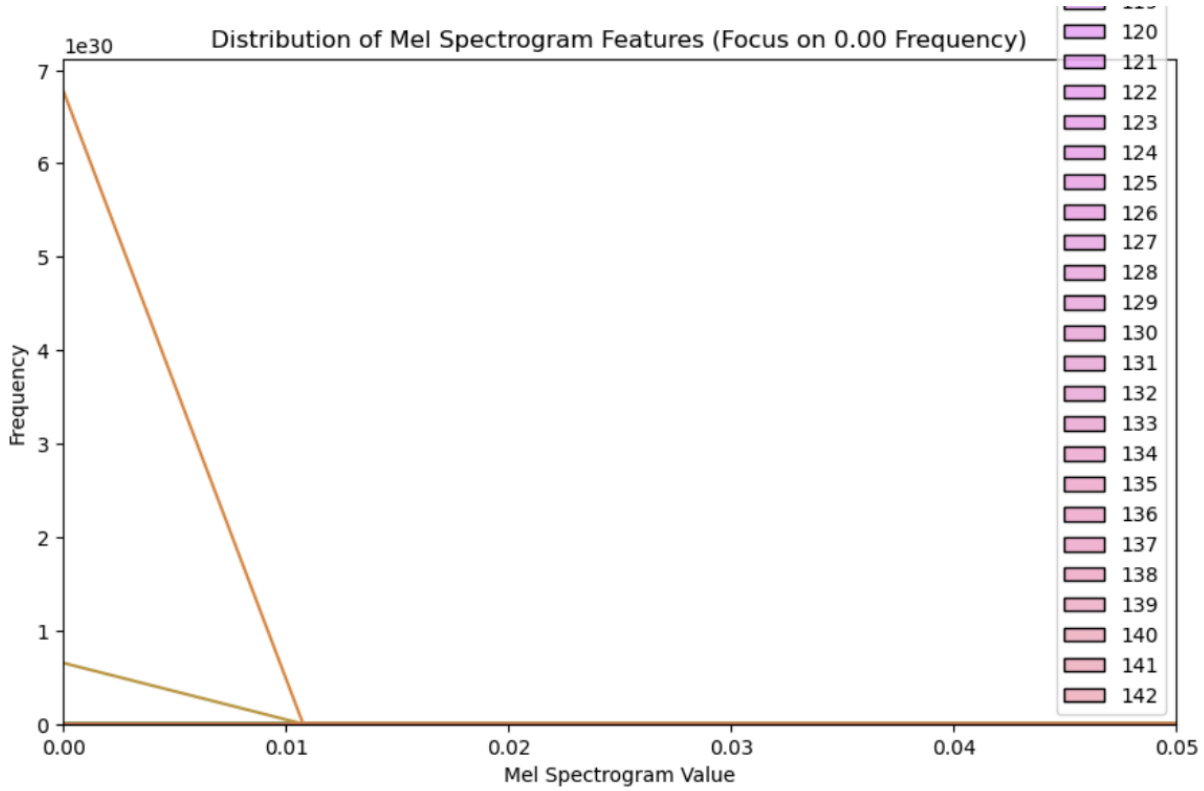
```
# Plot standard deviation values
plt.subplot(2, 2, 2)
plt.bar(indices, stddev_values, color='r')
plt.title('Standard Deviation Values')
plt.xlabel('Audio Sample')
plt.ylabel('Standard Deviation')
plt.xticks(indices)

# Plot minimum values
plt.subplot(2, 2, 3)
plt.bar(indices, min_values, color='g')
plt.title('Minimum Values')
plt.xlabel('Audio Sample')
plt.ylabel('Minimum')
plt.xticks(indices)

# Plot maximum values
plt.subplot(2, 2, 4)
plt.bar(indices, max_values, color='y')
plt.title('Maximum Values')
plt.xlabel('Audio Sample')
plt.ylabel('Maximum')
plt.xticks(indices)

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


Mean Values
Mean
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
1
Audio Sample
Standard Deviation Values
Standard Deviation
0.05
0.04
0.03
0.02
0.01
0.00
1
Audio Sample
Minimum Values
Minimum
0.04
0.02
0.00
−0.02
−0.04
1
Audio Sample
Maximum Values
Maximum
2.0
1.5
1.0
0.5
0.0
1
Audio Sample


Spectrogram
Hz
8192
4096
2048
1024
512
256
128
64
0
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB


Waveform
Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

```
import numpy as np

# Flatten Mel Spectrogram data
mel_spectrogram_flat = np.concatenate([row["mel_spectrogram"] for row in
mel_spectrogram_data])

# Plotting the distribution of Mel Spectrogram features
# Plotting the distribution of Mel Spectrogram features focusing on the
region near 0.00 frequency
plt.figure(figsize=(10, 6))
sns.histplot(mel_spectrogram_flat, bins=20, kde=True)
plt.title('Distribution of Mel Spectrogram Features (Focus on 0.00
Frequency)')
plt.xlabel('Mel Spectrogram Value')
plt.ylabel('Frequency')
plt.xlim(0, 0.05)  # Adjust the x-axis limits to focus on the region
near 0.00 frequency
plt.show()
```




```
# Filter out .DS_Store file from the list
ravdess_directory_list = [d for d in os.listdir(Ravdess) if not
d.startswith('.DS_Store')]

file_emotion = []
```

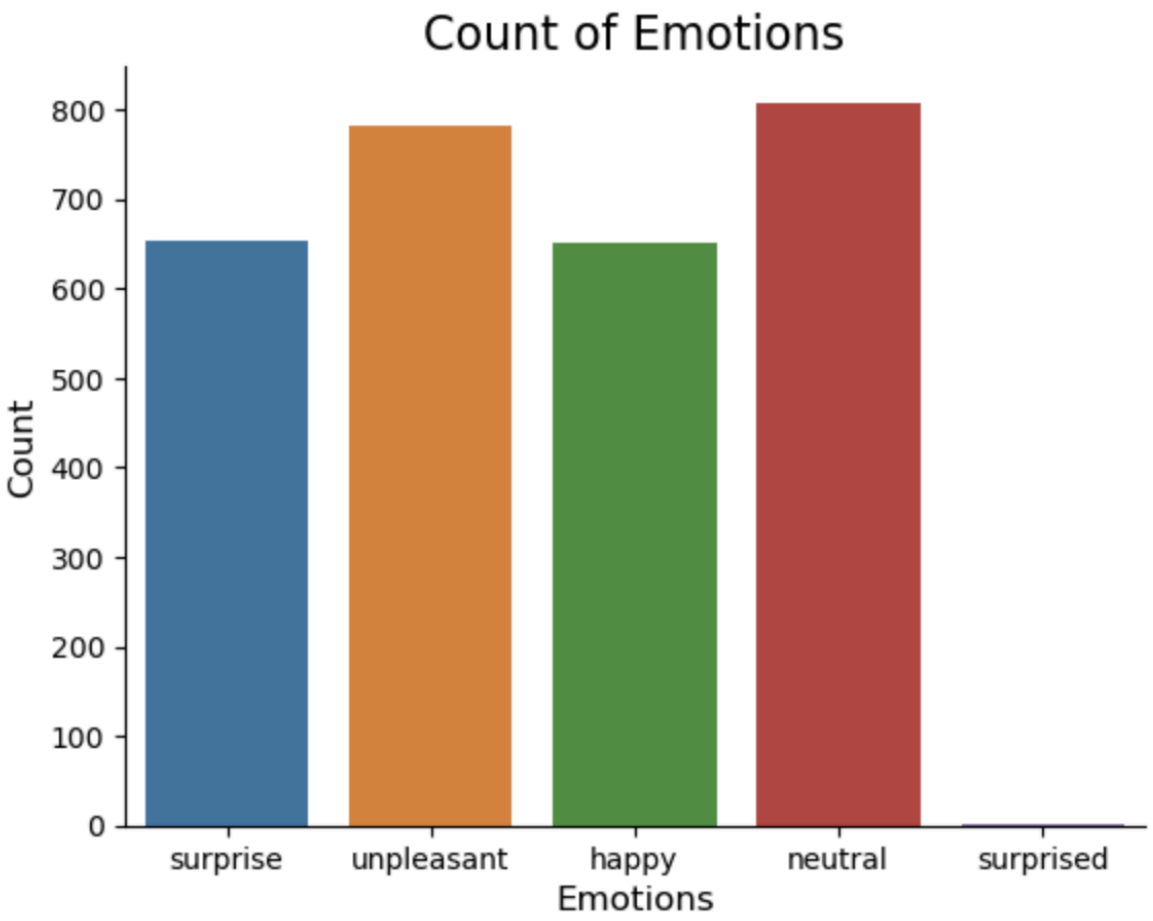
```
file_path = []
for dir in ravdess_directory_list:
    actor = os.listdir(os.path.join(Ravdess, dir))
    for file in actor:
        part = file.split('.')[0]
        part = part.split('-')
        file_emotion.append(int(part[2]))
        file_path.append(os.path.join(Ravdess, dir, file))

