EmoQuant:- Quantitative Emotion and Sentiment Analysis for Teletherapy

GitHub-repository:- https://github.com/ayush09062004/EmoQuant

Tech-Flow:-

1. Choosing appropriate dataset
2. Training DeepLearning Models
3. Emotion-recognition using a trained model
4. Audio-based analysis
5. Final Web app preparation for the product

Code Submission :- Following are the indices of the submitted code in pdf format:-

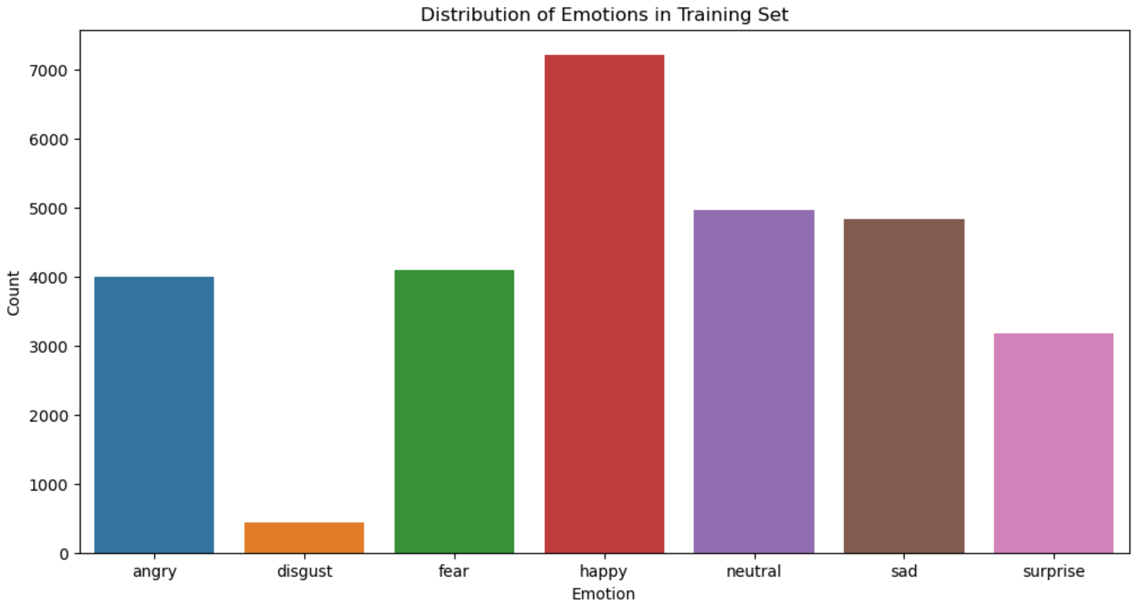
1. Exploratory data analysis of the Dataset chosen (Page no. 1-3)
2. Training the model on CNNs & VGG16 (Page no. 4-21) [The model was also trained on ViT & ResNet50 in Vs-Code so pdf file with their results are not available. For detailed code, kindly visit our project GitHub repository :- https://github.com/ayush09062004/EmoQuant]
3. Using the finalised trained model to utilise emotion recognition of face for the whole project (Page22-37)
4. Splitting audio from video for transcription & sentiment analysis. (Page37-39)
5. Audio transcription using whisper (Page40-43)
6. Transcription-based sentiment analysis using textblob & ncrlex(Page44-49)
7. Live audio based(Page50-51)

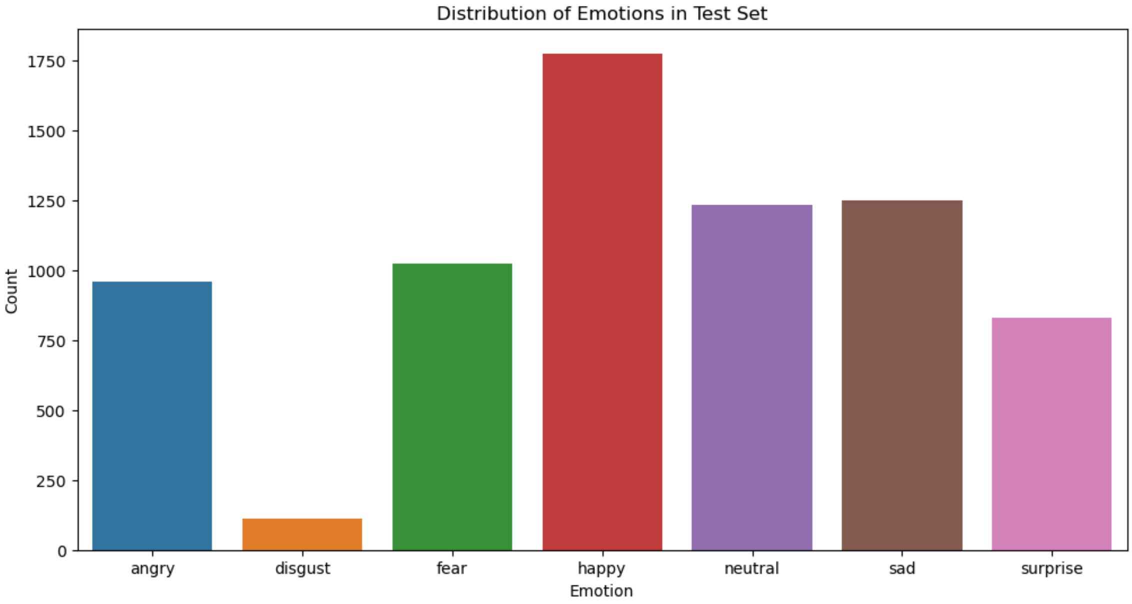
Team Members:-

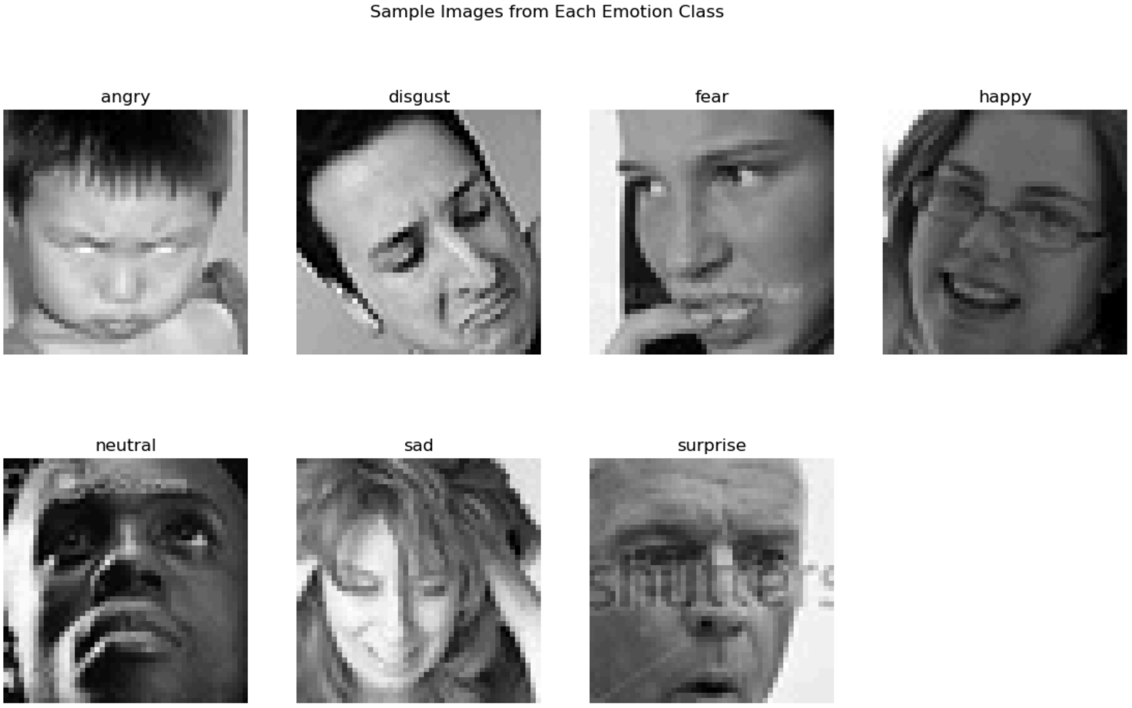
1. Ayush Raj
   B.tech in Biomedical Engg., School of Biomedical Engineering,
   Indian Institute of Technology (Banaras Hindu University), Varanasi,
   Uttar Pradesh, India-221005 [1]
   Contact:- +91-6207637403
   E-mail:- ayush.raj.bme22@iitbhu.ac.in
2. Hardik Goyal [1]
   Contact:- +91-9301714918
   E-mail:- hardik.goyal.bme22@itbhu.ac.in
3. Akshita R. [1]
   Contact:-+91-6232106749
   E-mail:- akshita.r.bme22@iitbhu.ac.in

```
In [*]: #Exploaratory Data Analysis on FER2013 dataset
        # Step 1: Import libraries
        import os
        import matplotlib.pyplot as plt
        import seaborn as sns
        import pandas as pd
        from PIL import Image

        # Step 2: Define paths for train and test directories
        train_dir = r"C:\Users\ayush\Downloads\fer2013\train"
        test_dir = r"C:\Users\ayush\Downloads\fer2013\test"

        # Step 3: Automatically detect emotion labels from folder names
        emotion_labels = sorted([folder for folder in os.listdir(train_dir) if os.p
        print(f"Detected emotion labels: {emotion_labels}")

        # Step 4: Check the number of images in each class for train and test sets
        train_data_count = {}
        test_data_count = {}

        # Check if each folder exists before counting images
        for emotion in emotion_labels:
            emotion_folder_train = os.path.join(train_dir, emotion)
            emotion_folder_test = os.path.join(test_dir, emotion)

            if os.path.exists(emotion_folder_train):
                train_data_count[emotion] = len(os.listdir(emotion_folder_train))
            else:
                print(f"Train folder for emotion '{emotion}' not found.")

            if os.path.exists(emotion_folder_test):
                test_data_count[emotion] = len(os.listdir(emotion_folder_test))
            else:
                print(f"Test folder for emotion '{emotion}' not found.")

        # Step 5: Visualize the distribution of classes
        train_df = pd.DataFrame(list(train_data_count.items()), columns=['Emotion',
        test_df = pd.DataFrame(list(test_data_count.items()), columns=['Emotion', '

        plt.figure(figsize=(12, 6))
        sns.barplot(x='Emotion', y='Count', data=train_df)
        plt.title('Distribution of Emotions in Training Set')
        plt.show()

        plt.figure(figsize=(12, 6))
        sns.barplot(x='Emotion', y='Count', data=test_df)
        plt.title('Distribution of Emotions in Test Set')
        plt.show()

        # Step 6: Visualize some images from each class in the training set
        plt.figure(figsize=(14, 8))
        for i, emotion in enumerate(emotion_labels):
            emotion_folder_train = os.path.join(train_dir, emotion)
            if os.path.exists(emotion_folder_train) and len(os.listdir(emotion_fold
                img_path = os.path.join(emotion_folder_train, os.listdir(emotion_fo
                try:
                    img = Image.open(img_path)
                    plt.subplot(2, 4, i+1)
                    plt.imshow(img, cmap='gray')
                    plt.title(emotion)
                    plt.axis('off')
                except Exception as e:
                    print(f"Error loading image from {img_path}: {e}")
            else:
                plt.subplot(2, 4, i+1)
                plt.title(f"Missing: {emotion}")
                plt.axis('off')
        plt.suptitle('Sample Images from Each Emotion Class')
        plt.show()

        # Step 7: Check image dimensions and other statistics
        image_dims = []
        for emotion in emotion_labels:
            emotion_folder_train = os.path.join(train_dir, emotion)
            if os.path.exists(emotion_folder_train):
                for img_file in os.listdir(emotion_folder_train):
                    img_path = os.path.join(emotion_folder_train, img_file)
                    try:
                        with Image.open(img_path) as img:
                            image_dims.append(img.size)
                    except Exception as e:
                        print(f"Error loading image from {img_path}: {e}")

        # Convert to DataFrame for better visualization
        dims_df = pd.DataFrame(image_dims, columns=['Width', 'Height'])

        plt.figure(figsize=(12, 6))
        sns.histplot(dims_df['Width'], kde=True, color='blue', label='Width')
        sns.histplot(dims_df['Height'], kde=True, color='red', label='Height')
        plt.title('Distribution of Image Dimensions in Training Set')
        plt.legend()
        plt.show()

        # Check for unique dimensions
        unique_dims = dims_df.drop_duplicates()
        print("Unique image dimensions in the dataset:")
        print(unique_dims)
```

```
Detected emotion labels: ['angry', 'disgust', 'fear', 'happy', 'neutral',
'sad', 'surprise']
```




Distribution of Emotions in Test Set
Count
0
250
500
750
1000
1250
1500
1750
angry
disgust
fear
happy
neutral
sad
surprise
Emotion


Sample Images from Each Emotion Class
angry
disgust
fear
happy
neutral
sad
surprise

In [ ]:

