

CANTILEVER AIML PROTERNSHIP 2025

PROJECT REPORT ON “VOICE- BASED AI MENTAL HEALTH ASSISTANT FOR EMOTIONAL SUPPORT ”

Under the esteemed guidance of

Adgaonkar Shashank

by

KOTA SAI VASAVI (23R11A05X0)

P LAKSHMI PRIYA (23R11A6284)

N SATHYASRI (23R11A67G6)

Geethanjali College of Engineering and Technology, Cheeryal

Table of Contents

CONTENTS	PAGE NO.
1. Abstract and Introduction	3
2. Problem Statement	4
3. Proposed Solution	4
4. Objectives	4
5. Technologies Used	4
6. Implementation Overview	4
7. Modules	5
8. Testing and Results	5
9. Existing System	5
10. Proposed System	5
11. Coding	6
12. Output Screens	94
13. Conclusion	99
14. Future Enhancements	99
15. References	99

Project Title:

Voice-Based AI Mental Health Assistant for Emotional Support

Abstract:

Mental health concerns such as stress, anxiety, and depression are increasingly prevalent, yet access to timely and effective support remains limited due to stigma, cost, and shortage of professionals. This paper presents a Voice-Based AI Mental Health Assistant designed to provide emotional support through natural, empathetic conversations. The assistant utilizes advanced technologies including Automatic Speech Recognition (ASR), Natural Language Processing (NLP), and emotion recognition algorithms to interpret users' vocal and linguistic cues. It offers features such as mood tracking, guided mindfulness exercises, and personalized motivational feedback. The system aims to serve as an accessible, non-judgmental, and always-available companion for emotional well-being. It also includes safety mechanisms to recommend professional help in high-risk scenarios, while ensuring user privacy. Though not a substitute for therapy, this AI assistant has the potential to supplement mental health care, especially in resource-limited settings.

1. Introduction

Mental health issues are a global concern with rising cases of stress, anxiety, and depression. Traditional solutions are often inaccessible due to stigma and cost. This project aims to bridge that gap by providing an AI-powered voice assistant for emotional support.

2. Problem Statement

There is a significant lack of accessible, non-judgmental, and immediate emotional support for individuals suffering from mental health challenges, especially in resource-constrained settings.

3. Proposed Solution

A Voice-Based AI Assistant that offers 24/7 emotional support using speech recognition, NLP, and emotion detection. It provides mood tracking, mindfulness exercises, and motivational feedback while maintaining user privacy and suggesting help in high-risk cases.

4. Objectives

- To design a voice-based AI assistant capable of understanding and responding empathetically.
- To integrate mood tracking and mindfulness exercises.
- To ensure user privacy and include safety protocols for high-risk scenarios.

5. Technologies Used

- Backend: Python
- Frontend: HTML (for web version), SpeechRecognition, pyttsx3
- NLP & ASR: NLTK, TextBlob, SpeechRecognition API
- Emotion Recognition: VADER Sentiment Analysis, custom logic

6. Implementation Overview

The system leverages speech recognition to convert voice input into text, analyzes it using NLP and sentiment analysis, and provides relevant responses using text-to-speech. It includes guided exercises, sentiment tracking, and escalation features when distress is detected.

7. Modules

- Input Module: Captures voice input using microphone.
- Processing Module: Uses ASR and NLP to analyze user input.
- Output Module: Responds with motivational or supportive messages via text-to-speech.
- Safety Module: Detects risk and recommends professional help.
- Mood Tracker: Records and evaluates mood patterns over time.

8. Testing and Results

Testing was conducted on real-time inputs to ensure accuracy in emotion detection and reliability in generating supportive responses. The assistant consistently provided appropriate and empathetic feedback.

9. Existing System

- Most current apps (e.g., Wysa, Woebot) offer text-based chatbots.
- Lack real-time voice interaction and deep emotional understanding.
- Often limited in providing personalized, emotion-aware responses.
- Users may feel disconnected due to robotic or scripted replies.

10. Proposed System

- A Voice-Based AI Assistant that interacts naturally via speech.
- Integrates ASR, NLP, and emotion detection for empathetic conversations.
- Offers mood tracking, guided mindfulness, and motivational feedback.
- Provides non-judgmental, 24/7 emotional support with privacy and safety features.