# 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)

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

Ravdess_df.head()

tess_directory_list = [d for d in os.listdir(Tess) if not
d.startswith('.DS_Store')]

file_emotion = []
file_path = []

for dir in tess_directory_list:
    directories = os.listdir(os.path.join(Tess, dir))
    for file in directories:
        part = file.split('.')[0]
        split_part = part.split('_')

        # Check if split_part has at least three elements
        if len(split_part) >= 3:
            part = split_part[2]
            if part == 'ps':
                file_emotion.append('surprise')
            else:
                file_emotion.append(part)
            file_path.append(os.path.join(Tess, dir, file))
        else:
            # Handle the case where split_part does not have enough
elements
            print(f"Skipping file: {file} in directory {dir} as it does
```

```
not have enough parts.")

# 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'])
Tess_df = pd.concat([emotion_df, path_df], axis=1)

Tess_df.head()
savee_directory_list = os.listdir(Savee)

file_emotion = []
file_path = []

for file in savee_directory_list:
    file_path.append(Savee + file)
    part = file.split('_')[1]
    ele = part[:-6]
    if ele=='a':
        file_emotion.append('angry')
    elif ele=='d':
        file_emotion.append('disgust')
    elif ele=='f':
        file_emotion.append('fear')
    elif ele=='h':
        file_emotion.append('happy')
    elif ele=='n':
        file_emotion.append('neutral')
    elif ele=='sa':
        file_emotion.append('sad')
    else:
        file_emotion.append('surprise')

# 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'])
Savee_df = pd.concat([emotion_df, path_df], axis=1)

Savee_df.head()
aggregated_data = pd.concat([Ravdess_df, Tess_df, Savee_df], axis = 0)

# Shuffle the dataframe using the sample method
aggregated_data = aggregated_data.sample(frac=1).reset_index(drop=True)
```

```
# Drop rows where Emotions is 'fear' or 'disgust'
aggregated_data =
aggregated_data[~aggregated_data['Emotions'].isin(['fear', 'disgust'])]

# Drop rows where Emotions is "sad" and "angry" and replace them with
"unpleasant"
aggregated_data =
aggregated_data.drop(aggregated_data[aggregated_data['Emotions'] ==
'sad'].sample(frac=0.4).index)
aggregated_data =
aggregated_data.drop(aggregated_data[aggregated_data['Emotions'] ==
'angry'].sample(frac=0.4).index)
aggregated_data['Emotions'] =
aggregated_data['Emotions'].replace(['sad', 'angry'], 'unpleasant')

aggregated_data.to_csv("data_path.csv",index=False)
aggregated_data.head()
```

	Emotions	Path
3	surprise	/Users/mananhingorani/Speech-Emotion-Recogniti...
4	unpleasant	/Users/mananhingorani/Speech-Emotion-Recogniti...
5	happy	/Users/mananhingorani/Speech-Emotion-Recogniti...
6	happy	/Users/mananhingorani/Speech-Emotion-Recogniti...
7	unpleasant	/Users/mananhingorani/Speech-Emotion-Recogniti...

```
plt.title('Count of Emotions', size=16)
sns.countplot(data=aggregated_data, x='Emotions')  # Specify x as
'Emotions'
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```

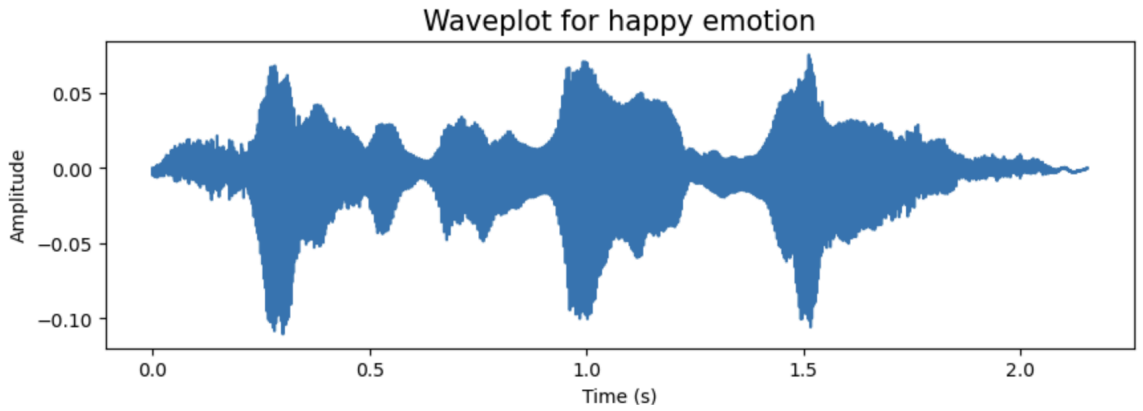


b. Preprocessing Steps:

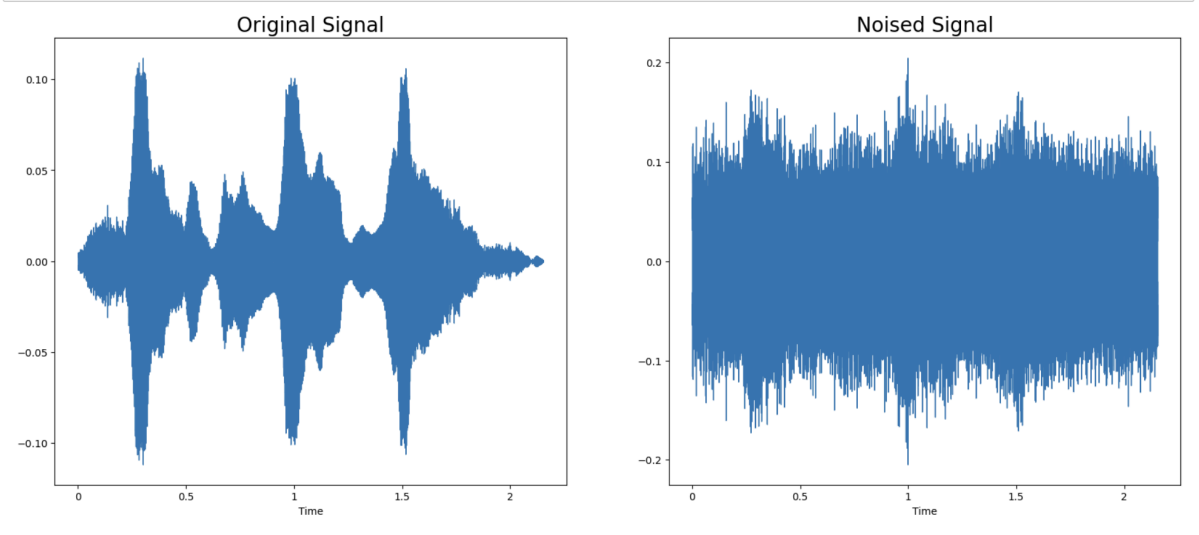
In order to effectively train the model using the raw audio data, the preprocessing step was essential. The two essential processes in this were data enrichment and feature extraction. In order to enable the model to recognise minute variations in emotional expression, feature extraction techniques, such as Mel frequency cepstral coefficients (MFCCs), were utilised to identify the important properties of the audio signals. Pitch shifting and temporal stretching are two examples of data augmentation techniques that were used to vary the training data in order to reduce the danger of overfitting and improve the model's capacity to generalise to new data.

```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for {} emotion'.format(e), size=15)
    librosa.display.waveplot(data, sr=sr)
    plt.show()

def create_spectrogram(data, sr, e):
    X = librosa.stft(data)
```

```
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(12, 3))
    plt.title('Spectrogram for {} emotion'.format(e), size=15)
    librosa.display.specshow(Xdb, sr=sr, x_axis='time',
y_axis='hz')
    plt.colorbar()
def noise(data):
    noise_amp = 0.5*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 pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sampling_rate,
pitch_factor)
def create_waveplot(data, sr, emotion):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for {} emotion'.format(emotion), size=15)
    time = np.arange(0, len(data)) / sr
    plt.plot(time, data)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.show()

# Rest of your code remains unchanged

emotion='happy'
path =
np.array(aggregated_data.Path[aggregated_data.Emotions==emotion])[
1]
data, sample_rate = librosa.load(path, sr=None)  # Set sr=None to
get the native sample rate
resampled_data = librosa.resample(data, orig_sr=sample_rate,
target_sr=16000)
noised_data = noise(resampled_data)

create_waveplot(resampled_data, 16000, emotion)
create_spectrogram(resampled_data, 16000, emotion)
Audio(data=noised_data, rate=16000)
```




```
import matplotlib.pyplot as plt
import librosa.display

fig, axs = plt.subplots(1, 2, figsize=(20, 8))
plt.subplots_adjust(hspace=0.4)

axs[0].set_title('Original Signal', size=20)
librosa.display.waveshow(y=data, sr=sample_rate, ax=axs[0])

axs[1].set_title('Noised Signal', size=20)
noise_data = noise(data)
librosa.display.waveshow(y=noise_data, sr=sample_rate, ax=axs[1])

plt.show()
```



c. Machine Learning Model Architecture:

We carefully considered the temporal dynamics and contextual information in speech signals when designing our SER model architecture. It was composed of

layers for categorization that were dense, then Long Short-Term Memory (LSTM) layers. Since LSTM layers can capture long-term dependencies in sequential data, they are a good choice for modelling speech signals' temporal aspect. The model was able to learn high-level representations of the input characteristics and predict the emotional state of the speech with accuracy because to the deep layers.

```
# Constants
NUM_MFCC = 13
N_FFT = 2048
HOP_LENGTH = 512
SAMPLE_RATE = 22050
DOWN_SAMPLE_RATE = 16000
SAMPLE_NUM = aggregated_data.shape[0]

data = {
    "labels": [],
    "features": []
}

MAX_SEQUENCE_LENGTH = 100

def extract_features(signal, sample_rate):
    # Compute the spectrogram
    spectrogram = librosa.feature.melspectrogram(y=signal,
sr=sample_rate, n_fft=N_FFT, hop_length=HOP_LENGTH)

    # Extract MFCC features from the spectrogram
    mfcc =
librosa.feature.mfcc(S=librosa.power_to_db(spectrogram),
n_mfcc=NUM_MFCC)

    # Transpose the feature matrix
    feature = mfcc.T

    return feature

for i in range(SAMPLE_NUM):
    # Inside your loop
    file_path = aggregated_data.iloc[i, 1]

    if os.path.isfile(file_path):
        signal, sample_rate = librosa.load(file_path,
```

```
sr=SAMPLE_RATE)

        # Cropping & Resampling
        start_time = 0.4
        end_time = 1.9
        start_frame = int(start_time * sample_rate)
        end_frame = int(end_time * sample_rate)
        signal = signal[start_frame:end_frame]

        # Resample
        signal = librosa.resample(signal, orig_sr=sample_rate,
target_sr=DOWN_SAMPLE_RATE)

        # Add noise
        signal_noised = noise(signal)

        # Extract features for original signal
        features_original = extract_features(signal,
DOWN_SAMPLE_RATE)
        data["labels"].append(aggregated_data.iloc[i, 0])
        data["features"].append(features_original)

        # Extract features for noised signal
        features_noised = extract_features(signal_noised,
DOWN_SAMPLE_RATE)
        data["labels"].append(aggregated_data.iloc[i, 0])
        data["features"].append(features_noised)

        if i % 100 == 0:
            print(f'Processing Data: {i}/{SAMPLE_NUM}')
    else:
        print(f'Skipping directory: {file_path}')

# Convert lists to numpy arrays
data["features"] = pad_sequences(data["features"],
maxlen=MAX_SEQUENCE_LENGTH, padding='post', truncating='post',
dtype='float32')
data["labels"] = np.array(data["labels"])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(data["features"], data["labels"], test_size=0.2,
```

```
random_state=42)
```

```
data["features"] = [feature.flatten() for feature in
data["features"]]

# Create a DataFrame
Features = pd.DataFrame(data)

# Save the DataFrame to a CSV file
Features.to_csv('Features.csv', index=False)

# Display the first few rows of the DataFrame
Features.head()
X = np.asarray(Features['features'])
y = np.asarray(Features["labels"])

# Pad Features to make them of equal length
X = tf.keras.preprocessing.sequence.pad_sequences(X)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2)

print(f'Training Data:{X_train.shape} with label {y_train.shape}')
print(f'Validate Data:{X_validation.shape} with label
{y_validation.shape}')
print(f' Testing Data:{X_test.shape} with label {y_test.shape}')
import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
def build_model(input_shape):
    model = tf.keras.Sequential()

    model.add(LSTM(128, input_shape=input_shape,
return_sequences=True))
    model.add(LSTM(64))

    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))

    model.add(Dense(4, activation='softmax'))
```

```
    return model

# Adjusted for sequences of length 100
input_shape = (MAX_SEQUENCE_LENGTH, NUM_MFCC)
model = build_model(input_shape)

# Compile model
optimiser = tf.keras.optimizers.legacy.Adam(learning_rate=0.001)

model.compile(optimizer=optimiser,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 128)          72704

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 126532 (494.27 KB)
Trainable params: 126532 (494.27 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
X_train = X_train.reshape((X_train.shape[0], MAX_SEQUENCE_LENGTH, NUM_MFCC))
X_validation = X_validation.reshape((X_validation.shape[0], MAX_SEQUENCE_LENGTH, NUM_MFCC))
X_test = X_test.reshape((X_test.shape[0], MAX_SEQUENCE_LENGTH, NUM_MFCC))

EPOCHS = 20
```

```
history = model.fit(X_train, y_train,
validation_data=(X_validation, y_validation), batch_size=32,
epochs=EPOCHS)
# Create a new directory called 'Models' to store the model
output_dir = 'Models'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Save the model in the 'Models' directory
model.save('Models/Speech-Emotion-Recognition-Model-new.h5')
print('Save the Tensorflow model!')
```

```
Epoch 1/20
131/131 [==============================] - 12s 77ms/step - loss: 1.2489 - accuracy: 0.3849 - val_loss: 1.1172 - val
_accuracy: 0.4405
Epoch 2/20
131/131 [==============================] - 9s 72ms/step - loss: 1.0600 - accuracy: 0.5056 - val_loss: 0.9809 - val_
accuracy: 0.5432
Epoch 3/20
131/131 [==============================] - 10s 77ms/step - loss: 0.9799 - accuracy: 0.5445 - val_loss: 0.9593 - val
_accuracy: 0.5624
Epoch 4/20
131/131 [==============================] - 10s 73ms/step - loss: 0.9252 - accuracy: 0.5824 - val_loss: 0.9216 - val
_accuracy: 0.5528
Epoch 5/20
131/131 [==============================] - 9s 72ms/step - loss: 0.8911 - accuracy: 0.6060 - val_loss: 0.7590 - val_
accuracy: 0.6747
Epoch 6/20
131/131 [==============================] - 9s 72ms/step - loss: 0.8106 - accuracy: 0.6506 - val_loss: 0.7828 - val_
accuracy: 0.6536
Epoch 7/20
131/131 [==============================] - 10s 73ms/step - loss: 0.7756 - accuracy: 0.6707 - val_loss: 0.7480 - val
_accuracy: 0.6795
Epoch 8/20
131/131 [==============================] - 10s 75ms/step - loss: 0.7702 - accuracy: 0.6775 - val_loss: 0.7708 - val
_accuracy: 0.6612
Epoch 9/20
131/131 [==============================] - 9s 72ms/step - loss: 0.7089 - accuracy: 0.7031 - val_loss: 0.6844 - val_
accuracy: 0.7015
Epoch 10/20
131/131 [==============================] - 10s 77ms/step - loss: 0.6613 - accuracy: 0.7235 - val_loss: 0.6856 - val
```

