Name : Hassan Ali

Task # 1

Extract text from handwritten notes or posters shared by users or therapists.

Clean and understand the extracted text using NLP.

Detect emotions or mental health indicators like anxiety, stress, or sadness.

Provide results in a simple dashboard or app for users and counselors.

**********Mental Health Emotion Analyzer**********

These tools will be used in this.

Image & Text Extraction

OpenCV: Preprocess images (e.g., resize, denoise, binarize) to improve OCR accuracy.

Pytesseract: Extract text from handwritten notes or posters using OCR.

Text Cleaning & Understanding

spaCy / NLTK:

Tokenization, lemmatization, stopword removal, POS tagging.

Helps clean and structure the extracted text for analysis.

Emotion & Mental Health Detection

Hugging Face Transformers:

Use pre-trained models (like BERT, RoBERTa) for sentiment/emotion classification.

Detect indicators like stress, anxiety, or sadness in the text.

Scikit-learn:

I don’t use scikit learn because am using in this pretrained model

So that I don’t use

Frontend Display

Streamlit or Flask:

Streamlit: Quick and interactive dashboards for non-tech users.

Flask: Custom web app backend for full-stack integration with APIs or databases.

Coding files

App.py

```
import streamlit as st
from PIL import Image
import numpy as np
import cv2
import time
import matplotlib.pyplot as plt
from ocr_utils import extract_text_from_image
from nlp_clean import clean_text
from emotion_model import detect_emotions

# Initialize the app
st.set_page_config(page_title="Mental Health Emotion Analyzer",
layout="centered")
st.title("Mental Health Emotion Analyzer")
st.markdown("""
Upload handwritten or printed images. This app will:
- Extract text using advanced OCR
- Clean the text using NLP
- Detect emotional indicators (Stress, Anxiety, Sadness, etc.)
- Visualize emotional state
""")

# Sidebar settings
```

```python
with st.sidebar:
    st.header("Settings")
    ocr_mode = st.selectbox(
        "OCR Mode",
        ["Standard", "Enhanced"],
        help="Use 'Enhanced' for handwritten text"
    ).lower()
    show_steps = st.checkbox("Show processing steps", value=True)

def plot_emotions(emotions):
    """Visualize emotion analysis results"""
    fig, ax = plt.subplots(figsize=(10, 4))
    labels = [emo['label'] for emo in emotions]
    scores = [emo['score'] for emo in emotions]

    colors = []
    for label in labels:
        if label.lower() in ['sadness', 'anger', 'fear']:
            colors.append('red')
        elif label.lower() in ['joy', 'love']:
            colors.append('green')
        else:
            colors.append('blue')

    bars = ax.barh(labels, scores, color=colors)
    ax.set_xlim(0, 1)
    ax.set_title('Emotional State Analysis')
    ax.set_xlabel('Confidence Score')

    for bar in bars:
        width = bar.get_width()
        ax.text(width + 0.02, bar.get_y() + bar.get_height()/2,
                f'{width:.2f}',
                ha='left', va='center')

    st.pyplot(fig)

# Main app logic from hassan
uploaded_file = st.file_uploader("Upload an image", type=["jpg", "jpeg", "png"])

if uploaded_file:
    image = Image.open(uploaded_file)
    image_np = np.array(image)
    st.image(image, caption="Uploaded Image", use_column_width=True)
```

```
    if st.button("Extract Text and Analyze"):
        with st.spinner("Processing..."):
            start_time = time.time()

            # Debug output
            st.write("Running OCR in", ocr_mode, "mode...")

            # Extract text
            extracted_text = extract_text_from_image(
                image_np,
                mode=ocr_mode.lower(),
                show_steps=show_steps
            )
            st.subheader("Extracted Text")
            st.text_area("Raw OCR Output", extracted_text, height=150)

            # Clean text
            cleaned_text = clean_text(extracted_text)
            st.subheader("Cleaned Text")
            st.text_area("After NLP Processing", cleaned_text, height=100)

            # Emotion detection
            st.subheader("Emotion Analysis")
            if cleaned_text.strip():
                try:
                    emotions = detect_emotions(cleaned_text)
                    plot_emotions(emotions)

                    # Mental health indicators
                    mental_health_indicators = {
                        'sadness': next((e for e in emotions if
e['label'].lower() == 'sadness'), None),
                        'anxiety': next((e for e in emotions if
e['label'].lower() == 'fear'), None),
                        'stress': next((e for e in emotions if e['label'].lower()
== 'anger'), None)
                    }

                    st.markdown("### Mental Health Indicators")
                    cols = st.columns(3)
                    for i, (name, data) in
enumerate(mental_health_indicators.items()):
                        if data:
                            score = data['score']
```

```
                            color = "red" if score > 0.5 else "orange" if score >
0.3 else "green"
                            cols[i].metric(
                                label=name.capitalize(),
                                value=f"{score*100:.1f}%",
                                delta="High concern" if score > 0.5 else
"Moderate" if score > 0.3 else "Low",
                                delta_color="off"
                            )
                except Exception as e:
                    st.error(f"Emotion detection failed: {str(e)}")
            else:
                st.warning("No meaningful text found for emotion analysis")

            st.success(f"Analysis completed in {time.time()-start_time:.2f}
seconds")

st.sidebar.markdown("---")
st.sidebar.caption("App version 1.0.1")
```