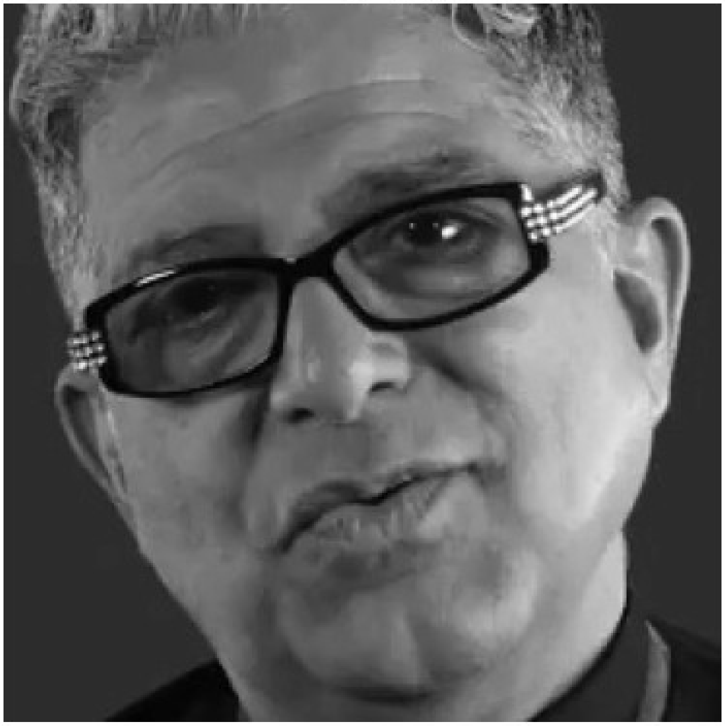
```
from google.colab import files
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Step 1: Upload video
uploaded = files.upload()
video_path = list(uploaded.keys())[0]

# Step 2: Extract frames every 90 seconds with face detection, crop
faces, and convert to grayscale
output_folder = '/content/frames/'
os.makedirs(output_folder, exist_ok=True)

# Load Haar Cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(video_path)
fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_interval = 90 * fps
frame_count = 0
saved_face_count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break

    if frame_count % frame_interval == 0:
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray_frame,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

        for (x, y, w, h) in faces:
            face = gray_frame[y:y+h, x:x+w]  # Crop the face from the
grayscale frame
            face_path = os.path.join(output_folder,
f'face_{saved_face_count}.jpg')
            cv2.imwrite(face_path, face)
            saved_face_count += 1

    frame_count += 1

cap.release()
print(f"Saved {saved_face_count} grayscale face images.")

# Step 3: Display saved face images
```

```
saved_faces = [f for f in os.listdir(output_folder) if
f.startswith('face_')]

for face_file in saved_faces[:5]:  # Display first 5 face images
    img_path = os.path.join(output_folder, face_file)
    img = mpimg.imread(img_path)
    plt.imshow(img, cmap='gray')  # Display images in grayscale
    plt.title(face_file)
    plt.axis('off')
    plt.show()
```

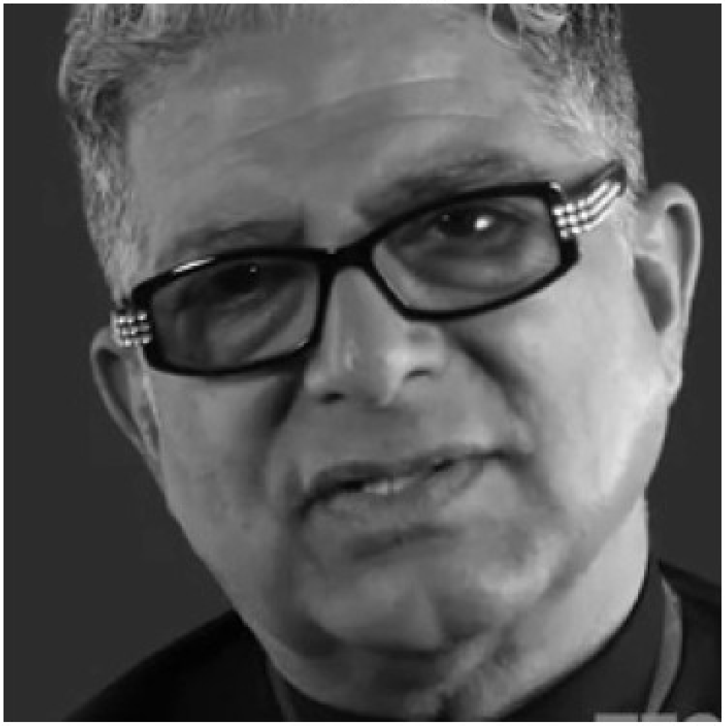
```
<IPython.core.display.HTML object>

Saving videoplayback.mp4 to videoplayback (4).mp4
Saved 3 grayscale face images.
```

face_0.jpg

face_1.jpg



face_2.jpg



```python
from google.colab import files
import zipfile
import os

# Upload the FER-2013 dataset zip file
uploaded = files.upload()

# Unzip the dataset
dataset_zip = list(uploaded.keys())[0]
with zipfile.ZipFile(dataset_zip, 'r') as zip_ref:
    zip_ref.extractall('/content/fer2013')

print("Dataset extracted!")
```

```
<IPython.core.display.HTML object>
```

```
Saving archive (3).zip to archive (3).zip
Dataset extracted!
```

```python
import numpy as np
import cv2
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Define paths
train_folder = '/content/fer2013/train/'
test_folder = '/content/fer2013/test/'

# Categories
emotion_labels = ['angry', 'disgust', 'fear', 'happy', 'sad',
'surprise', 'neutral']

# Load images from folder
def load_images_from_folder(folder):
    images = []
    labels = []
    for label in emotion_labels:
        emotion_folder = os.path.join(folder, label)
        for filename in os.listdir(emotion_folder):
            if filename.endswith('.jpg'):
                img_path = os.path.join(emotion_folder, filename)
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
                img = cv2.resize(img, (48, 48))  # Resize to 48x48
                images.append(img)
                labels.append(label)
    return np.array(images), np.array(labels)

# Load training and test data
X_train, y_train = load_images_from_folder(train_folder)
```

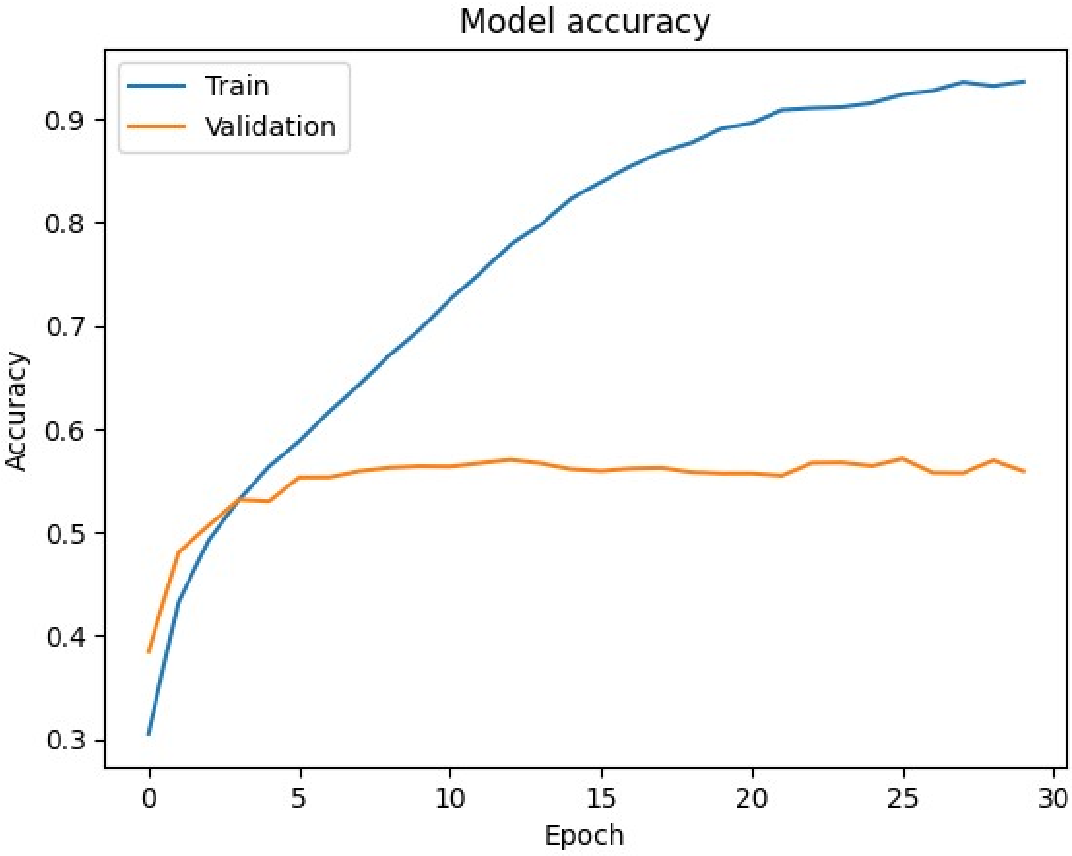
```python
X_test, y_test = load_images_from_folder(test_folder)

# Normalize images
X_train = X_train / 255.0
X_test = X_test / 255.0

# Reshape for CNN input
X_train = X_train.reshape(-1, 48, 48, 1)
X_test = X_test.reshape(-1, 48, 48, 1)

# Encode labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Split training data for validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

# Build the model
model = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(7, activation='softmax')  # 7 emotions
])

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=64,
    validation_data=(X_val, y_val),
    verbose=1
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')

# Save the model
model.save('/content/emotion_detection_model.h5')
```

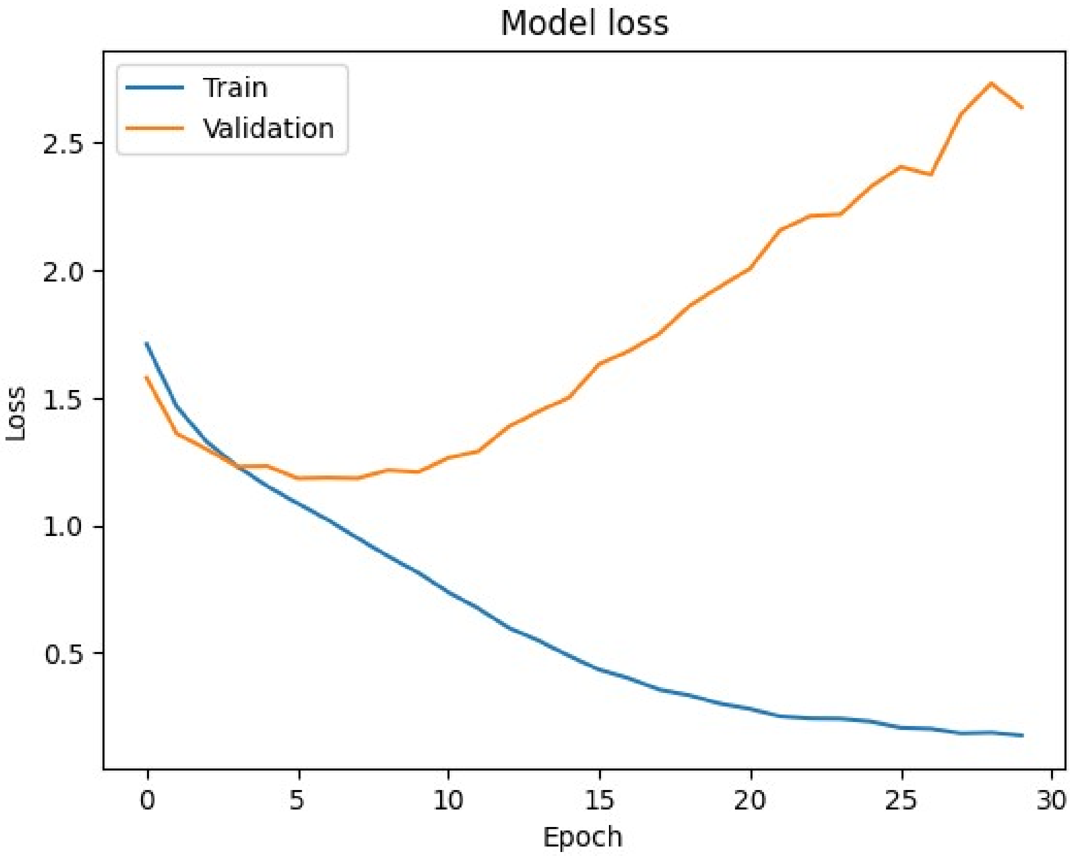
```
Epoch 1/30
 33/359 ━━━━━━━━━━━━━━━━━━━━ 5:56 1s/step - accuracy: 0.2284 - loss:
1.8766

----------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-10-248b42363d51> in <cell line: 27>()
     25
     26 # Train the model
---> 27 history = model.fit(
     28     X_train, y_train,
     29     epochs=30,

/usr/local/lib/python3.10/dist-packages/keras/src/utils/traceback_util
s.py in error_handler(*args, **kwargs)
    115         filtered_tb = None
    116         try:
--> 117             return fn(*args, **kwargs)
    118         except Exception as e:
    119             filtered_tb =
_process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/
trainer.py in fit(self, x, y, batch_size, epochs, verbose, callbacks,
validation_split, validation_data, shuffle, class_weight,
sample_weight, initial_epoch, steps_per_epoch, validation_steps,
validation_batch_size, validation_freq)
    316                 for step, iterator in
epoch_iterator.enumerate_epoch():
    317                     callbacks.on_train_batch_begin(step)
--> 318                     logs = self.train_function(iterator)
    319                     logs = self._pythonify_logs(logs)
    320                     callbacks.on_train_batch_end(step, logs)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/util/traceba
```

```
ck_utils.py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/polymo
rphic_function/polymorphic_function.py in __call__(self, *args,
**kwds)
    831 
    832       with OptionalXlaContext(self._jit_compile):
--> 833         result = self._call(*args, **kwds)
    834 
    835       new_tracing_count =
self.experimental_get_tracing_count()

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/polymo
rphic_function/polymorphic_function.py in _call(self, *args, **kwds)
    876       # In this case we have not created variables on the
first call. So we can
    877       # run the first trace but we should fail if variables
are created.
--> 878       results = tracing_compilation.call_function(
    879           args, kwds, self._variable_creation_config
    880       )

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/polymo
rphic_function/tracing_compilation.py in call_function(args, kwargs,
tracing_options)
    137   bound_args = function.function_type.bind(*args, **kwargs)
    138   flat_inputs =
function.function_type.unpack_inputs(bound_args)
--> 139   return function._call_flat(  # pylint: disable=protected-
access
    140       flat_inputs, captured_inputs=function.captured_inputs
    141   )

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/polymo
rphic_function/concrete_function.py in _call_flat(self, tensor_inputs,
captured_inputs)
   1320         and executing_eagerly):
   1321       # No tape is watching; skip to running the function.
-> 1322       return self._inference_function.call_preflattened(args)
   1323     forward_backward =
self._select_forward_and_backward_functions(
   1324         args,

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/polymo
rphic_function/atomic_function.py in call_preflattened(self, args)
```

```
    214   def call_preflattened(self, args: Sequence[core.Tensor]) ->
Any:
    215     """Calls with flattened tensor inputs and returns the
structured output."""
--> 216     flat_outputs = self.call_flat(*args)
    217     return self.function_type.pack_output(flat_outputs)
    218

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/polymo
rphic_function/atomic_function.py in call_flat(self, *args)
    249         with record.stop_recording():
    250           if self._bound_context.executing_eagerly():
--> 251             outputs = self._bound_context.call_function(
    252                 self.name,
    253                 list(args),

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/contex
t.py in call_function(self, name, tensor_inputs, num_outputs)
   1550     cancellation_context = cancellation.context()
   1551     if cancellation_context is None:
-> 1552       outputs = execute.execute(
   1553           name.decode("utf-8"),
   1554           num_outputs=num_outputs,

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execut
e.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle,
device_name, op_name,
     54                                         inputs, attrs,
num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