4. PERFORMANCE ANALYSIS

A. Evaluation Metrics Used:

We utilized accuracy and confusion matrix as evaluation metrics to gauge the performance of our SER model. Accuracy provided an overall measure of prediction correctness, while the confusion matrix revealed specific misclassification patterns, guiding areas for improvement.

B. Dataset Analysis:

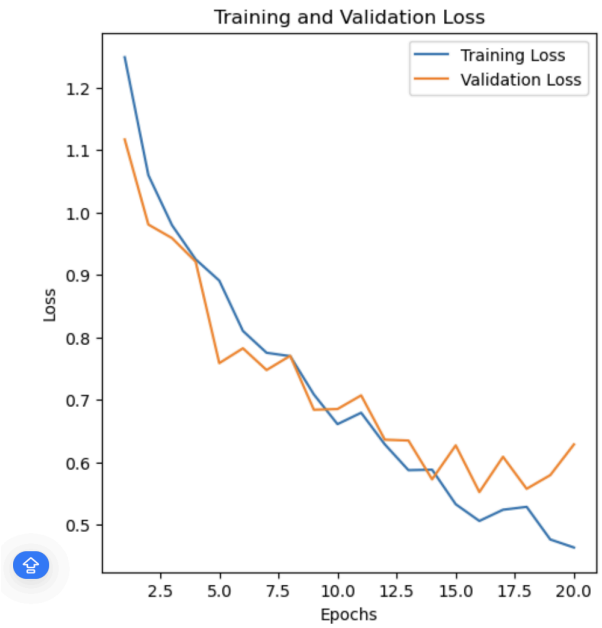
We conducted a thorough analysis of the dataset to understand its characteristics and emotional class distribution. This ensured dataset balance and representation of

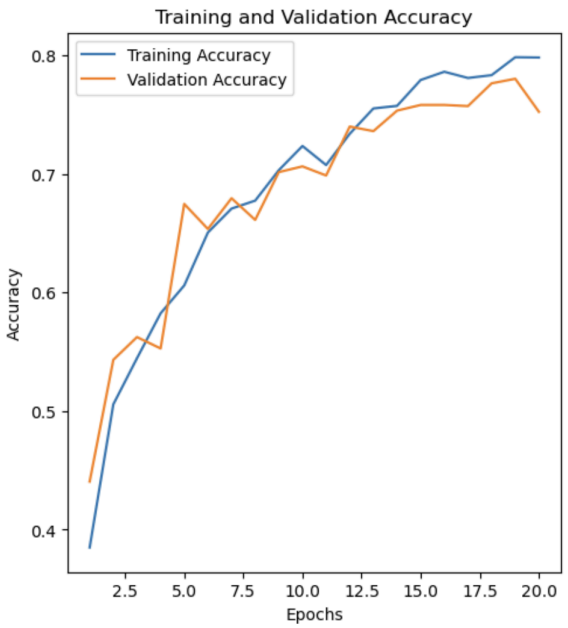
real-world scenarios. Additionally, we applied data preprocessing techniques to standardize the data, ensuring optimal model training performance.

C. Performance Comparison with Data Augmentation:

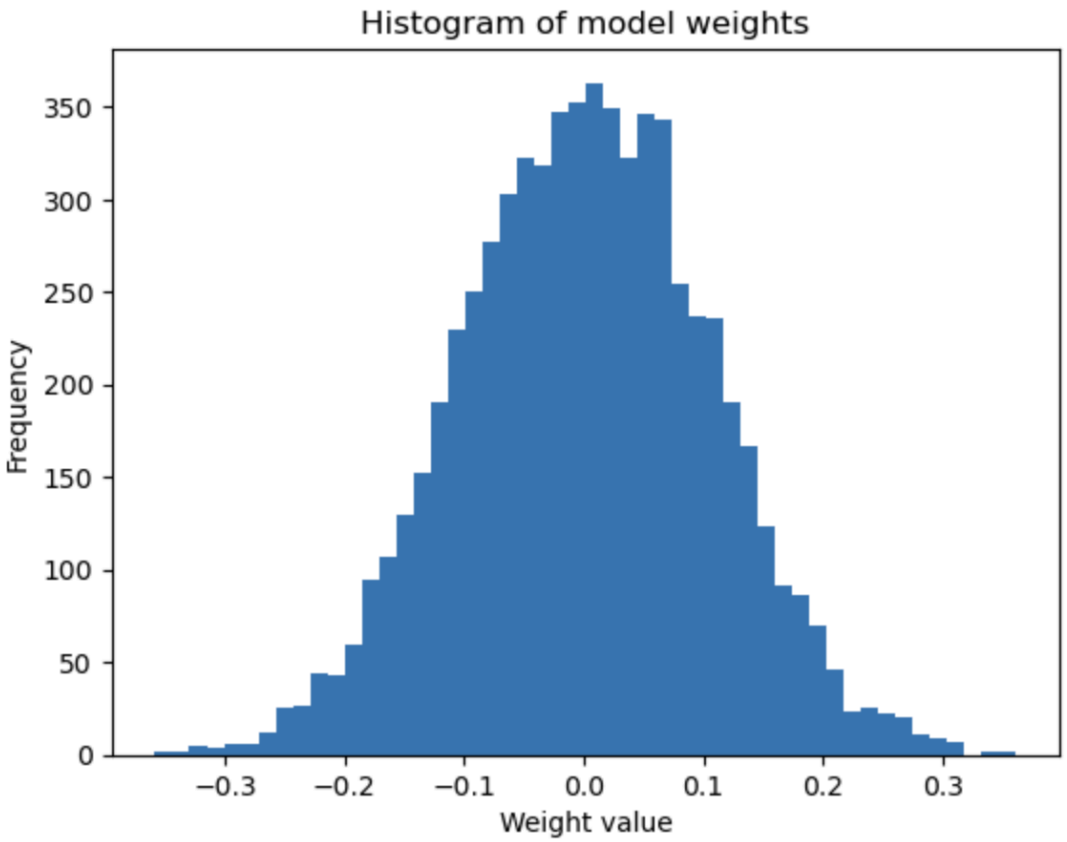
Comparative experiments were conducted to assess the impact of data augmentation on the SER model's performance. This analysis allowed us to evaluate techniques such as pitch shifting and time stretching, aiding in generalization to unseen data and mitigating overfitting risks.

```
import matplotlib.pyplot as plt

# Access the training history
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Plot training and validation loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, EPOCHS+1), train_loss, label='Training Loss')
plt.plot(range(1, EPOCHS+1), val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, EPOCHS+1), train_acc, label='Training Accuracy')
plt.plot(range(1, EPOCHS+1), val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```