11. Coding

```
import streamlit as st

import pandas as pd

import nltk

from textblob import TextBlob

import json

import datetime

import re

import random

import time

from dataclasses import dataclass

from typing import List, Dict, Optional

import logging

from collections import defaultdict

from difflib import get_close_matches

import plotly.express as px

import plotly.graph_objects as go

from plotly.subplots import make_subplots

import speech_recognition as sr

import pydub
```

```
from pydub import AudioSegment

import io

import tempfile

import os

import pyttsx3

from gtts import gTTS

import pygame

import base64


# Download required NLTK data

@st.cache_resource

def download_nltk_data():

    try:

        nltk.download('vader_lexicon', quiet=True)

        nltk.download('punkt', quiet=True)

        from nltk.sentiment import SentimentIntensityAnalyzer

        return SentimentIntensityAnalyzer()

    except:

        st.warning("NLTK data download may be required. Some features  
might be limited.")
```

```
return None
```

```
@dataclass
```

```
class ConversationEntry:
```

```
    timestamp: str
```

```
    user_input: str
```

```
    emotion_score: float
```

```
    emotion_label: str
```

```
    assistant_response: str
```

```
@dataclass
```

```
class MoodEntry:
```

```
    date: str
```

```
    mood_score: int # 1-10 scale
```

```
    mood_label: str
```

```
    energy_level: int # 1-10 scale
```

```
    sleep_quality: int # 1-10 scale
```

```
    stress_level: int # 1-10 scale
```

```
    notes: str
```

```
    activities: List[str]
```



```
@dataclass
```

```
class TestResult:
```

```
    test_name: str
```

```
    score: int
```

```
    max_score: int
```

```
    result_category: str
```

```
    recommendations: List[str]
```

```
    timestamp: str
```

```
class PsychometricTests:
```

```
    def __init__(self):
```

```
        # PHQ-9 Depression Screening Test
```

```
        self.phq9_questions = [
```

```
            "Little interest or pleasure in doing things",
```

```
            "Feeling down, depressed, or hopeless",
```

```
            "Trouble falling or staying asleep, or sleeping too much",
```

```
            "Feeling tired or having little energy",
```

```
            "Poor appetite or overeating",
```

"Feeling bad about yourself or that you are a failure or have let yourself or your family down",

"Trouble concentrating on things, such as reading the newspaper or watching television",

"Moving or speaking so slowly that other people could have noticed, or the opposite - being so fidgety or restless that you have been moving around a lot more than usual",

"Thoughts that you would be better off dead, or of hurting yourself"

]

GAD-7 Anxiety Screening Test

self.gad7_questions = [

"Feeling nervous, anxious, or on edge",

"Not being able to stop or control worrying",

"Worrying too much about different things",

"Trouble relaxing",

"Being so restless that it is hard to sit still",

"Becoming easily annoyed or irritable",

"Feeling afraid, as if something awful might happen"

]

```

# Stress Assessment Questions

self.stress_questions = [

    "I feel overwhelmed by my responsibilities",

    "I have difficulty sleeping due to worry",

    "I feel irritable or short-tempered",

    "I have physical symptoms like headaches or muscle
tension",

    "I find it hard to concentrate on tasks",

    "I feel like I can't cope with daily demands",

    "I avoid social situations or activities I used to enjoy",

    "I feel constantly rushed or pressed for time",

    "I have changes in appetite (eating more or less than
usual)",

    "I feel emotionally drained most of the time"

]

```

```

# Self-Esteem Assessment

self.self_esteem_questions = [

    "I feel that I'm a person of worth, at least on an equal
plane with others",

```

```

    "I feel that I have a number of good qualities",
    "All in all, I am inclined to feel that I am a failure",
    "I am able to do things as well as most other people",
    "I feel I do not have much to be proud of",
    "I take a positive attitude toward myself",
    "On the whole, I am satisfied with myself",
    "I wish I could have more respect for myself",
    "I certainly feel useless at times",
    "At times I think I am no good at all"
]

```

```

def calculate_phq9_score(self, responses: List[int]) -> tuple:
    """Calculate PHQ-9 depression score and category"""
    total_score = sum(responses)

    if total_score <= 4:
        category = "Minimal Depression"
        recommendations = [
            "Your responses suggest minimal depressive symptoms.",
            "Continue maintaining healthy lifestyle habits.",

```

```

        "Stay connected with supportive friends and family.",

        "Consider regular exercise and stress management
techniques."

    ]

elif total_score <= 9:

    category = "Mild Depression"

    recommendations = [

        "You may be experiencing mild depressive symptoms.",

        "Consider talking to a counselor or therapist.",

        "Focus on self-care activities and stress reduction.",

        "Monitor your symptoms and seek help if they worsen.",

        "Regular exercise and social connection can be
helpful."

    ]

elif total_score <= 14:

    category = "Moderate Depression"

    recommendations = [

        "Your responses indicate moderate depressive
symptoms.",

        "It's recommended to speak with a mental health
professional.",