```
files.download('/content/emotion_detection_model.h5')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
from google.colab import drive
drive.mount('/content/drive')
model_path = '/content/drive/My Drive/emotion_detection_model.h5'

# Save the model to Google Drive
model.save(model_path)
import os

# Define the path to the specific folder in Google Drive
# Replace with the path to your specific folder
drive_folder_path = '/content/drive/My
Drive/17FRUVkDI8h6StPa6R4Qf0cW2oms4NX1N'
model_path = os.path.join(drive_folder_path,
'emotion_detection_model.h5')

# Save the model to the specified folder
model.save(model_path)
```

```python
print(f"Model saved to {model_path}")
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```
Mounted at /content/drive
```

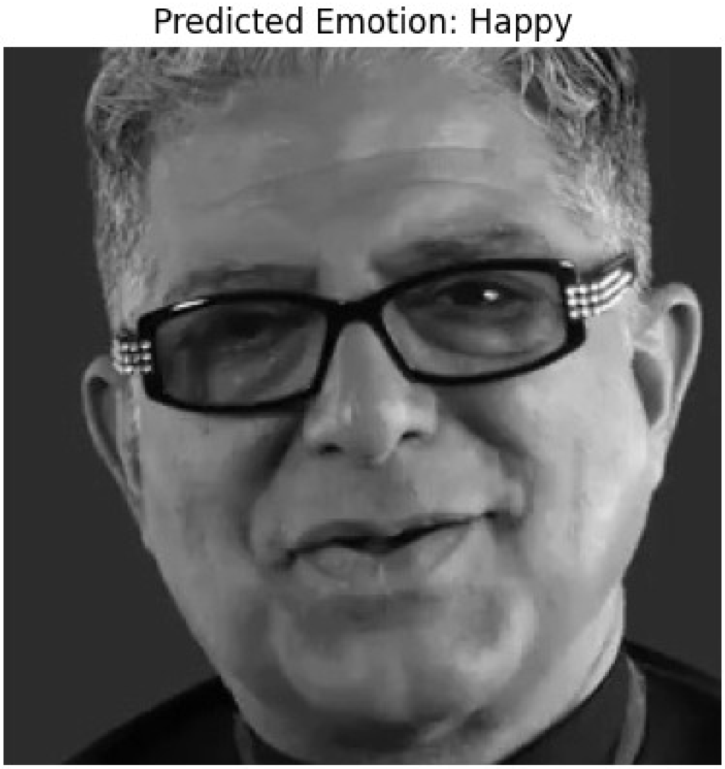
```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```
Model saved to /content/drive/My
Drive/17FRUVkDI8h6StPa6R4Qf0cW2oms4NX1N/emotion_detection_model.h5
```

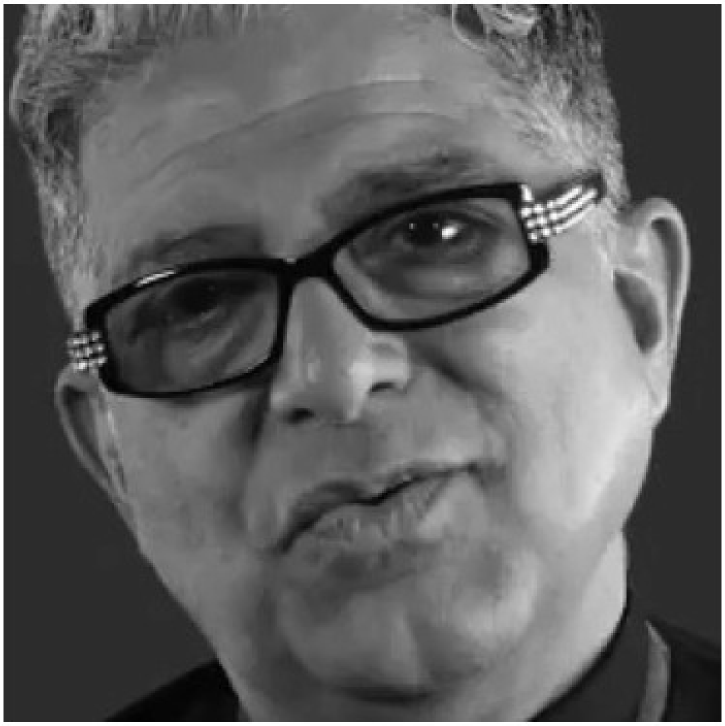
```python
def preprocess_image(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Load image
in grayscale
    img = cv2.resize(img, (48, 48))  # Resize to match model input
    img = img / 255.0  # Normalize pixel values
    img = img.reshape(1, 48, 48, 1)  # Reshape for model prediction
    return img

# Define paths to the images
image_paths = [
    '/content/frames/face_0.jpg',
    '/content/frames/face_1.jpg',
    '/content/frames/face_2.jpg'
]

# Preprocess the images
preprocessed_images = [preprocess_image(path) for path in image_paths]

emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad',
'Surprise', 'Neutral']

# Predict emotions
predictions = [model.predict(img) for img in preprocessed_images]
predicted_labels = [emotion_labels[np.argmax(pred)] for pred in
predictions]

print("Predicted Emotions:")
for path, label in zip(image_paths, predicted_labels):
    print(f"Image: {path} -> Emotion: {label}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
Predicted Emotions:
Image: /content/frames/face_0.jpg -> Emotion: Happy
Image: /content/frames/face_1.jpg -> Emotion: Happy
Image: /content/frames/face_2.jpg -> Emotion: Happy
```

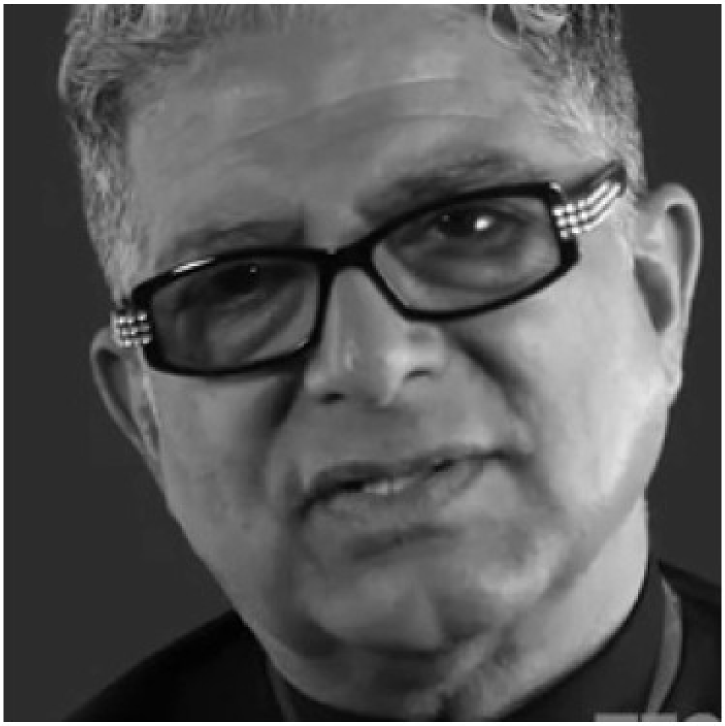
```python
import matplotlib.pyplot as plt

def display_image_with_prediction(image_path, label):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    plt.imshow(img, cmap='gray')
    plt.title(f'Predicted Emotion: {label}')
    plt.axis('off')
    plt.show()

# Display images with predicted emotions
for path, label in zip(image_paths, predicted_labels):
    display_image_with_prediction(path, label)
```



Predicted Emotion: Happy



Predicted Emotion: Happy



```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Example usage with a batch of images
# datagen.flow(X_train, y_train, batch_size=32)

from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import GlobalAveragePooling2D

base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(48, 48, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(7, activation='softmax')(x)

model = tf.keras.Model(inputs=base_model.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```python
def convert_to_rgb(images):
    # Convert grayscale images to RGB
    return np.stack([np.stack([img.squeeze()]*3, axis=-1) for img in
images])

# Convert the images
X_train_rgb = convert_to_rgb(X_train)
X_val_rgb = convert_to_rgb(X_val)
X_test_rgb = convert_to_rgb(X_test)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Create data generator for training data
train_generator = datagen.flow(X_train_rgb, y_train, batch_size=64)

# Optionally, create a generator for validation data if needed
val_datagen = ImageDataGenerator()
val_generator = val_datagen.flow(X_val_rgb, y_val, batch_size=64)

import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense,
Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Build Model using VGG16 as base
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(48, 48, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(7, activation='softmax')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Define callbacks
lr_reduction = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=2, verbose=1, min_lr=0.00001)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
```

```
# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator,
    callbacks=[lr_reduction, early_stopping],
    verbose=1
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test_rgb, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')

# Save the model
model.save('/content/drive/My
Drive/17FRUVkDI8h6StPa6R4Qf0cW2oms4NX1N/emotion_detection_model2.h5')

# Predict on the test set
y_pred = model.predict(X_test_rgb)
y_pred_labels = np.argmax(y_pred, axis=1)

# Generate confusion matrix and classification report
cm = confusion_matrix(np.argmax(y_test, axis=1), y_pred_labels)
cr = classification_report(np.argmax(y_test, axis=1), y_pred_labels)

print("Confusion Matrix:\n", cm)
print("Classification Report:\n", cr)
```

```
Epoch 1/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 848s 2s/step - accuracy: 0.2666 - loss:
1.8204 - val_accuracy: 0.3584 - val_loss: 1.6435 - learning_rate:
0.0010
Epoch 2/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 840s 2s/step - accuracy: 0.3214 - loss:
1.6950 - val_accuracy: 0.3546 - val_loss: 1.6389 - learning_rate:
0.0010
Epoch 3/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 829s 2s/step - accuracy: 0.3325 - loss:
1.6785 - val_accuracy: 0.3751 - val_loss: 1.6116 - learning_rate:
0.0010
Epoch 4/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 866s 2s/step - accuracy: 0.3285 - loss:
1.6700 - val_accuracy: 0.3769 - val_loss: 1.6050 - learning_rate:
0.0010
Epoch 5/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 884s 2s/step - accuracy: 0.3371 - loss:
1.6717 - val_accuracy: 0.3678 - val_loss: 1.6160 - learning_rate:
0.0010
Epoch 6/10
```