ocr_utils.py:

```
import cv2
import pytesseract
import numpy as np
from PIL import Image
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel

# Set Tesseract path (update if needed)
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-
OCR\tesseract.exe"

# Load TrOCR model and processor (for handwriting)
processor = TrOCRProcessor.from_pretrained("microsoft/trocr-base-handwritten")
model = VisionEncoderDecoderModel.from_pretrained("microsoft/trocr-base-
handwritten")

def preprocess_image(image, mode="standard", show_steps=False):
    """Preprocess image based on OCR mode."""
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

```
    if mode == "enhanced":
        # Best for handwriting
        denoised = cv2.fastNlMeansDenoising(gray, None, 10, 7, 21)
        equalized = cv2.equalizeHist(denoised)
        processed = cv2.adaptiveThreshold(
            equalized, 255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY_INV,
            15, 10
        )
    else:
        # Best for printed images
        blur = cv2.GaussianBlur(gray, (5, 5), 0)
        _, processed = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    if show_steps:
        cv2.imshow("Processed", processed)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return processed

def extract_text_from_image(image, mode="standard", show_steps=False):
    """
    Extract text using OCR.

    Parameters:
        image (np.array): Input image
        mode (str): "standard" = printed text via Tesseract
                    "enhanced" = handwriting via TrOCR
        show_steps (bool): Optional for debugging preprocessing

    Returns:
        str: Extracted text
    """
    if mode == "enhanced":
        # Use TrOCR for handwritten text
        try:
            pil_image = Image.fromarray(image).convert("RGB")
            pixel_values = processor(images=pil_image,
return_tensors="pt").pixel_values
            generated_ids = model.generate(pixel_values)
            text = processor.batch_decode(generated_ids,
skip_special_tokens=True)[0]
```

```
            return text.strip()
        except Exception as e:
            print(f"[TrOCR Error] {str(e)}")
            return "Handwriting OCR failed."
    else:
        # Use Tesseract for printed text in try
        try:
            processed = preprocess_image(image, mode, show_steps)
            config = r'--oem 3 --psm 6'
            text = pytesseract.image_to_string(processed, config=config)
            return text.strip() if text else "No text extracted."
        except Exception as e:
            print(f"[Tesseract Error] {str(e)}")
            return "Printed OCR failed."
```

nlp_clean.py:

```
import nltk
import re
import spacy
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download required NLTK data with error handling
def download_nltk_resources():
    try:
        nltk.data.find('tokenizers/punkt')
    except LookupError:
        nltk.download('punkt', quiet=True)
```

```
    try:
        stopwords.words('english')
    except LookupError:
        nltk.download('stopwords', quiet=True)
    try:
        nltk.data.find('corpora/wordnet')
    except LookupError:
        nltk.download('wordnet', quiet=True)
    try:
        nltk.data.find('taggers/averaged_perceptron_tagger')
    except LookupError:
        nltk.download('averaged_perceptron_tagger', quiet=True)

# Call this at module level to ensure resources are available
download_nltk_resources()

# Load spaCy model
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    from spacy.cli import download
    download("en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")

# Stopwords
stop_words = set(stopwords.words("english"))
custom_stopwords = {"im", "ive", "youre", "theyre", "dont", "wont", "cant", "us",
"pm", "am"}
stop_words.update(custom_stopwords)

def clean_text(text: str) -> str:
    if not text.strip():
        return ""

    text = text.lower()

    # Clean up special characters (expand if needed)
    text = re.sub(r'[^a-zA-Z0-9\s\'.,!?]', '', text)

    # Expand contractions
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
```

```
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)

    text = re.sub(r'\s+', ' ', text).strip()

    # Tokenize with error handling
    try:
        tokens = word_tokenize(text)
    except Exception as e:
        print(f"Tokenization error: {e}")
        tokens = text.split()  # Fallback to simple whitespace tokenization

    filtered_tokens = [word for word in tokens if word not in stop_words and
len(word) > 2]