```
weights = model.get_weights()

# Plot a histogram of the weights
plt.hist(weights[0].flatten(), bins=50)
plt.xlabel('Weight value')
plt.ylabel('Frequency')
plt.title('Histogram of model weights')
plt.show()
```




```
# Convert to TensorFlow Lite model with TF kernels fallback
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.target_spec.supported_ops = [
    tf.lite.OpsSet.TFLITE_BUILTINS,
    tf.lite.OpsSet.SELECT_TF_OPS
]
tflite_model = converter.convert()

# Save the TensorFlow Lite model
with tf.io.gfile.GFile("Models/SER.tflite", 'wb') as f:
    f.write(tflite_model)

# Apply optimizations and convert to quantized TensorFlow Lite
model
converter.optimizations = [tf.lite.Optimize.DEFAULT]
quant_tflite_model = converter.convert()

# Save the quantized TensorFlow Lite model
with tf.io.gfile.GFile("Models/SER_quant.tflite", 'wb') as f:
    f.write(quant_tflite_model)
```

```
print("Save the TensorFlow 'Lite' model!")
```

5. OUTPUT PREDICTION

```
import tensorflow as tf
from tensorflow.keras.models import load_model
import librosa
import numpy as np

# Load the saved model
saved_model_path =
'/Users/mananhingorani/Speech-Emotion-Recognition-TinyML/Models/Speech-E
motion-Recognition-Model-new.h5'
loaded_model = load_model(saved_model_path)
```

```
# Function to extract features from sound data
def extract_features(file_path, max_sequence_length=100):
    # Load sound file using librosa
    signal, sample_rate = librosa.load(file_path, sr=None)

    # Cropping & Resampling (similar to what you did during training)
    start_time = 0.4
    end_time = 1.9
    start_frame = int(start_time * sample_rate)
    end_frame = int(end_time * sample_rate)
    signal = signal[start_frame:end_frame]

    # Resample
    signal = librosa.resample(signal, orig_sr=sample_rate,
target_sr=16000)

    # Extract features for the sound data
    spectrogram = librosa.feature.melspectrogram(y=signal, sr=16000,
n_fft=2048, hop_length=512)
    mfcc = librosa.feature.mfcc(S=librosa.power_to_db(spectrogram),
n_mfcc=13)

    # Transpose features to have shape (num_features, sequence_length)
    features = mfcc.T

    # Pad or truncate features to a fixed length
    features = np.pad(features, ((0, max_sequence_length -
features.shape[0]), (0, 0)), mode='constant')
```

```
    return features

def predict_emotion(sound_file_path):
    # Extract features
    sound_features = extract_features(sound_file_path)

    # Ensure sound_features has the correct shape (adjust if necessary)
    sound_features = np.expand_dims(sound_features, axis=0)

    # Make predictions using the loaded model
    emotion_probabilities = loaded_model.predict(sound_features)

    # Convert the probabilities to emotion labels
    predicted_emotion = np.argmax(emotion_probabilities, axis=1)

    # Map the predicted label to the actual emotion class (e.g., using a
dictionary)
    emotion_mapping = {0: 'neutral', 1: 'happy', 2: 'surprise', 3:
'unpleasant'}
    predicted_emotion_label = emotion_mapping[predicted_emotion[0]]

    return predicted_emotion_label
```

```
sound_file_path = '/Users/mananhingorani/Downloads/Business Badhao
4.m4a'
predicted_emotion = predict_emotion(sound_file_path)