```
359/359 ━━━━━━━━━━━━━━━━━━━━ 908s 2s/step - accuracy: 0.3423 - loss:
1.6590 - val_accuracy: 0.3816 - val_loss: 1.5990 - learning_rate:
0.0010
Epoch 7/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 879s 2s/step - accuracy: 0.3529 - loss:
1.6407 - val_accuracy: 0.3830 - val_loss: 1.5909 - learning_rate:
0.0010
Epoch 8/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 875s 2s/step - accuracy: 0.3462 - loss:
1.6478 - val_accuracy: 0.3723 - val_loss: 1.5979 - learning_rate:
0.0010
Epoch 9/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.3486 - loss:
1.6390
Epoch 9: ReduceLROnPlateau reducing learning rate to
0.0005000000237487257.
359/359 ━━━━━━━━━━━━━━━━━━━━ 923s 2s/step - accuracy: 0.3486 - loss:
1.6390 - val_accuracy: 0.3748 - val_loss: 1.6014 - learning_rate:
0.0010
Epoch 10/10
359/359 ━━━━━━━━━━━━━━━━━━━━ 931s 2s/step - accuracy: 0.3545 - loss:
1.6347 - val_accuracy: 0.3833 - val_loss: 1.5788 - learning_rate:
5.0000e-04
225/225 ━━━━━━━━━━━━━━━━━━━━ 205s 909ms/step - accuracy: 0.2923 -
loss: 1.7231

WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

Test accuracy: 0.3869
225/225 ━━━━━━━━━━━━━━━━━━━━ 205s 909ms/step
Confusion Matrix:
 [[ 131    0   51  476  174   70   56]
 [  10    0    8   57   25    6    5]
 [  45    0  118  455  156   94  156]
 [  31    0   31 1419  162   49   82]
 [  35    0   53  574  466   49   56]
 [  59    0   68  673  212  188   47]
 [  17    0   58  195   93   13  455]]
Classification Report:
               precision    recall  f1-score   support

           0       0.40      0.14      0.20       958
           1       0.00      0.00      0.00       111
           2       0.30      0.12      0.17      1024
           3       0.37      0.80      0.50      1774
           4       0.36      0.38      0.37      1233
           5       0.40      0.15      0.22      1247
           6       0.53      0.55      0.54       831

    accuracy                           0.39      7178
   macro avg       0.34      0.30      0.29      7178
weighted avg       0.38      0.39      0.34      7178


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1471: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1471: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1471: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
!pip install deepface
!pip install opencv-python-headless
!pip install matplotlib
```

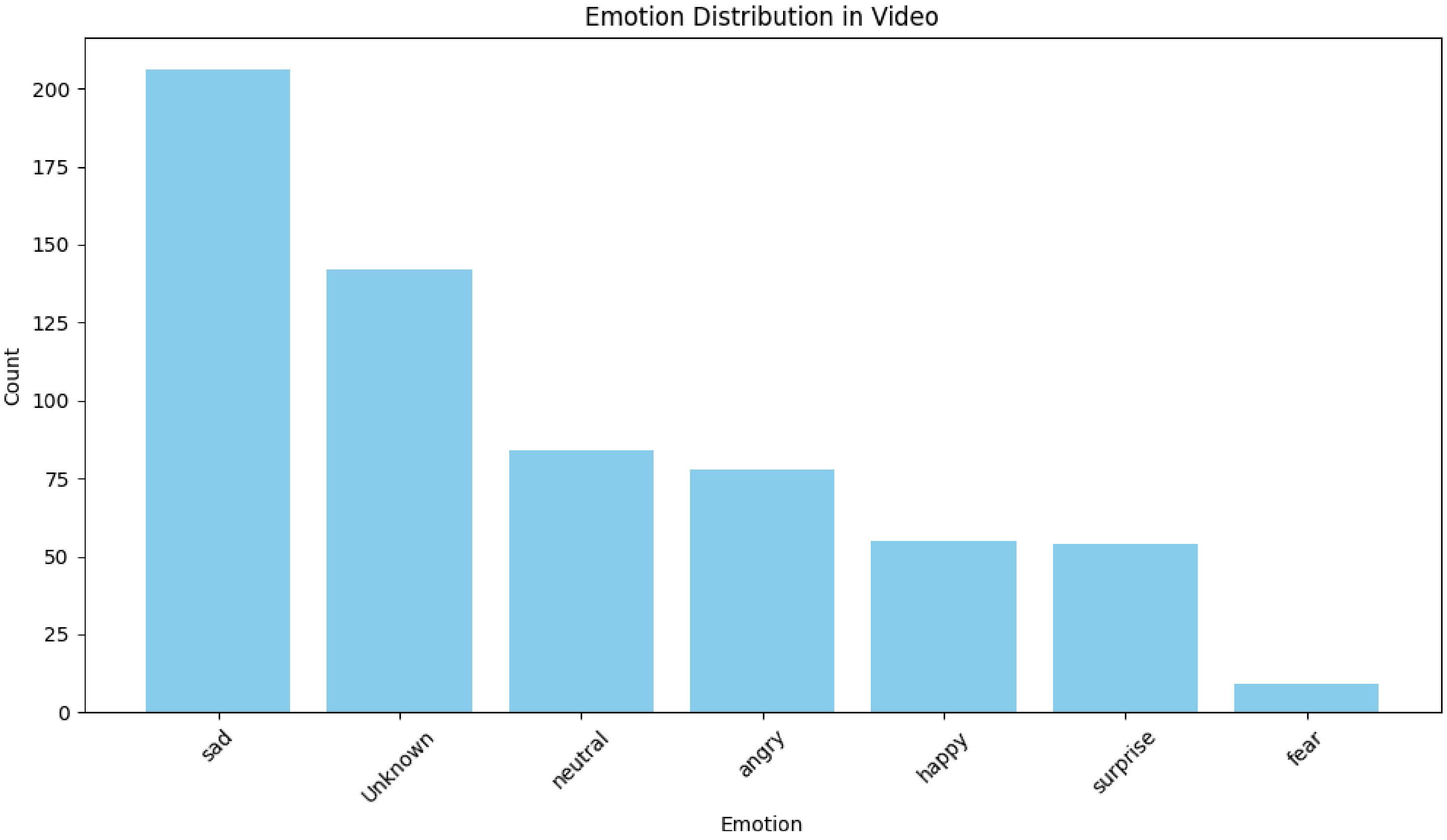
```
deepface
ng deepface-0.0.92-py3-none-any.whl.metadata (27 kB)
 already satisfied: requests>=2.27.1 in /usr/local/lib/python3.10/dist-packages (from deepface) (2.32.3)
 already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from deepface) (1.26.4)
 already satisfied: pandas>=0.23.4 in /usr/local/lib/python3.10/dist-packages (from deepface) (2.1.4)
 already satisfied: gdown>=3.10.1 in /usr/local/lib/python3.10/dist-packages (from deepface) (5.1.0)
 already satisfied: tqdm>=4.30.0 in /usr/local/lib/python3.10/dist-packages (from deepface) (4.66.5)
 already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.10/dist-packages (from deepface) (9.4.0)
 already satisfied: opencv-python>=4.5.5.64 in /usr/local/lib/python3.10/dist-packages (from deepface) (4.10.0.84)
 already satisfied: tensorflow>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from deepface) (2.17.0)
 already satisfied: keras>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from deepface) (3.4.1)
 already satisfied: Flask>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from deepface) (2.2.5)
mtcnn>=0.1.0 (from deepface)
ng mtcnn-0.1.1-py3-none-any.whl.metadata (5.8 kB)
retina-face>=0.0.1 (from deepface)
ng retina_face-0.0.17-py3-none-any.whl.metadata (10 kB)
fire>=0.4.0 (from deepface)
ng fire-0.6.0.tar.gz (88 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.4/88.4 kB 3.3 MB/s eta 0:00:00
 metadata (setup.py) ... done
gunicorn>=20.1.0 (from deepface)
ng gunicorn-22.0.0-py3-none-any.whl.metadata (4.4 kB)
 already satisfied: six in /usr/local/lib/python3.10/dist-packages (from fire>=0.4.0->deepface) (1.16.0)
 already satisfied: termcolor in /usr/local/lib/python3.10/dist-packages (from fire>=0.4.0->deepface) (2.4.0)
 already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from Flask>=1.1.2->deepface) (3.0.3)
 already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=1.1.2->deepface) (3.1.4)
 already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=1.1.2->deepface) (2.2.0)
 already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=1.1.2->deepface) (8.1.7)
 already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown>=3.10.1->deepface) (4.12.3)
 already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown>=3.10.1->deepface) (3.15.4)
 already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from gunicorn>=20.1.0->deepface) (24.1)
 already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras>=2.2.0->deepface) (1.4.0)
 already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=2.2.0->deepface) (13.7.1)
 already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=2.2.0->deepface) (0.0.8)
 already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras>=2.2.0->deepface) (3.11.0)
 already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=2.2.0->deepface) (0.12.1)
 already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras>=2.2.0->deepface) (0.4.0)
 already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.23.4->deepface) (2.8.2)
 already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.23.4->deepface) (2024.1)
 already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.23.4->deepface) (2024.1)
 already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27.1->deepface) (3.3.2)
 already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27.1->deepface) (3.7)
 already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27.1->deepface) (2.0.7)
 already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27.1->deepface) (2024.7.4)
 already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (1.6.3)
 already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (24.3.25)
 already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (0.6.0)
 already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (0.2.0)
 already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (18.1.1)
 already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (3.3.0)
 already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deep
 already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (71.0.4)
 already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (4.12.2)
 already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (1.16.0)
 already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (1.64.1)
 already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=1.9.0->deepface) (2.17.0)
```

```
# Step 1: Import Libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from deepface import DeepFace
from google.colab import files

# Step 2: Upload Video File
uploaded = files.upload()

# Assuming one video file is uploaded
video_file = list(uploaded.keys())[0]

# Load Haar Cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Step 3: Process Video, Detect Faces, and Analyze Emotions
def analyze_emotions_from_faces(video_path):
    cap = cv2.VideoCapture(video_path)

    emotions = []
    frame_count = 0

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Convert frame to grayscale for face detection
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces
        faces = face_cascade.detectMultiScale(gray_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

        for (x, y, w, h) in faces:
            # Extract the face from the frame
            face_img = frame[y:y+h, x:x+w]

            # Convert face image to RGB
            rgb_face_img = cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB)

            # Detect emotions
            try:
                result = DeepFace.analyze(rgb_face_img, actions=['emotion'])
                emotion = result[0]['dominant_emotion']
            except Exception as e:
                emotion = 'Unknown'

            emotions.append(emotion)
            frame_count += 1
    cap.release()
    return emotions, frame_count

# Analyze the uploaded video
emotions, total_frames = analyze_emotions_from_faces(video_file)

# Step 4: Create a DataFrame for Visualization
df = pd.DataFrame(emotions, columns=['Emotion'])
emotion_counts = df['Emotion'].value_counts()

# Step 5: Visualize Emotion Distribution
plt.figure(figsize=(12, 6))
plt.bar(emotion_counts.index, emotion_counts.values, color='skyblue')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.title('Emotion Distribution in Video')
plt.xticks(rotation=45)
plt.show()

# Display the results
print(f"Total frames processed: {total_frames}")
print("Emotion counts:")
print(emotion_counts)

# Step 6: Create a Detailed Dashboard (Optional)
fig, ax = plt.subplots(figsize=(12, 8))
ax.bar(emotion_counts.index, emotion_counts.values, color='lightgreen')
ax.set_xlabel('Emotion')
ax.set_ylabel('Count')
ax.set_title('Emotion Distribution Dashboard')