    # Lemmatize using spaCy
    doc = nlp(" ".join(filtered_tokens))
    lemmatized_tokens = [token.lemma_ if token.pos_ in ['VERB', 'NOUN', 'ADJ',
'ADV'] else token.text for token in doc]

    return " ".join(lemmatized_tokens)
```

emotion_model.py:

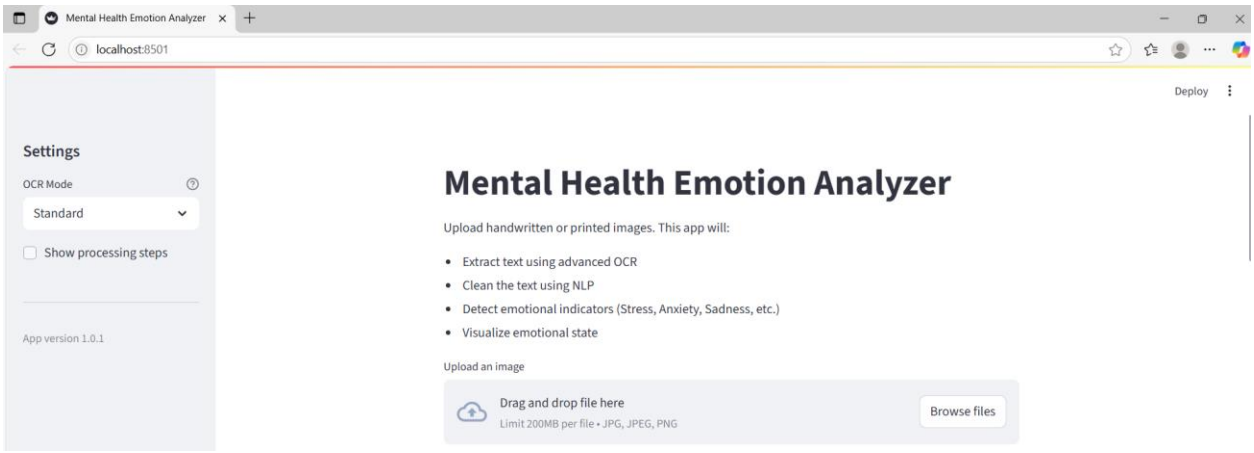
```
from transformers import pipeline
import torch

# Initialize the emotion classifier
classifier = pipeline(
    "text-classification",
    model="j-hartmann/emotion-english-distilroberta-base",
    top_k=None,
    device=0 if torch.cuda.is_available() else -1
)

# Map emotions to mental health categories
MENTAL_HEALTH_MAPPING = {
    'sadness': 'depression',
    'fear': 'anxiety',
    'anger': 'stress',
```

```
    'joy': 'well-being',
    'disgust': 'negative_outlook',
    'surprise': 'uncertainty',
    'neutral': 'neutral'
}

def detect_emotions(text):
    if not text.strip():
        return [{"label": "No text", "score": 1.0}]

    try:
        results = classifier(text)
        emotions = results[0]

        # Enhance results with mental health context
        for emotion in emotions:
            emotion['mental_health_category'] = MENTAL_HEALTH_MAPPING.get(
                emotion['label'].lower(),
                'other'
            )

        return emotions
    except Exception as e:
        # Fallback to simple keyword matching if model fails
        return simple_emotion_detection(text)

def simple_emotion_detection(text):
    """Fallback method when transformer model fails"""
    text_lower = text.lower()
    emotions = []

    # Check for depression indicators
    sad_words = ['sad', 'depress', 'hopeless', 'empty', 'lonely']
    sadness_score = sum(text_lower.count(word) for word in sad_words) / 10

    # Check for anxiety indicators
    anxiety_words = ['anxious', 'worry', 'fear', 'scared', 'nervous']
    anxiety_score = sum(text_lower.count(word) for word in anxiety_words) / 10