```

```

        "Consider therapy and/or psychiatric consultation.",

        "Develop a consistent daily routine and sleep
schedule.",

        "Reach out to trusted friends, family, or support
groups."

    ]

elif total_score <= 19:

    category = "Moderately Severe Depression"

    recommendations = [

        "You're experiencing significant depressive symptoms.",

        "Professional mental health treatment is strongly
recommended.",

        "Consider both therapy and medication evaluation.",

        "Inform trusted friends/family about your struggles.",

        "Create a safety plan and emergency contacts list."

    ]

else:

    category = "Severe Depression"

    recommendations = [

        "Your responses indicate severe depressive symptoms.",

        "Immediate professional help is strongly recommended.",

```

```

        "Contact a mental health professional or crisis line.",
        "Consider intensive treatment options.",
        "Ensure you have a strong support system in place.",
        "If having thoughts of self-harm, seek emergency help
immediately."
    ]

```

```

    return total_score, category, recommendations

```

```

def calculate_gad7_score(self, responses: List[int]) -> tuple:

```

```

    """Calculate GAD-7 anxiety score and category"""

```

```

    total_score = sum(responses)

```

```

    if total_score <= 4:

```

```

        category = "Minimal Anxiety"

```

```

        recommendations = [

```

```

            "Your responses suggest minimal anxiety symptoms.",

```

```

            "Continue practicing stress management techniques.",

```

```

            "Maintain regular exercise and healthy sleep habits.",

```

```

            "Stay mindful of stress triggers in your environment."

```

```

    ]

elif total_score <= 9:

    category = "Mild Anxiety"

    recommendations = [

        "You may be experiencing mild anxiety symptoms.",

        "Practice relaxation techniques like deep breathing.",

        "Consider mindfulness or meditation practices.",

        "Monitor your caffeine intake and stress levels.",

        "Talk to someone you trust about your concerns."

    ]

elif total_score <= 14:

    category = "Moderate Anxiety"

    recommendations = [

        "Your responses indicate moderate anxiety symptoms.",

        "Consider speaking with a mental health professional.",

        "Learn and practice anxiety management techniques.",

        "Regular exercise can significantly help with
anxiety.",

        "Consider therapy such as CBT (Cognitive Behavioral
Therapy)."

```



```

    ]

else:

    category = "Severe Anxiety"

    recommendations = [

        "You're experiencing significant anxiety symptoms.",

        "Professional mental health treatment is recommended.",

        "Consider both therapy and medication evaluation.",

        "Practice grounding techniques during anxiety
episodes.",

        "Develop a comprehensive anxiety management plan."

    ]

return total_score, category, recommendations


def calculate_stress_score(self, responses: List[int]) -> tuple:

    """Calculate stress assessment score and category"""

    total_score = sum(responses)

    max_score = len(responses) * 4 # Assuming 0-4 scale

    percentage = (total_score / max_score) * 100

```

```

if percentage <= 25:

    category = "Low Stress"

    recommendations = [

        "You're managing stress well overall.",

        "Continue your current coping strategies.",

        "Maintain work-life balance and self-care routines.",

        "Stay aware of potential stress triggers."

    ]

elif percentage <= 50:

    category = "Moderate Stress"

    recommendations = [

        "You're experiencing moderate stress levels.",

        "Focus on stress management techniques.",

        "Consider time management and prioritization
strategies.",

        "Make sure to take regular breaks and practice
self-care.",

        "Talk to someone about your stressors."

    ]

elif percentage <= 75:

```

```
category = "High Stress"

recommendations = [

    "You're experiencing high levels of stress.",

    "Consider professional support or counseling.",

    "Implement stress reduction techniques immediately.",

    "Evaluate your current responsibilities and
commitments.",

    "Prioritize sleep, exercise, and relaxation
activities."

]

else:

    category = "Very High Stress"

    recommendations = [

        "You're experiencing very high stress levels.",

        "Immediate stress management intervention is
recommended.",

        "Consider professional help from a counselor or
therapist.",

        "Evaluate major life changes or support system needs.",

        "Implement multiple stress reduction strategies
simultaneously."
```

```
]
```

```
return total_score, category, recommendations
```

```
def calculate_self_esteem_score(self, responses: List[int]) ->
tuple:

    """Calculate self-esteem score and category"""

    # Reverse scoring for negative items (items 3, 5, 8, 9, 10)

    reverse_items = [2, 4, 7, 8, 9] # 0-indexed

    adjusted_responses = []

    for i, response in enumerate(responses):

        if i in reverse_items:

            adjusted_responses.append(4 - response) # Reverse 0-4
scale

        else:

            adjusted_responses.append(response)

    total_score = sum(adjusted_responses)
```

```

if total_score >= 30:

    category = "High Self-Esteem"

    recommendations = [

        "You have healthy self-esteem levels.",

        "Continue practicing self-compassion and positive
self-talk.",

        "Use your confidence to help and support others.",

        "Maintain realistic goals and celebrate your
achievements."

    ]

elif total_score >= 20:

    category = "Moderate Self-Esteem"

    recommendations = [

        "Your self-esteem is in a moderate range.",

        "Work on identifying and challenging negative
self-thoughts.",

        "Practice self-compassion and acknowledge your
strengths.",

        "Set achievable goals to build confidence.",

        "Consider therapy if self-esteem issues persist."

    ]