# Add annotations
for i, value in enumerate(emotion_counts.values):
    ax.text(i, value + 2, str(value), ha='center', va='bottom')

plt.xticks(rotation=45)
plt.show()
```

```
24-08-10 05:50:48 - Directory /root/.deepface created
24-08-10 05:50:48 - Directory /root/.deepface/weights created
```

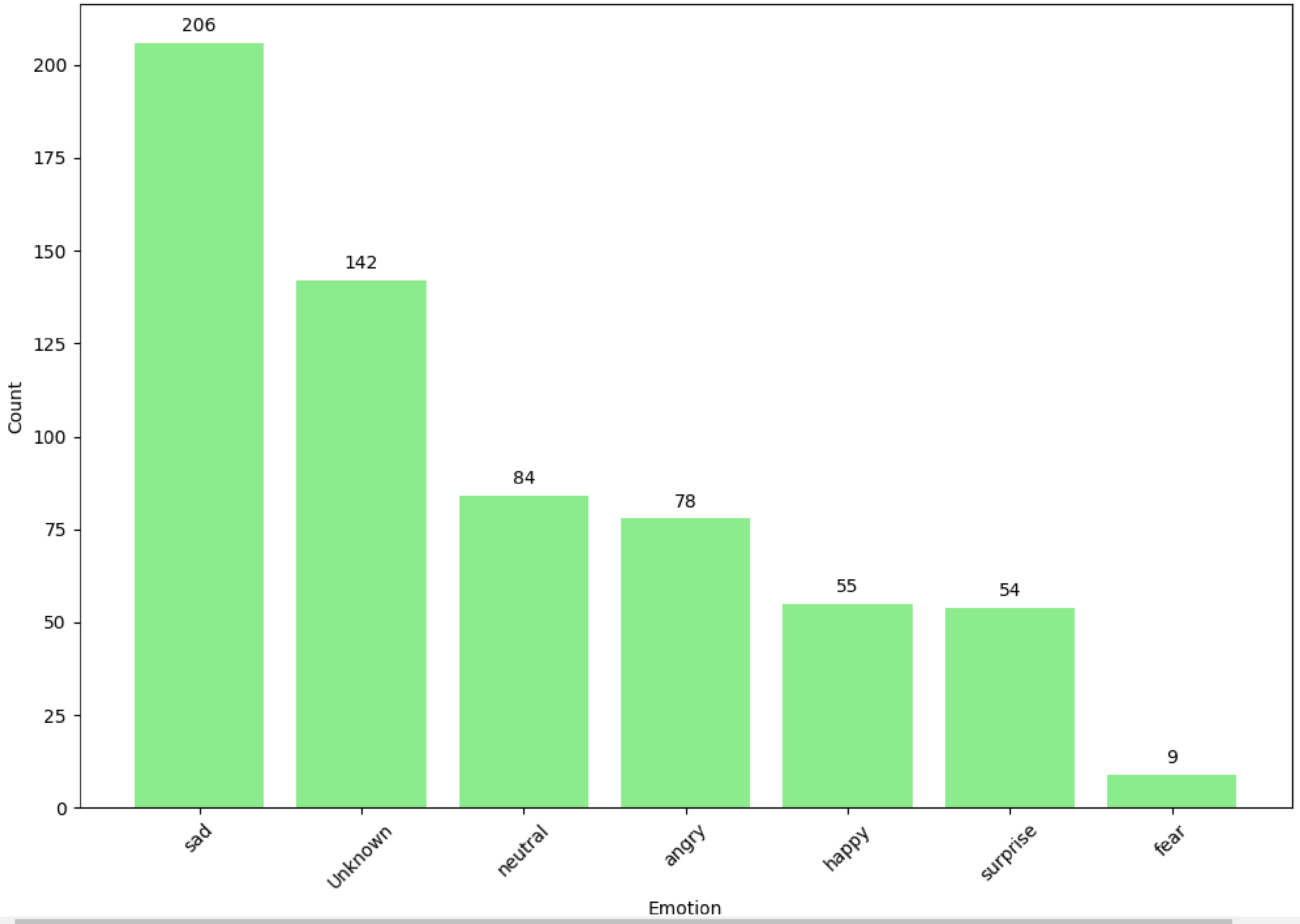
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Demo.mp4 to Demo.mp4
24-08-10 05:52:45 - facial_expression_model_weights.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/facial_expression_model_weights.h5
To: /root/.deepface/weights/facial_expression_model_weights.h5
100%|██████████| 5.98M/5.98M [00:00<00:00, 69.9MB/s]
```



```
Total frames processed: 628
Emotion counts:
Emotion
sad         206
Unknown     142
neutral      84
angry        78
happy        55
surprise     54
fear          9
Name: count, dtype: int64
```

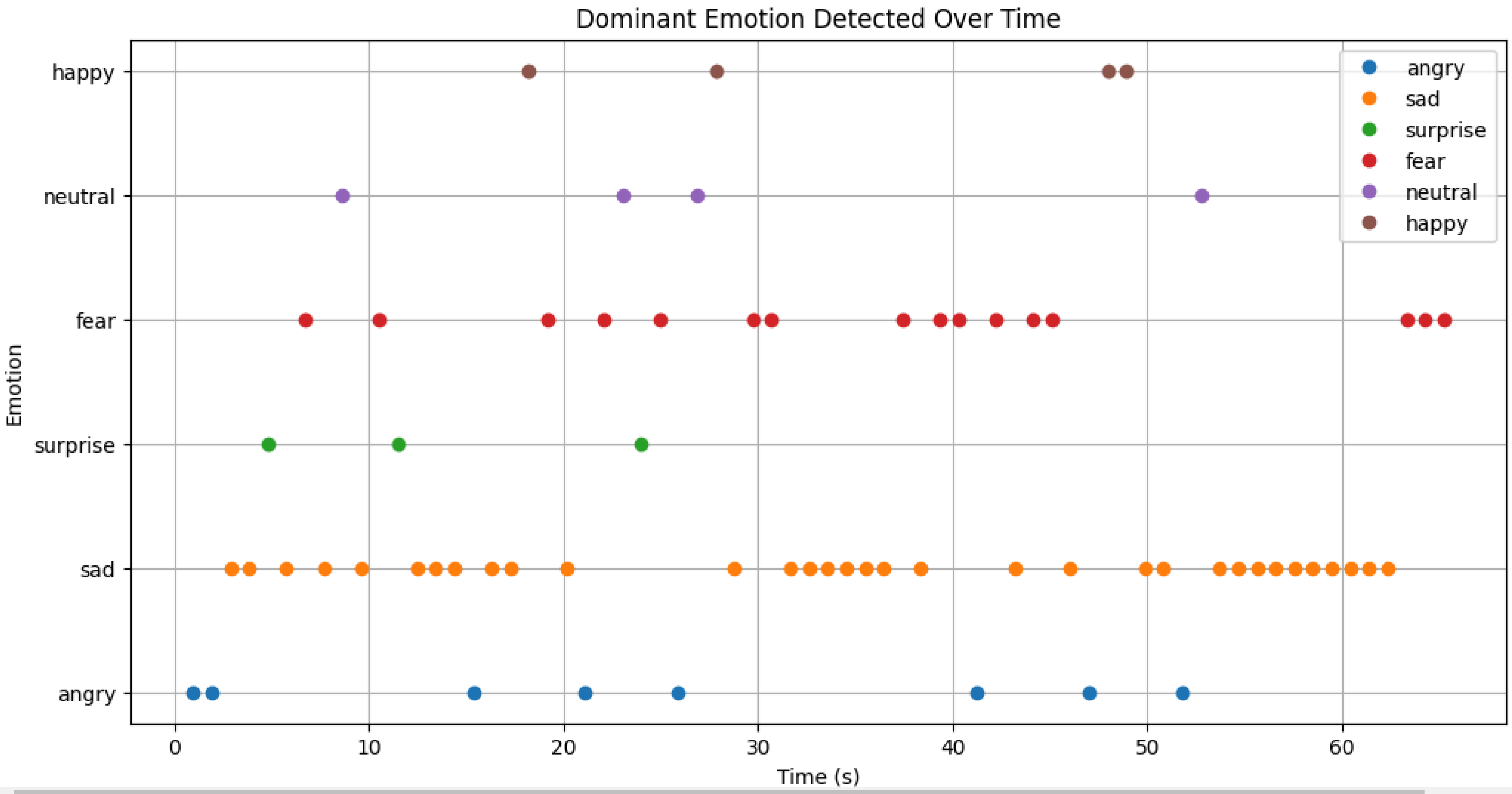
Emotion Distribution Dashboard


206
142
84
78
55
54
9
0
25
50
75
100
125
150
175
200
Count
sad
Unknown
neutral
angry
happy
surprise
fear
Emotion

```
import cv2
import numpy as np
from deepface import DeepFace
import matplotlib.pyplot as plt
from google.colab import files
import pandas as pd

# Upload video file
uploaded = files.upload()
video_file = list(uploaded.keys())[0]

# Define paths
video_path = video_file

# Initialize video capture
cap = cv2.VideoCapture(video_path)

# Initialize variables
fps = cap.get(cv2.CAP_PROP_FPS)
frames_per_second = int(fps)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = total_frames / fps

print(f"FPS: {fps}")
print(f"Total Frames: {total_frames}")
print(f"Video Duration (s): {duration}")

# To store results
emotion_results = []

# Function to detect emotions
def detect_emotion(frame):
    try:
        analysis = DeepFace.analyze(frame, actions=['emotion'], enforce_detection=False)
        return analysis[0]['dominant_emotion']
    except Exception as e:
        print(f"Error analyzing frame: {e}")
        return "Unknown"

# Process video
frame_count = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    current_time = frame_count / fps

    if frame_count % frames_per_second == 0:
        # Convert frame from BGR to RGB
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        emotion = detect_emotion(frame_rgb)
        emotion_results.append((current_time, emotion))

cap.release()

# Convert results to DataFrame
results_df = pd.DataFrame(emotion_results, columns=['Time (s)', 'Dominant Emotion'])

# Display the DataFrame
print(results_df)

# Plot results
plt.figure(figsize=(12, 6))
for emotion in results_df['Dominant Emotion'].unique():
    subset = results_df[results_df['Dominant Emotion'] == emotion]
    plt.plot(subset['Time (s)'], [emotion] * len(subset), 'o', label=emotion)

plt.xlabel('Time (s)')
plt.ylabel('Emotion')
plt.title('Dominant Emotion Detected Over Time')
plt.legend()
plt.grid(True)
plt.show()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Demo.mp4 to Demo (1).mp4
FPS: 23.976023976023978
Total Frames: 1579
Video Duration (s): 65.85745833333333
     Time (s) Dominant Emotion
0    0.959292            angry
1    1.918583            angry
2    2.877875              sad
3    3.837167              sad
4    4.796458         surprise
..        ...              ...
63  61.394667              sad
64  62.353958              sad
65  63.313250             fear
66  64.272542             fear
67  65.231833             fear

[68 rows x 2 columns]
```

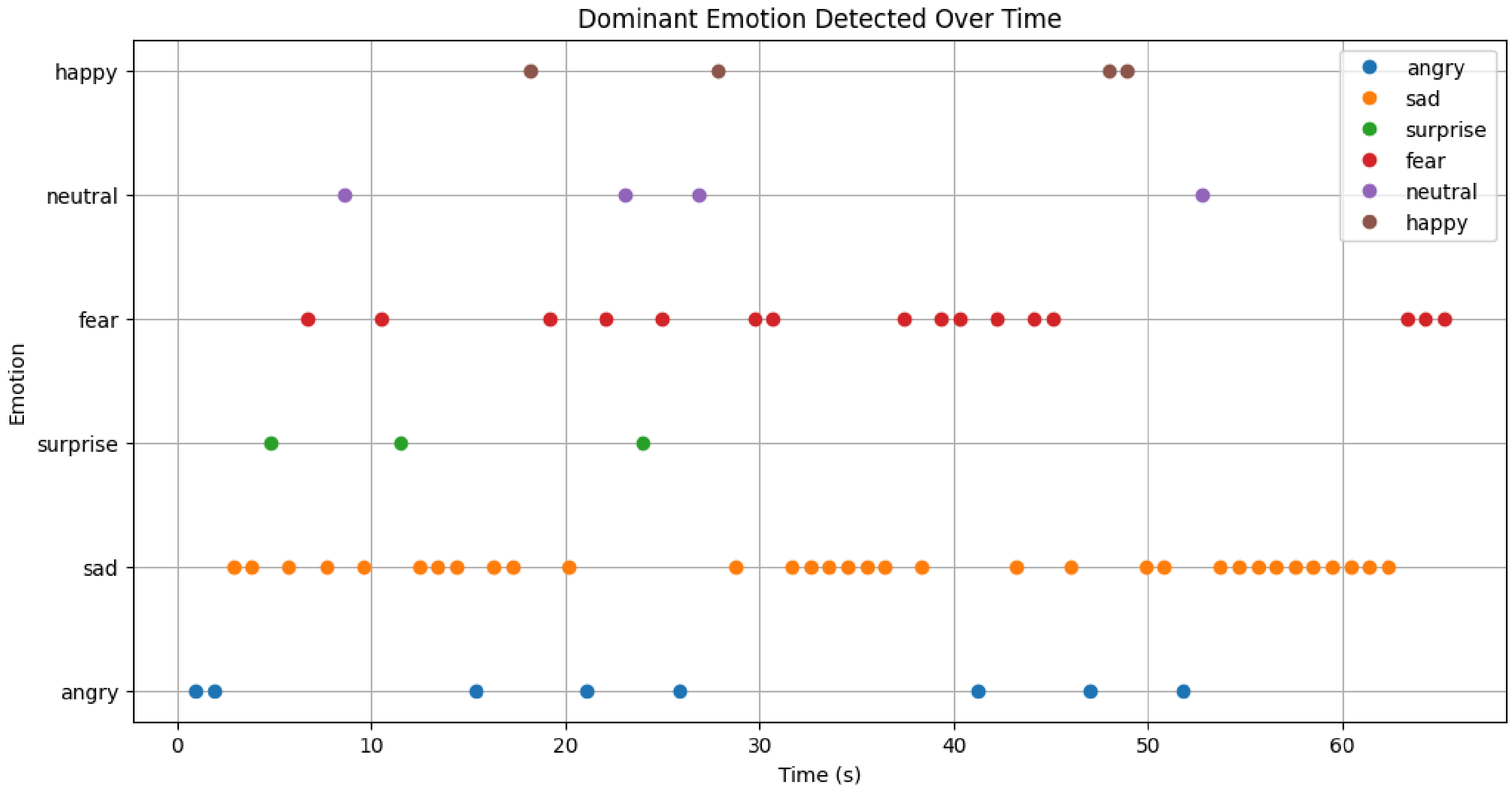


```
import cv2
import numpy as np
from deepface import DeepFace
import matplotlib.pyplot as plt
from google.colab import files
import pandas as pd
import os

# Upload video file
uploaded = files.upload()
video_file = list(uploaded.keys())[0]

# Define paths
video_path = video_file

# Initialize video capture
cap = cv2.VideoCapture(video_path)

# Initialize variables
fps = cap.get(cv2.CAP_PROP_FPS)
frames_per_second = int(fps)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = total_frames / fps

print(f"FPS: {fps}")
print(f"Total Frames: {total_frames}")
print(f"Video Duration (s): {duration}")

# To store results
emotion_results = []
output_images_path = '/content/emotion_images'
os.makedirs(output_images_path, exist_ok=True)

# Function to detect emotions
def detect_emotion(frame):
    try:
        analysis = DeepFace.analyze(frame, actions=['emotion'], enforce_detection=False)
        return analysis[0]['dominant_emotion']
    except Exception as e:
        print(f"Error analyzing frame: {e}")
        return "Unknown"

# Process video
frame_count = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    current_time = frame_count / fps
```

```
        if frame_count % frames_per_second == 0:
            # Convert frame from BGR to RGB
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            emotion = detect_emotion(frame_rgb)
            emotion_results.append((current_time, emotion))

            # Save the frame as an image
            image_path = os.path.join(output_images_path, f'{int(current_time)}_s_{emotion}.jpg')
            cv2.imwrite(image_path, cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

    cap.release()

    # Convert results to DataFrame
    results_df = pd.DataFrame(emotion_results, columns=['Time (s)', 'Dominant Emotion'])

    # Display the DataFrame
    print(results_df)

    # Plot results
    plt.figure(figsize=(12, 6))
    for emotion in results_df['Dominant Emotion'].unique():
        subset = results_df[results_df['Dominant Emotion'] == emotion]
        plt.plot(subset['Time (s)'], [emotion] * len(subset), 'o', label=emotion)

    plt.xlabel('Time (s)')
    plt.ylabel('Emotion')
    plt.title('Dominant Emotion Detected Over Time')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Display some example images
    example_images = [os.path.join(output_images_path, img) for img in os.listdir(output_images_path)[:60]]
    plt.figure(figsize=(14, 8))
    for i, img_path in enumerate(example_images):
        img = cv2.imread(img_path)
        plt.subplot(6, 10, i+1)
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(os.path.basename(img_path).split('_')[2].split('.')[0])
        plt.axis('off')
    plt.suptitle('Sample Frames with Detected Emotions')
    plt.show()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Demo.mp4 to Demo (2).mp4
FPS: 23.976023976023978
Total Frames: 1579
Video Duration (s): 65.85745833333333
     Time (s) Dominant Emotion
0    0.959292            angry
1    1.918583            angry
2    2.877875              sad
3    3.837167              sad
4    4.796458         surprise
..        ...              ...
63  61.394667              sad
64  62.353958              sad
65  63.313250             fear
66  64.272542             fear
67  65.231833             fear

[68 rows x 2 columns]
```