# Print or use the predicted emotion label
print("Predicted Emotion:", predicted_emotion)
```

```
1/1 [==============================] - 0s 379ms/step
Predicted Emotion: surprise
```

6. DOCKER AND STREAMLIT IMPLEMENTATION

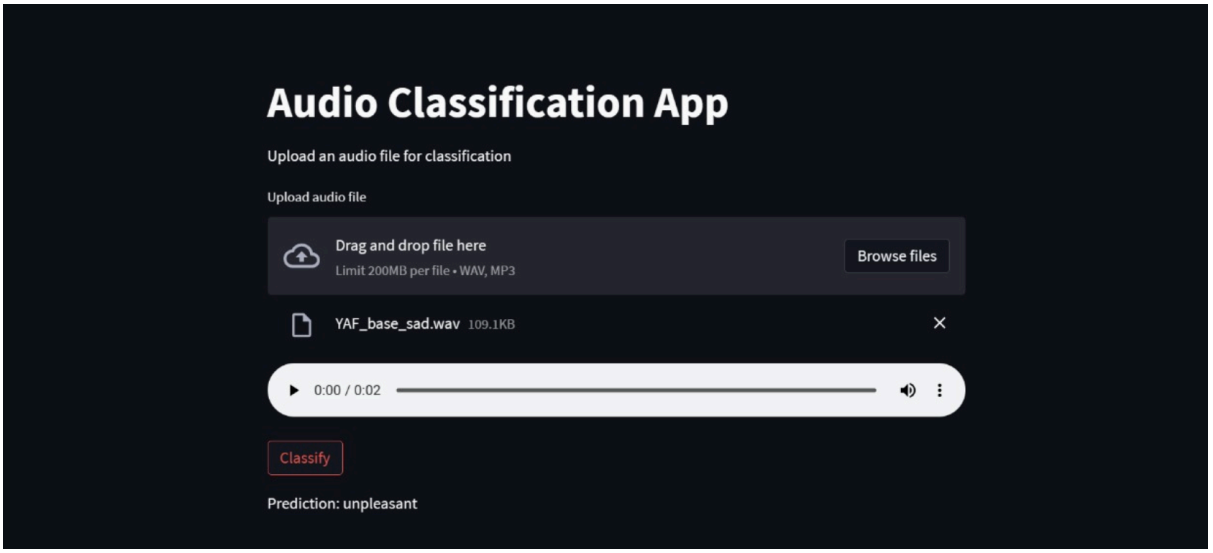
We deployed our speech emotion recognition (SER) system using Streamlit and Docker, streamlining the process for users and ensuring portability and scalability.

a. Streamlit Development:
   - Developed a user-friendly interface with Streamlit, allowing seamless interaction with our SER model.

- Users can input audio, which is processed by the model, and receive real-time emotion predictions.

b. Docker Containerization:
   - Containerized the Streamlit app with Docker for easy deployment and scalability.
   - Docker encapsulates the app, ensuring consistency and portability across different platforms.

c. Deployment Process:
   - Deployed the Docker container on a cloud-based or on-premises server.
   - Users access the SER application via a web browser, benefiting from Docker's automated handling of app execution.
   - Offers scalability and ease of maintenance for both developers and end-users.



7. DATA ANALYSIS

We have done data analysis of RAVDESS, TESS and SAVEE dataset using PowerBI and R.

a. <u>Analysis using R</u>

In this project, we're exploring three diverse datasets of human speech audio recordings to understand emotional content. The datasets - TESS, RAVDESS, and SAVEE - contain recordings showcasing different emotional states expressed by

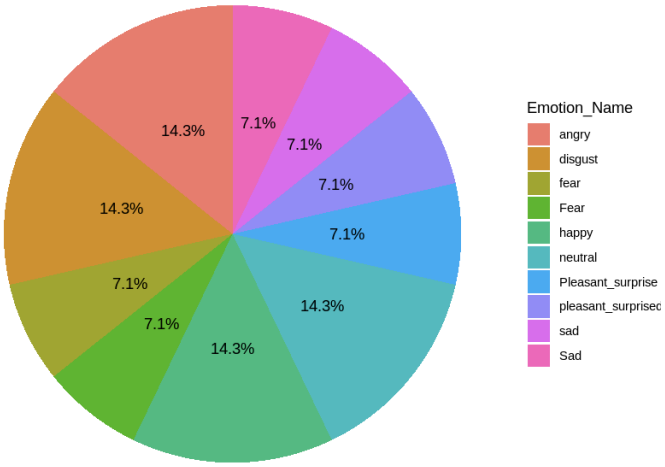
speakers. Our goal is to conduct exploratory analysis on these datasets to gain insights into the conveyed emotions.

I. **TESS DATASET**

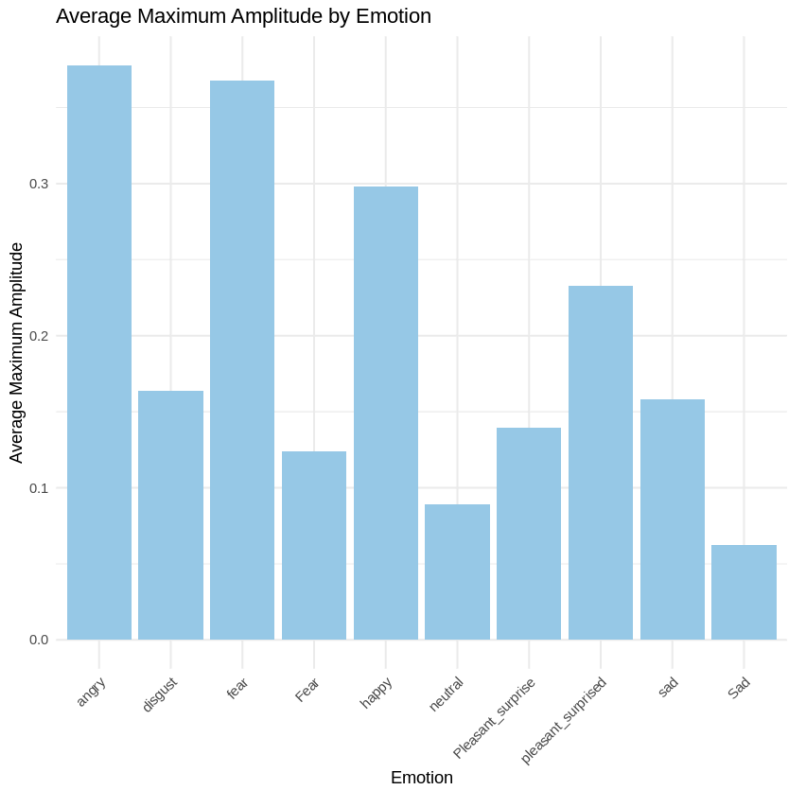
The dataset contains 2800 WAV format audio recordings featuring 200 target words spoken by two actresses (aged 26 and 64) expressing seven emotions. Each actress's recordings are categorized by emotion, allowing research into speech emotion recognition and age-related variations. This dataset offers insights into acoustic features of speech across emotions and age groups.

```
 # Load the ggplot2 library
library(ggplot2)

# Read the CSV file into a data frame
emotions <- read.csv("TESS_analysis.csv")

# Create a pie chart using ggplot2
pie_chart <- ggplot(emotions, aes(x = "", fill = Emotion_Name)) +
  geom_bar(width = 1) +
  coord_polar("y", start = 0) +
  labs(title = "Distribution of Emotion Names") +
  theme_void() +
  theme(legend.position = "right") +  # Optional: adjust legend
position
  geom_text(aes(label =
scales::percent((..count..)/sum(..count..)),
                y = ..count..), stat = "count", position =
position_stack(vjust = 0.5))

# Display the pie chart
print(pie_chart)
```

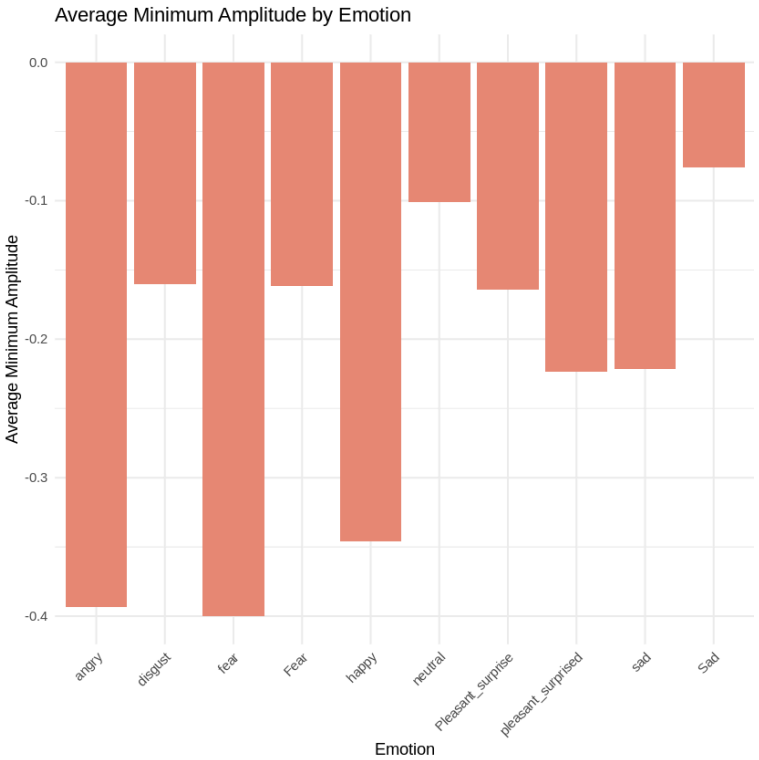
Distribution of Emotion Names




```r
# Convert columns to numeric if needed
emotions$Max_Amplitude <- as.numeric(emotions$Max_Amplitude)

# Calculate average maximum amplitude for each emotion
average_max_amplitude <- aggregate(Max_Amplitude ~ Emotion_Name, data =
emotions, FUN = mean)

# Create a bar chart for average maximum amplitude by emotion
max_amplitude_chart <- ggplot(average_max_amplitude, aes(x =
Emotion_Name, y = Max_Amplitude)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Average Maximum Amplitude by Emotion", x = "Emotion", y
= "Average Maximum Amplitude") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Display the bar chart
print(max_amplitude_chart)
```




```
# Convert columns to numeric if needed
emotions$Min_Amplitude <- as.numeric(emotions$Min_Amplitude)

# Calculate average minimum amplitude for each emotion
average_min_amplitude <- aggregate(Min_Amplitude ~ Emotion_Name, data =
emotions, FUN = mean)