    # Check for stress indicators
    stress_words = ['stress', 'angry', 'frustrat', 'overwhelm', 'pressure']
    stress_score = sum(text_lower.count(word) for word in stress_words) / 10

    emotions.extend([
```

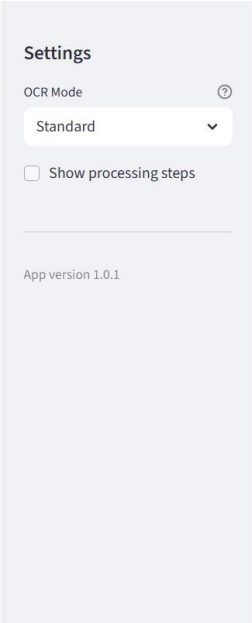
```
        {"label": "sadness", "score": min(sadness_score, 0.99),
"mental_health_category": "depression"},
        {"label": "fear", "score": min(anxiety_score, 0.99),
"mental_health_category": "anxiety"},
        {"label": "anger", "score": min(stress_score, 0.99),
"mental_health_category": "stress"},
        {"label": "neutral", "score": max(0, 1 - (sadness_score + anxiety_score +
stress_score)), "mental_health_category": "neutral"}
    ])

    # Sort by score descending
    emotions.sort(key=lambda x: x['score'], reverse=True)
    return emotions[:4]
```

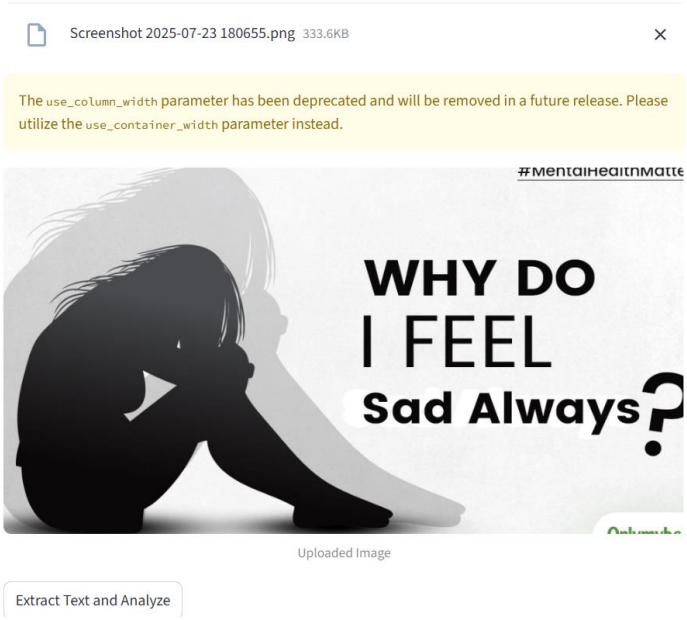
output:

Upload an image

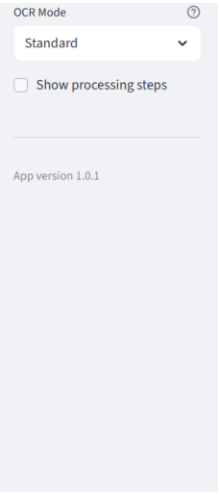


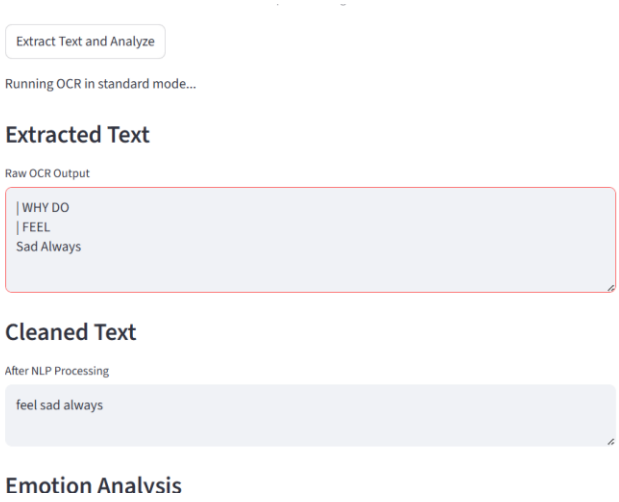
Images is loaded and select standard mode for printed images





When i clickedon button:





Dashboard:

Settings

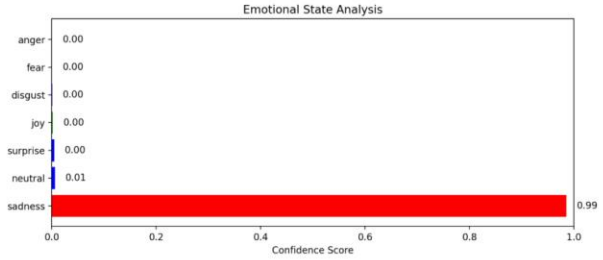
OCR Mode

Standard

Show processing steps

App version 1.0.1

Emotion Analysis



Mental Health Indicators

Sadness	Anxiety	Stress
98.5%	0.1%	0.1%
↑ High concern	↑ Low	↑ Low

Analysis completed in 6.85 seconds