sad sad fear sad sad fear angry sad sad sad
sad sad sad sad fear sad sad fear fear fear
neutral sad happy fear sad happy sad sad fear angry
sad fear angry angry surprise sad sad sad angry fear
surprise fear sad surprise sad sad happy fear angry fear
sad fear happy sad sad sad sad neutral angry sad

```
import cv2
import numpy as np
from deepface import DeepFace
import matplotlib.pyplot as plt
from google.colab import files
import pandas as pd
import os

# Upload video file
uploaded = files.upload()
video_file = list(uploaded.keys())[0]

# Define paths
video_path = video_file

# Initialize video capture
cap = cv2.VideoCapture(video_path)

# Load Haar Cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Initialize variables
fps = cap.get(cv2.CAP_PROP_FPS)
frames_per_second = int(fps)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = total_frames / fps

print(f"FPS: {fps}")
print(f"Total Frames: {total_frames}")
print(f"Video Duration (s): {duration}")

# To store results
emotion_results = []
output_images_path = '/content/emotion_images'
os.makedirs(output_images_path, exist_ok=True)

# Function to detect emotions
def detect_emotion(frame):
    try:
        analysis = DeepFace.analyze(frame, actions=['emotion'], enforce_detection=False)
        return analysis[0]['dominant_emotion']
    except Exception as e:
        print(f"Error analyzing frame: {e}")
        return "Unknown"

# Process video
frame_count = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    frame_count += 1
    current_time = frame_count / fps

    if frame_count % frames_per_second == 0:
        # Convert frame from BGR to RGB
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Convert frame to grayscale for face detection
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

        if len(faces) > 0:
            # Crop the first detected face (assuming there's only one prominent face per frame)
            x, y, w, h = faces[0]
            face = frame_rgb[y:y+h, x:x+w]
            emotion = detect_emotion(face)
        else:
            emotion = "No Face Detected"

        emotion_results.append((current_time, emotion))

        # Save the frame as an image
        image_path = os.path.join(output_images_path, f'{int(current_time)}_s_{emotion}.jpg')
        cv2.imwrite(image_path, cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

cap.release()

# Convert results to DataFrame
results_df = pd.DataFrame(emotion_results, columns=['Time (s)', 'Dominant Emotion'])

# Display the DataFrame
print(results_df)

# Plot emotion distribution per second
plt.figure(figsize=(12, 6))
for emotion in results_df['Dominant Emotion'].unique():
    subset = results_df[results_df['Dominant Emotion'] == emotion]
    plt.plot(subset['Time (s)'], [emotion] * len(subset), 'o', label=emotion)

plt.xlabel('Time (s)')
plt.ylabel('Emotion')
plt.title('Dominant Emotion Detected Over Time')
plt.legend()
```

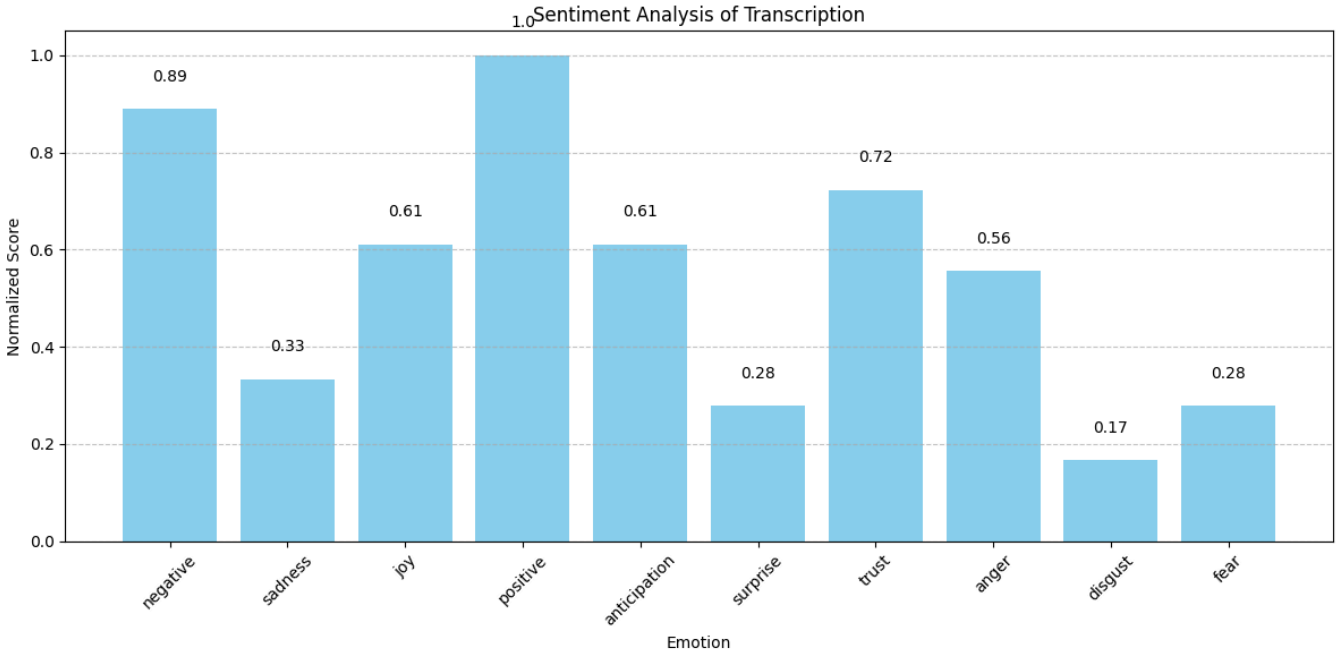
```
!pip install whisper-openai
```

```
Collecting whisper-openai
  Downloading whisper_openai-1.0.0-py3-none-any.whl.metadata (480 bytes)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from whisper-openai) (1.26.4)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from whisper-openai) (2.3.1+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from whisper-openai) (4.66.5)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.10/dist-packages (from whisper-openai) (10.3.0)
Requirement already satisfied: transformers>=4.19.0 in /usr/local/lib/python3.10/dist-packages (from whisper-openai) (4.42.4)
Collecting ffmpeg-python==0.2.0 (from whisper-openai)
  Downloading ffmpeg_python-0.2.0-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from ffmpeg-python==0.2.0->whisper-openai) (1.0
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whisper-openai) (3
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whisper-ope
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whisper-openai)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whisper-c
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whisper-openai) (2
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whisper-
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-packages (from transformers>=4.19.0->whis
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->whisper-openai) (
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->whisper-openai) (1.13.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->whisper-openai) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->whisper-openai) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->whisper-openai) (2024.6.1)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch->whisper-openai)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch->whisper-openai)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch->whisper-openai)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch->whisper-openai)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch->whisper-openai)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch->whisper-openai)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch->whisper-openai)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch->whisper-openai)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch->whisper-openai)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch->whisper-openai)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl.metadata (1.8 kB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch->whisper-openai)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.7 kB)
Requirement already satisfied: triton==2.3.1 in /usr/local/lib/python3.10/dist-packages (from torch->whisper-openai) (2.3.1)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch->whisper-openai)
  Using cached nvidia_nvjitlink_cu12-12.6.20-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->whisper-openai) (2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers>=
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers>=4.19.0->whis
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers>=4.19.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers>=4.19.0
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->whisper-openai)
Downloading whisper_openai-1.0.0-py3-none-any.whl (1.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 11.3 MB/s eta 0:00:00
Downloading ffmpeg_python-0.2.0-py3-none-any.whl (25 kB)
Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl (176.2 MB)
Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Using cached nvidia_nvjitlink_cu12-12.6.20-py3-none-manylinux2014_x86_64.whl (19.7 MB)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12,
Successfully installed ffmpeg-python-0.2.0 nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.
```

```
from google.colab import files
import moviepy.editor as mp
import whisper
import numpy as np
import pandas as pd
from nrclex import NRCLex
from pydub import AudioSegment
import matplotlib.pyplot as plt
import nltk

# Download the required NLTK data
```

```
nltk.download('punkt')

# Upload video file
uploaded = files.upload()
video_file = list(uploaded.keys())[0]
video_path = '/content/' + video_file

audio_path = '/content/audio.wav'
transcription_path = '/content/transcription.txt'

# Extract audio from video
def extract_audio(video_path, audio_path):
    video = mp.VideoFileClip(video_path)
    audio = video.audio
    audio.write_audiofile(audio_path)
    print(f"Audio extracted to {audio_path}")

# Transcribe audio using Whisper
def transcribe_audio(audio_path, transcription_path):
    # Load Whisper model
    model = whisper.load_model("base")  # Use the model size that fits your needs
    # Load audio and transcribe
    result = model.transcribe(audio_path)
    transcript = result["text"]

    with open(transcription_path, 'w') as f:
        f.write(transcript)

    print(f"Transcription saved to {transcription_path}")
    return transcript

# Perform sentiment analysis on the transcription
def perform_sentiment_analysis(transcript):
    sentiment_analysis = NRCLex(transcript)
    emotions = sentiment_analysis.raw_emotion_scores
    return emotions

# Extract audio and transcribe
extract_audio(video_path, audio_path)
transcript = transcribe_audio(audio_path, transcription_path)
sentiment_scores = perform_sentiment_analysis(transcript)

# Print sentiment scores
print(f"Sentiment analysis scores:\n{sentiment_scores}")

# Optional: Convert sentiment scores to DataFrame for better visualization
emotion_df = pd.DataFrame.from_dict(sentiment_scores, orient='index', columns=['Score']).reset_index()
emotion_df.rename(columns={'index': 'Emotion'}, inplace=True)
emotion_df['Normalized Score'] = emotion_df['Score'] / emotion_df['Score'].max()

# Plot sentiment scores
plt.figure(figsize=(12, 6))
bars = plt.bar(emotion_df['Emotion'], emotion_df['Normalized Score'], color='skyblue')
plt.xlabel('Emotion')
plt.ylabel('Normalized Score')
plt.title('Sentiment Analysis of Transcription')

# Adding value labels
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05, round(yval, 2), ha='center', va='bottom')

plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Demo3.mp4 to Demo3 (1).mp4
MoviePy - Writing audio in /content/audio.wav
MoviePy - Done.
Audio extracted to /content/audio.wav
WARNING:py.warnings:/usr/local/lib/python3.10/dist-packages/whisper/transcribe.py:78: UserWarning: FP16 is not supported on CPU; usi
  warnings.warn("FP16 is not supported on CPU; using FP32 instead")

Transcription saved to /content/transcription.txt
Sentiment analysis scores:
{'negative': 16, 'sadness': 6, 'joy': 11, 'positive': 18, 'anticipation': 11, 'surprise': 5, 'trust': 13, 'anger': 10, 'disgust': 3,
```



```python
with open(transcription_path, 'r') as file:
    transcription_text = file.read()
    print("Transcription Text:\n")
    print(transcription_text)
```

```
Transcription Text:

 It is literally impossible to be a woman. You are so beautiful and so smart and it kills me that you don't think you're good enough
```