# Create a bar chart for average minimum amplitude by emotion
min_amplitude_chart <- ggplot(average_min_amplitude, aes(x = Emotion_Name, y
= Min_Amplitude)) +
 geom_bar(stat = "identity", fill = "salmon") +
 labs(title = "Average Minimum Amplitude by Emotion", x = "Emotion", y =
"Average Minimum Amplitude") +
 theme_minimal() +
 theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Display the bar chart
print(min_amplitude_chart)
```

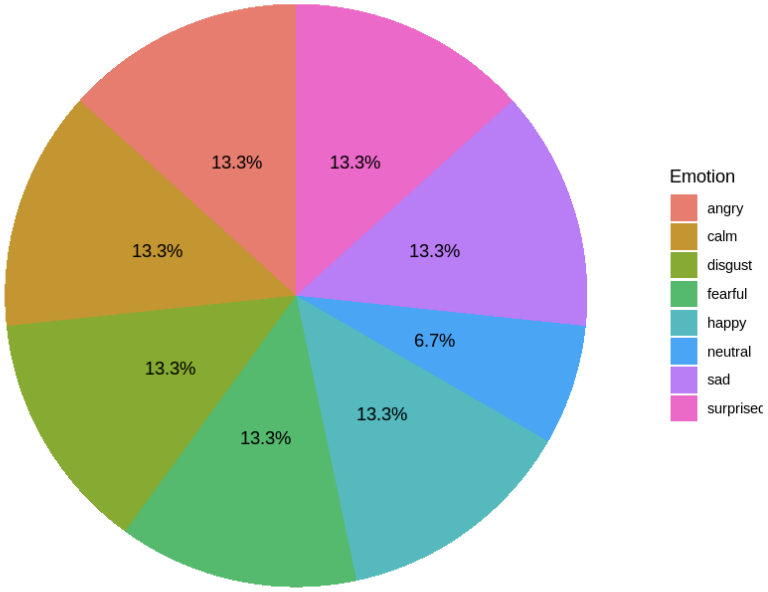


II. **RAVDESS DATASET**

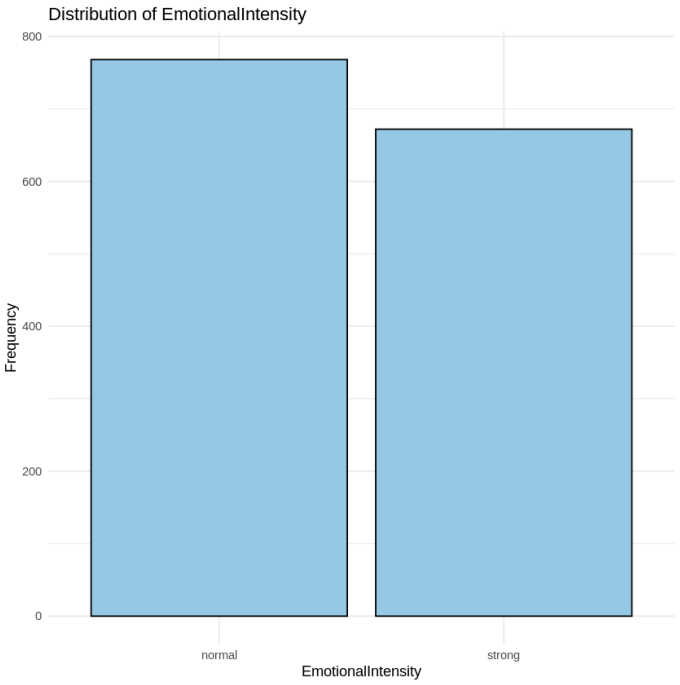
RAVDESS is a robust dataset for affective computing and emotion recognition research, featuring 24 professional actors expressing a wide range of emotions. It includes audio and video recordings showcasing emotions like neutral, happy, sad, angry, and more, at various intensities. With a systematic naming convention for each audio file, RAVDESS simplifies data organization and retrieval, making it a valuable resource for studying emotional expression across demographics.

```
# Read the CSV file into a data frame
emotions <- read.csv("RAVDESS_analysis.csv")

# Create a pie chart using ggplot2
pie_chart <- ggplot(emotions, aes(x = "", fill = Emotion)) +
 geom_bar(width = 1) +
 coord_polar("y", start = 0) +
 labs(title = "Distribution of Emotions") +
 theme_void() +
 theme(legend.position = "right") +  # Optional: adjust legend position
 geom_text(aes(label = scales::percent((..count..)/sum(..count..)),
               y = ..count..), stat = "count", position =
position_stack(vjust = 0.5))

# Display the pie chart
print(pie_chart)
```

Distribution of Emotions




```
# Plot the bar chart for distribution of Modality
bar_chart <- ggplot(data = data, aes(x = EmotionalIntensity)) +
 geom_bar(fill = "skyblue", color = "black") +
 labs(title = "Distribution of EmotionalIntensity", x =
"EmotionalIntensity", y = "Frequency") +
 theme_minimal()

# Show the plot
print(bar_chart)
```

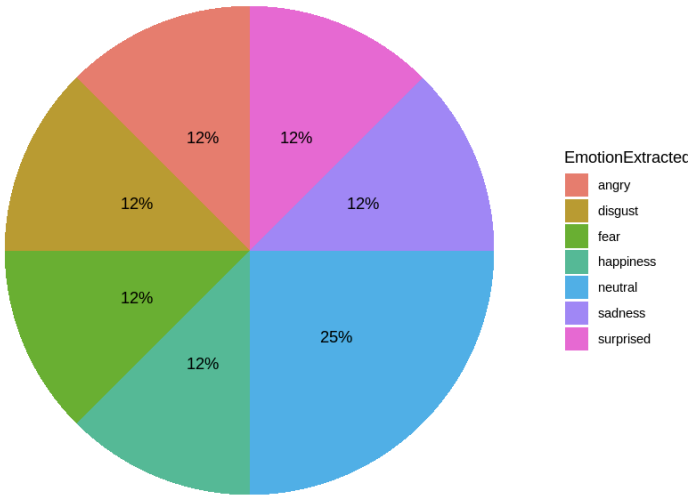


III. **SAVEE DATASET**

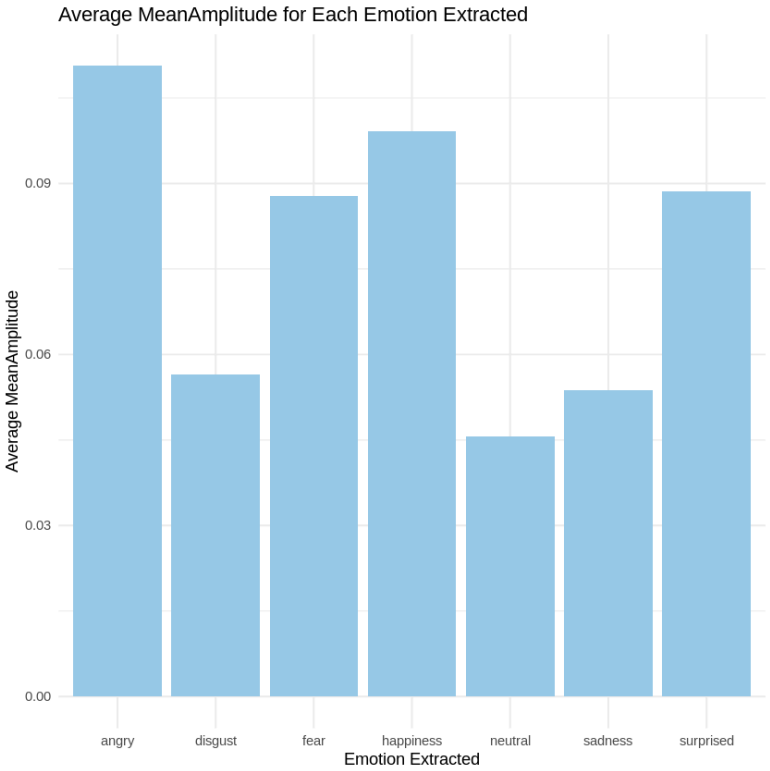
SAVEE is a widely used dataset for speech emotion recognition research, featuring recordings from four male speakers expressing seven emotions. It offers balanced representation across emotions and includes both acted and naturalistic expressions for realism. With high-quality audio files and detailed metadata, SAVEE facilitates emotion recognition algorithm development. It's a cornerstone resource for advancing affective computing and human-computer interaction research.