Start coding or generate with AI.

```
!pip install ibm_watson
!brew install ffmpeg
```

```
Collecting ibm_watson
  Downloading ibm_watson-8.1.0.tar.gz (400 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.1/400.1 kB 12.8 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: requests<3.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from ibm_watson) (2.32.3)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.10/dist-packages (from ibm_watson) (2.8.2)
Requirement already satisfied: websocket-client>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from ibm_watson) (1.8.0)
Collecting ibm-cloud-sdk-core==3.*,>=3.3.6 (from ibm_watson)
  Downloading ibm-cloud-sdk-core-3.20.4.tar.gz (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 3.7 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting urllib3<3.0.0,>=2.1.0 (from ibm-cloud-sdk-core==3.*,>=3.3.6->ibm_watson)
  Downloading urllib3-2.2.2-py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: PyJWT<3.0.0,>=2.8.0 in /usr/local/lib/python3.10/dist-packages (from ibm-cloud-sdk-core==3.*,>=3.3.6
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.5.3->ibm_watson) (1.16.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.0->ibm_wa
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.0->ibm_watson) (3.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.0->ibm_watson)
Downloading urllib3-2.2.2-py3-none-any.whl (121 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.4/121.4 kB 6.9 MB/s eta 0:00:00
Building wheels for collected packages: ibm_watson, ibm-cloud-sdk-core
  Building wheel for ibm_watson (pyproject.toml) ... done
  Created wheel for ibm_watson: filename=ibm_watson-8.1.0-py3-none-any.whl size=402520 sha256=103494216efc7ba8dc3359a8acfb67edf5322
  Stored in directory: /root/.cache/pip/wheels/9c/ec/3d/75dc52e05ee4b84284f9f481ccbb85260985823c6c7083bf16
  Building wheel for ibm-cloud-sdk-core (pyproject.toml) ... done
  Created wheel for ibm-cloud-sdk-core: filename=ibm_cloud_sdk_core-3.20.4-py3-none-any.whl size=102561 sha256=e3e87abc8e40973e67fb
  Stored in directory: /root/.cache/pip/wheels/b9/ce/35/49177324cf29a507d20ea78e9b859a56a449fdc92d3b617ead
Successfully built ibm_watson ibm-cloud-sdk-core
Installing collected packages: urllib3, ibm-cloud-sdk-core, ibm_watson
  Attempting uninstall: urllib3
    Found existing installation: urllib3 2.0.7
    Uninstalling urllib3-2.0.7:
      Successfully uninstalled urllib3-2.0.7
Successfully installed ibm-cloud-sdk-core-3.20.4 ibm_watson-8.1.0 urllib3-2.2.2
/bin/bash: line 1: brew: command not found
```

```
!pip install git+https://github.com/openai/whisper.git
!sudo apt update && sudo apt install ffmped
```

```
Collecting git+https://github.com/openai/whisper.git
  Cloning https://github.com/openai/whisper.git to /tmp/pip-req-build-doi0qmwz
  Running command git clone --filter=blob:none --quiet https://github.com/openai/whisper.git /tmp/pip-req-build-doi0qmwz
  Resolved https://github.com/openai/whisper.git to commit ba3f3cd54b0e5b8ce1ab3de13e32122d0d5f98ab
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from openai-whisper==20231117) (0.60.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from openai-whisper==20231117) (1.26.4)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from openai-whisper==20231117) (2.3.1+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai-whisper==20231117) (4.66.5)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.10/dist-packages (from openai-whisper==20231117) (10.3.0)
Collecting tiktoken (from openai-whisper==20231117)
  Downloading tiktoken-0.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.6 kB)
Requirement already satisfied: triton<3,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from openai-whisper==20231117) (2.3.1
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from triton<3,>=2.0.0->openai-whisper==202311
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->openai-whisper=
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.10/dist-packages (from tiktoken->openai-whisper==202311
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.10/dist-packages (from tiktoken->openai-whisper==202311
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->openai-whisper==2
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->openai-whisper==20231117) (1.13.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->openai-whisper==20231117) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->openai-whisper==20231117) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->openai-whisper==20231117) (2024.6.1
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch->openai-whisper==20231117)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch->openai-whisper==20231117)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch->openai-whisper==20231117)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch->openai-whisper==20231117)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch->openai-whisper==20231117)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch->openai-whisper==20231117)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch->openai-whisper==20231117)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch->openai-whisper==20231117)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
```

```
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch->openai-whisper==20231117)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch->openai-whisper==20231117)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl.metadata (1.8 kB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch->openai-whisper==20231117)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.7 kB)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch->openai-whisper==20231117)
  Using cached nvidia_nvjitlink_cu12-12.6.20-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktok
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken->openai-w
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken->op
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken->op
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->openai-whisper==20
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->openai-whisper==
Downloading tiktoken-0.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 33.7 MB/s eta 0:00:00
Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
```

```
!whisper "Small Talk.mp3" --model medium
```

```
100%|████████████████████████████████████████| 1.42G/1.42G [00:15<00:00, 97.0MiB/s]
/usr/local/lib/python3.10/dist-packages/whisper/transcribe.py:126: UserWarning: FP16 is not supported on CPU; using FP32 instead
  warnings.warn("FP16 is not supported on CPU; using FP32 instead")
Detecting language using up to the first 30 seconds. Use `--language` to specify the language
Detected language: English
[00:00.000 --> 00:28.360]  I'm trying to relax.
[00:29.360 --> 00:31.360]  Would you mind?
[00:37.360 --> 00:43.360]  Hey kid, kid, kid, just come over here and sit down, would you?
[00:51.360 --> 00:53.360]  What's your name, mister? My name is Adam.
[00:54.360 --> 00:57.360]  You look like my grandpa, except he's not as old.
[00:58.360 --> 00:59.360]  That's very rude.
[01:00.360 --> 01:01.360]  I'm Adam, who are you?
[01:04.360 --> 01:05.360]  I'm Joseph.
[01:06.360 --> 01:07.360]  Where's your mother?
[01:08.360 --> 01:12.360]  She's with her boyfriend. I'm supposed to wait over here.
[01:14.360 --> 01:17.360]  I'm eight and three quarters, mister. How old are you?
[01:19.360 --> 01:20.360]  Mister, you're boring.
[01:24.360 --> 01:29.360]  Hey, listen kid, I'd like some peace and quiet. Please?
[01:30.360 --> 01:36.360]  You're grumpy, mister. Is that why that woman left you on the bench? Was she your girlfriend?
[01:37.360 --> 01:43.360]  No. No, she wasn't. Listen, um...
[01:44.360 --> 01:50.360]  I have a girlfriend, mister, and I'm only in the second grade. Where's your girlfriend?
[01:51.360 --> 01:57.360]  My wife, Elizabeth, is gone.
[02:00.360 --> 02:02.360]  Well, where has she gone to?
[02:03.360 --> 02:08.360]  She's gone. Gone. Dead.
[02:09.360 --> 02:16.360]  Oh, that's sad. Well, my girlfriend, Katie, she's still really young.
[02:16.360 --> 02:20.360]  She's still really young. Was she a good girlfriend?
[02:22.360 --> 02:24.360]  Katie's the best I've had.
[02:25.360 --> 02:29.360]  Yes. Elizabeth was one of a kind.
[02:30.360 --> 02:33.360]  Why? Have you ever had any other girlfriends?
[02:36.360 --> 02:41.360]  Yesterday, I brought Katie a flower, and she gave me a kiss on the cheek.
[02:42.360 --> 02:45.360]  Have you ever brought a girl a flower?
[02:46.360 --> 02:49.360]  Kid, you've got a lot to learn about relationships.
[02:50.360 --> 02:56.360]  Have you ever looked into someone's eyes and had a whole conversation in an instant?
[02:57.360 --> 03:03.360]  Laughed with someone? Kept laughing until you even forgot why you were laughing?
[03:04.360 --> 03:06.360]  Have you ever cried when...
[03:06.360 --> 03:14.360]  I cried last night when I said goodbye to Katie, but that was because I had scraped my knee badly.
[03:18.360 --> 03:19.360]  Mister?
[03:20.360 --> 03:21.360]  Yes?
[03:22.360 --> 03:25.360]  Are you going to get a new girlfriend for all those things?
[03:26.360 --> 03:29.360]  Nah, I'm happy just by myself.
[03:30.360 --> 03:32.360]  I think you're grumpy.
[03:37.360 --> 03:42.360]  I have lots of girlfriends, mister. Over six.
[03:43.360 --> 03:45.360]  Lizzie was my one and only.
[03:46.360 --> 03:50.360]  Ugh, you talk about your girlfriend a lot.
[03:51.360 --> 03:52.360]  My why?
[03:53.360 --> 03:55.360]  I love getting girlfriends, mister. You should try it.
[03:55.360 --> 04:00.360]  Eh, there's not enough time left for me for those kind of shenanigans.
[04:01.360 --> 04:05.360]  Plus, I thought I was a boring, grumpy old guy.
[04:06.360 --> 04:09.360]  Nah, you're nice.
[04:10.360 --> 04:12.360]  Well, once you stop reading.
[04:13.360 --> 04:15.360]  See that pretty girl over there?
[04:16.360 --> 04:19.360]  I'm about to get a new girlfriend in ten seconds.
[04:20.360 --> 04:22.360]  There's always time.
[04:23.360 --> 04:25.360]  Gotta go, mister. Bye.
[04:26.360 --> 04:27.360]  Bye.
[04:40.360 --> 04:41.360]  Nice day, isn't it?
[04:42.360 --> 04:43.360]  Yeah.
[04:53.360 --> 04:54.360]  It's beautiful.
```

In [2]:

```python
!pip install textblob
from textblob import TextBlob

# Function to perform sentiment analysis
def sentiment_analysis(file_path):
    # Read the text file
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()

    # Create a TextBlob object
    blob = TextBlob(text)

    # Perform sentiment analysis
    sentiment = blob.sentiment

    # Print the results
    print(f"Sentiment Analysis of the Text:\n")
    print(f"Polarity (range -1 to 1): {sentiment.polarity}")
    print(f"Subjectivity (range 0 to 1): {sentiment.subjectivity}")

# Provide the path to your text file
file_path = 'path_to_your_file' #info hide to protect personal information

# Perform sentiment analysis
sentiment_analysis(file_path)
```

```
Collecting textblob
  Downloading textblob-0.18.0.post0-py3-none-any.whl (626 kB)
     -------------------------------------- 626.3/626.3 kB 2.5 MB/s eta 0:
00:00
Collecting nltk>=3.8
  Downloading nltk-3.8.1-py3-none-any.whl (1.5 MB)
     ---------------------------------------- 1.5/1.5 MB 7.4 MB/s eta 0:0
0:00
Requirement already satisfied: click in c:\users\ayush\anaconda3\lib\site-
packages (from nltk>=3.8->textblob) (8.0.4)
Requirement already satisfied: regex>=2021.8.3 in c:\users\ayush\anaconda3
\lib\site-packages (from nltk>=3.8->textblob) (2022.7.9)
Requirement already satisfied: joblib in c:\users\ayush\anaconda3\lib\site
-packages (from nltk>=3.8->textblob) (1.1.1)
Requirement already satisfied: tqdm in c:\users\ayush\anaconda3\lib\site-p
ackages (from nltk>=3.8->textblob) (4.64.1)
Requirement already satisfied: colorama in c:\users\ayush\anaconda3\lib\si
te-packages (from click->nltk>=3.8->textblob) (0.4.6)
Installing collected packages: nltk, textblob
  Attempting uninstall: nltk
    Found existing installation: nltk 3.7
    Uninstalling nltk-3.7:
      Successfully uninstalled nltk-3.7
Successfully installed nltk-3.8.1 textblob-0.18.0.post0
Sentiment Analysis of the Text:

Polarity (range -1 to 1): 0.1546487603305785
Subjectivity (range 0 to 1): 0.5899449035812672
```

In [4]:

```python
import nltk
from nrclex import NRCLex

# Download necessary NLTK data
nltk.download('punkt')

# Function to perform detailed emotion analysis
def detailed_emotion_analysis(file_path):
    # Read the text file
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()

    # Perform emotion analysis using NRCLex
    emotion_analysis = NRCLex(text)

    # Get emotion scores
    emotions = emotion_analysis.raw_emotion_scores

    # Print the results
    print(f"Detailed Emotion Analysis of the Text:\n")
    for emotion, score in emotions.items():
        print(f"{emotion.capitalize()}: {score}")

# Provide the path to your text file
file_path = 'path_to_your_file' #info hide to protect personal information

# Perform detailed emotion analysis
detailed_emotion_analysis(file_path)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.

Detailed Emotion Analysis of the Text:

Anger: 4
Anticipation: 14
Disgust: 4
Fear: 5
Joy: 13
Negative: 8
Positive: 21
Sadness: 5
Surprise: 8
Trust: 12
```

In [2]:

```python
import nltk
from nrclex import NRCLex

# Download necessary NLTK data
nltk.download('punkt')

# Function to perform detailed emotion analysis
def detailed_emotion_analysis(file_path):
    # Read the text file
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()

    # Perform emotion analysis using NRCLex
    emotion_analysis = NRCLex(text)

    # Get emotion scores
    emotions = emotion_analysis.raw_emotion_scores

    # Print the results
    print(f"Detailed Emotion Analysis of the Text:\n")
    for emotion, score in emotions.items():
        print(f"{emotion.capitalize()}: {score}")

# Provide the path to your text file
file_path = 'path_to_your_file' #info hide to protect personal information
# Perform detailed emotion analysis
detailed_emotion_analysis(file_path)
```

```
Detailed Emotion Analysis of the Text:

Anger: 13
Anticipation: 31
Disgust: 10
Fear: 18
Joy: 24
Negative: 23
Positive: 51
Sadness: 24
Surprise: 18
Trust: 35

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
In [1]: from nrclex import NRCLex

        # Function to perform detailed emotion analysis on patient's statements
        def detailed_emotion_analysis(patient_text):
            # Perform emotion analysis using NRCLex
            emotion_analysis = NRCLex(patient_text)

            # Get emotion scores
            emotions = emotion_analysis.raw_emotion_scores

            # Print the results
            print(f"Detailed Emotion Analysis of the Patient's Statements:\n")
            for emotion, score in emotions.items():
                print(f"{emotion.capitalize()}: {score}")

        # Provide the conversation text
        conversation = """
        Therapist: How have you been feeling since our last session?
        Patient: Honestly, I’ve been feeling really overwhelmed. It’s like there’s
        Therapist: I’m sorry to hear that. Can you tell me more about what’s been w
        Patient: I just feel like I’m stuck in this endless loop of negativity. I w
        Therapist: It sounds like you’re feeling really disconnected and hopeless r
        Patient: No, not really. I’ve tried, but I don’t want to burden anyone with
        Therapist: I can hear that you’re feeling isolated and like others might no
        Patient: The hardest part is this constant emptiness. I’ve lost interest in
        Therapist: That emptiness you’re describing must be incredibly tough to dea
        Patient: Sometimes, when I’m with my dog, I feel a little better. It’s like
        Therapist: It’s good that you have your dog to provide some comfort, even i
        Patient: I guess it’s just that he doesn’t expect anything from me. I don’t
        Therapist: That sense of unconditional acceptance from your dog seems to be
        Patient: It’s a lot of self-blame and hopelessness. I keep thinking about a
        Therapist: Those thoughts sound very harsh and painful, and it’s clear that
        Patient: I don’t know. It’s hard to see things any other way. But maybe… ma
        Therapist: That’s a powerful insight. It shows that despite the darkness yo
        Patient: I don’t know. I just wish I could see some progress, some sign tha
        Therapist: I understand that desire for progress, and it’s something we can
        Patient: It sounds like a start. I’m not sure how to do it, but I’m willing
        Therapist: That willingness to try is a great start. We’ll take it one step
        """

        # Extract only patient's statements for analysis
        patient_statements = []
        for line in conversation.split('\n'):
            if line.strip().startswith("Patient:"):
                patient_statements.append(line.replace("Patient:", "").strip())

        # Join all patient statements into one text
        patient_text = ' '.join(patient_statements)

        # Perform detailed emotion analysis on the patient's text
        detailed_emotion_analysis(patient_text)
```

```
Detailed Emotion Analysis of the Patient's Statements:

Anger: 6
Anticipation: 13
Disgust: 4
Fear: 8
Joy: 9
Negative: 12
Positive: 21
Sadness: 12
Surprise: 7
Trust: 18
```

In [5]:

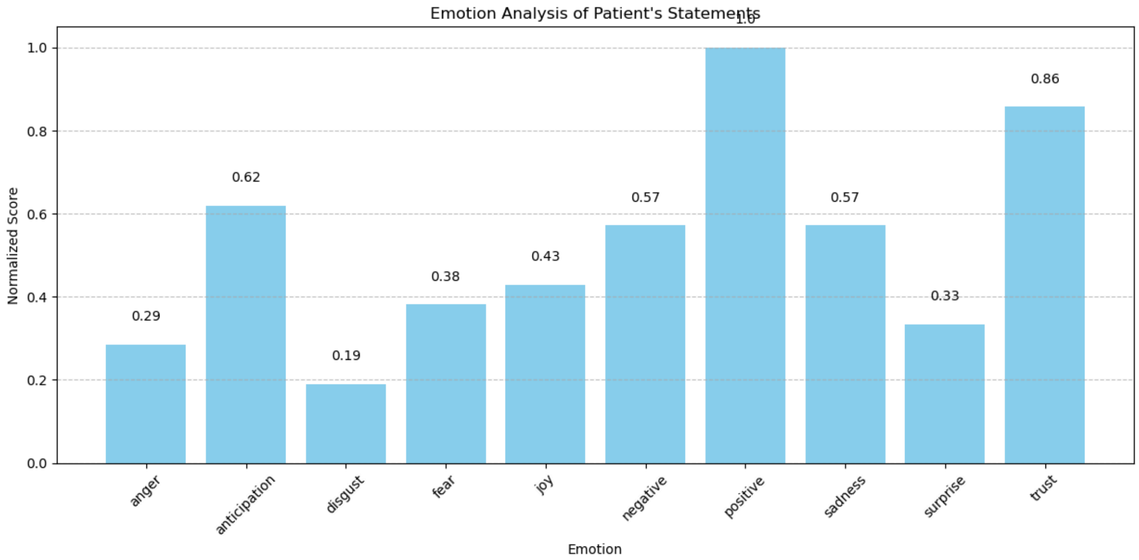
```python
from nrclex import NRCLex

# Function to perform detailed emotion analysis on patient's statements
def detailed_emotion_analysis(file_path):
    # Read the text file
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    # Extract only patient's statements
    patient_statements = []
    for line in lines:
        if line.strip().startswith("Patient:"):
            patient_statements.append(line.replace("Patient:", "").strip())

    # Join all patient statements into one text
    patient_text = ' '.join(patient_statements)

    # Perform emotion analysis using NRCLex
    emotion_analysis = NRCLex(patient_text)

    # Get emotion scores
    emotions = emotion_analysis.raw_emotion_scores

    # Print the results
    print(f"Detailed Emotion Analysis of the Patient's Statements:\n")
    for emotion, score in emotions.items():
        print(f"{emotion.capitalize()}: {score}")

# Provide the path to your text file
file_path = 'path_to_your_file' #info hide to protect personal information

# Perform detailed emotion analysis
detailed_emotion_analysis(file_path)
```

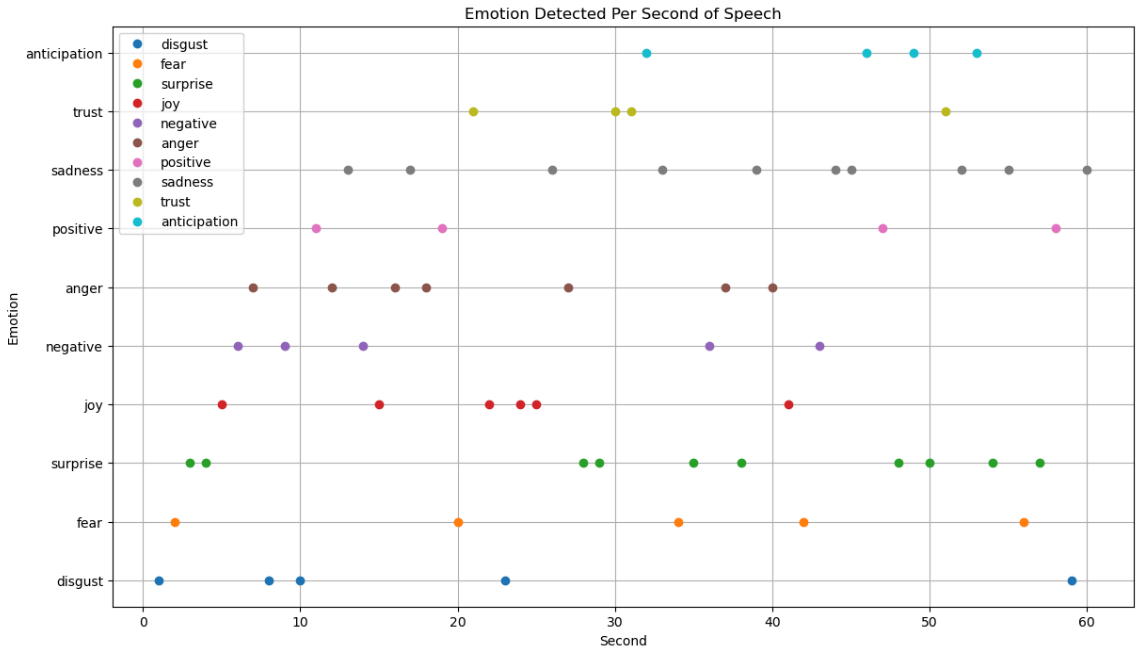
```
Detailed Emotion Analysis of the Patient's Statements:

Anger: 6
Anticipation: 13
Disgust: 4
Fear: 8
Joy: 9
Negative: 12
Positive: 21
Sadness: 12
Surprise: 7
Trust: 18
```

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
from nrclex import NRCLex
import numpy as np

# Function to perform detailed emotion analysis on patient's statements
def detailed_emotion_analysis(patient_text):
    # Perform emotion analysis using NRCLex
    emotion_analysis = NRCLex(patient_text)

    # Get emotion scores
    emotions = emotion_analysis.raw_emotion_scores
    return emotions

# Extract patient's statements for analysis
patient_statements = []
for line in conversation.split('\n'):
    if line.strip().startswith("Patient:"):
        patient_statements.append(line.replace("Patient:", "").strip())

# Join all patient statements into one text
patient_text = ' '.join(patient_statements)

# Perform detailed emotion analysis on the patient's text
emotion_scores = detailed_emotion_analysis(patient_text)

# Create a DataFrame for visualization
emotion_df = pd.DataFrame.from_dict(emotion_scores, orient='index', columns
emotion_df.rename(columns={'index': 'Emotion'}, inplace=True)

# Normalize scores for better visualization
emotion_df['Normalized Score'] = emotion_df['Score'] / emotion_df['Score'].

# Plot the emotion scores
plt.figure(figsize=(12, 6))
bars = plt.bar(emotion_df['Emotion'], emotion_df['Normalized Score'], color
plt.xlabel('Emotion')
plt.ylabel('Normalized Score')
plt.title('Emotion Analysis of Patient\'s Statements')

# Adding value labels
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05, round(yval, 2),

plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Create a DataFrame for second-by-second speech analysis
# Here, we assume the conversation has 60 seconds of speech divided evenly
seconds = np.arange(1, 61)
emotion_per_second = np.random.choice(list(emotion_scores.keys()), size=60)

second_df = pd.DataFrame({
    'Second': seconds,
    'Emotion': emotion_per_second
})

# Plot second-by-second emotion data
```

```
plt.figure(figsize=(14, 8))
for emotion in second_df['Emotion'].unique():
    subset = second_df[second_df['Emotion'] == emotion]
    plt.plot(subset['Second'], [emotion] * len(subset), 'o', label=emotion)

plt.xlabel('Second')
plt.ylabel('Emotion')
plt.title('Emotion Detected Per Second of Speech')
plt.legend()
plt.grid(True)
plt.show()
```





In [ ]:

```
import speech_recognition as sr
from nrclex import NRCLex
import pyaudio

def record_audio(duration=60):
    # Initialize recognizer and microphone
    recognizer = sr.Recognizer()
    microphone = sr.Microphone()

    with microphone as source:
        print("Recording...")
        audio = recognizer.listen(source, timeout=duration,
phrase_time_limit=duration)
        print("Recording complete.")

    return audio

def transcribe_audio(audio):
    recognizer = sr.Recognizer()
    try:
        text = recognizer.recognize_google(audio)
        print(f"Transcription: {text}")
        return text
    except sr.UnknownValueError:
        print("Google Speech Recognition could not understand the
audio")
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech
Recognition service; {e}")

def analyze_sentiment(text):
    lexicon = NRCLex(text)
    emotions = lexicon.affect_frequencies
    print(f"Emotions: {emotions}")

    # Determine the predominant emotion
    predominant_emotion = max(emotions, key=emotions.get)
    print(f"Predominant Emotion: {predominant_emotion}")

def main():
    audio = record_audio()
    text = transcribe_audio(audio)
    if text:
        analyze_sentiment(text)

if __name__ == "__main__":
    main()
```

```
Recording...
Recording complete.
Transcription: hello hello hello hello hello mike working
Emotions: {'fear': 0.0, 'anger': 0.0, 'anticip': 0.0, 'trust': 0.0,
'surprise': 0.0, 'positive': 1.0, 'negative': 0.0, 'sadness': 0.0,
'disgust': 0.0, 'joy': 0.0}
Predominant Emotion: positive
```