```
# Load the ggplot2 library
library(ggplot2)

# Read the CSV file into a data frame
emotions <- read.csv("SAVEE_analysis.csv")

# Create a pie chart using ggplot2
pie_chart <- ggplot(emotions, aes(x = "", fill = EmotionExtracted)) +
 geom_bar(width = 1) +
 coord_polar("y", start = 0) +
 labs(title = "Distribution of Emotion Names") +
 theme_void() +
 theme(legend.position = "right") +  # Optional: adjust legend position
 geom_text(aes(label = scales::percent((..count..)/sum(..count..)),
               y = ..count..), stat = "count", position =
position_stack(vjust = 0.5))

# Display the pie chart
print(pie_chart)
```

Distribution of Emotion Names

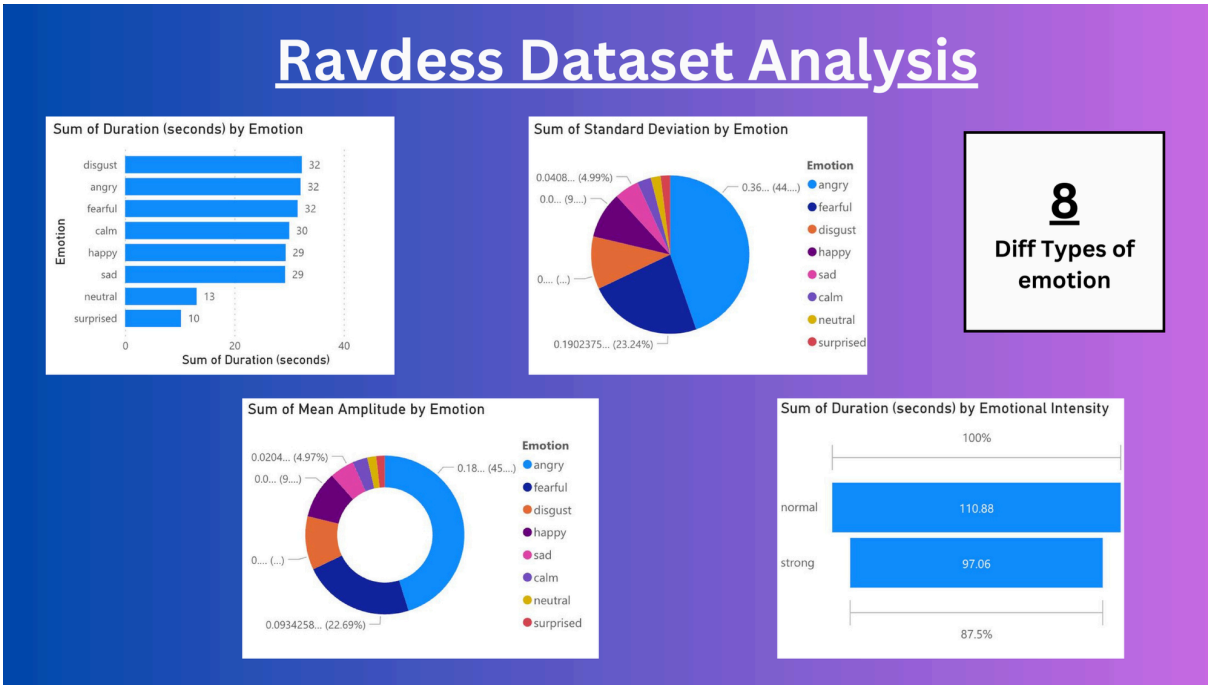


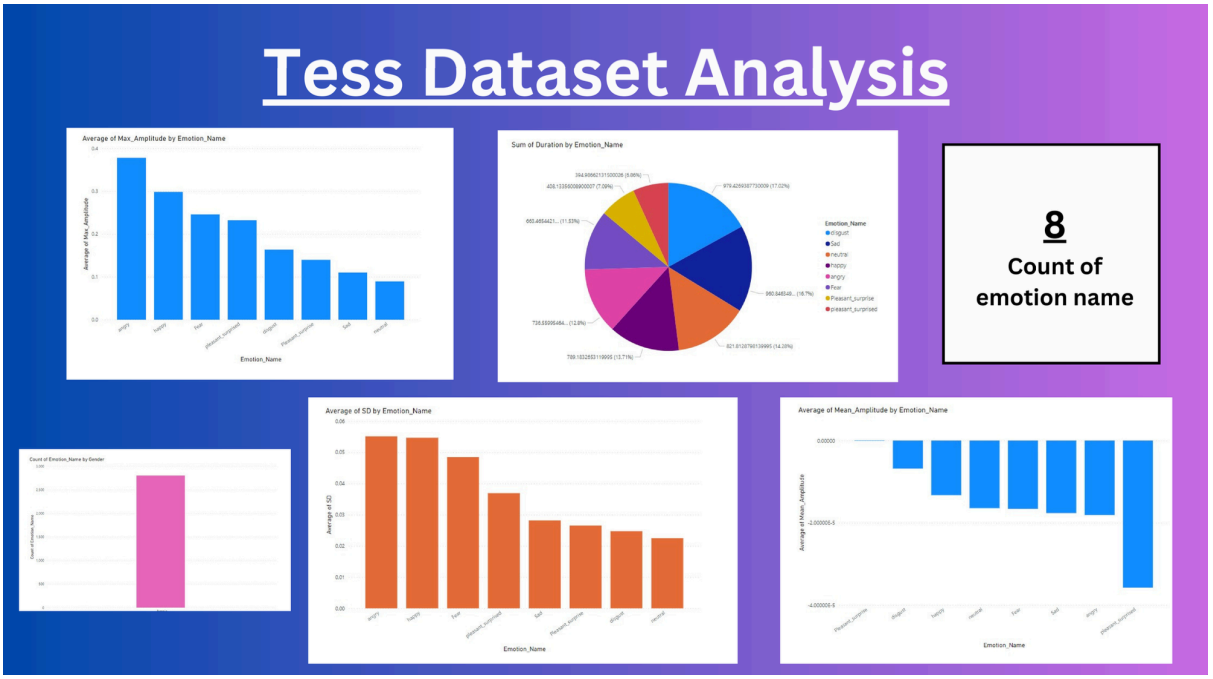

```
avg_max_amplitude <- data %>%
 group_by(EmotionExtracted) %>%
 summarize(avg_MaxAmplitude = mean(MeanAmplitude))

# Step 4: Create plot using ggplot2
ggplot(avg_max_amplitude, aes(x = EmotionExtracted, y = avg_MaxAmplitude)) +
 geom_bar(stat = "identity", fill = "skyblue") +
 labs(title = "Average MeanAmplitude for Each Emotion Extracted",
      x = "Emotion Extracted",
      y = "Average MeanAmplitude") +
 theme_minimal()
```



Exact detailed analysis using R for all 3 datasets can be found on this Google Collab Drive: https://colab.research.google.com/drive/1LHdFgN7JinKEpW_kSErDxnhpl5T_8NGq?usp=sharing

b. Analysis using PowerBi





8. CONCLUSION

In conclusion, our project demonstrated the effectiveness of integrating TinyML, Apache Spark, Docker, and PowerBI for speech emotion recognition. Through meticulous methodology and performance analysis, we showcased the potential of our SER model in accurately identifying emotions from speech data. This comprehensive approach lays a solid foundation for advancing human-machine interaction and emotional intelligence applications.