```

```

else:

    category = "Low Self-Esteem"

    recommendations = [

        "You may be struggling with low self-esteem.",

        "Consider working with a therapist or counselor.",

        "Practice daily self-affirmations and gratitude.",

        "Challenge negative self-talk patterns.",

        "Focus on your strengths and past accomplishments.",

        "Surround yourself with supportive, positive people."

    ]

    return total_score, category, recommendations

```

```

class MoodTracker:

    def __init__(self):

        if 'mood_entries' not in st.session_state:

            st.session_state.mood_entries = []

        self.mood_labels = {

```

```

        1: "Terrible", 2: "Very Bad", 3: "Bad", 4: "Poor", 5:
"Okay",

        6: "Good", 7: "Very Good", 8: "Great", 9: "Excellent", 10:
"Amazing"

    }

```

```

self.activity_options = [

    "Exercise", "Meditation", "Reading", "Socializing", "Work",

    "Cooking", "Music", "Nature/Outdoors", "Gaming",
"Shopping",

    "Cleaning", "Learning", "Creative Activities", "TV/Movies",

    "Rest/Sleep", "Family Time", "Hobbies", "Travel"

]

```

```

def add_mood_entry(self, mood_score: int, energy_level: int,
sleep_quality: int,

                    stress_level: int, notes: str, activities:
List[str]):

```

```

    """Add a new mood entry"""

```

```

    entry = MoodEntry(

```

```

        date=datetime.date.today().isoformat(),

```

```

        mood_score=mood_score,

        mood_label=self.mood_labels[mood_score],

        energy_level=energy_level,

        sleep_quality=sleep_quality,

        stress_level=stress_level,

        notes=notes,

        activities=activities

    )

    st.session_state.mood_entries.append(entry)

```

```

def get_mood_trends(self, days: int = 30) -> Dict:

    """Get mood trends for visualization"""

    if not st.session_state.mood_entries:

        return None

    # Filter entries by date range

    cutoff_date = datetime.date.today() -
datetime.timedelta(days=days)

    recent_entries = [

        entry for entry in st.session_state.mood_entries

```



```

        if datetime.date.fromisoformat(entry.date) >= cutoff_date
    ]

    if not recent_entries:

        return None

    # Create DataFrame for analysis
    df = pd.DataFrame([

        {

            'date': entry.date,

            'mood_score': entry.mood_score,

            'energy_level': entry.energy_level,

            'sleep_quality': entry.sleep_quality,

            'stress_level': entry.stress_level,

            'activities': ', '.join(entry.activities)

        }

        for entry in recent_entries

    ])

    return df

```

```

def create_mood_visualization(self, df: pd.DataFrame):

    """Create mood trend visualization"""

    if df is None or df.empty:

        return None

    # Create subplot with secondary y-axis

    fig = make_subplots(

        rows=2, cols=1,

        subplot_titles=('Mood Trends Over Time', 'Wellness
Factors'),

        vertical_spacing=0.1

    )

    # Mood trend line

    fig.add_trace(

        go.Scatter(

            x=df['date'],

            y=df['mood_score'],

            mode='lines+markers',

```

```

        name='Mood Score',

        line=dict(color='#2E86AB', width=3),

        marker=dict(size=8)

    ),

    row=1, col=1

)

```

```

# Wellness factors

```

```

fig.add_trace(

    go.Scatter(

        x=df['date'],

        y=df['energy_level'],

        mode='lines+markers',

        name='Energy Level',

        line=dict(color='#A23B72', width=2)

    ),

    row=2, col=1

)

```

```

fig.add_trace(

```

```

go.Scatter(

    x=df['date'],

    y=df['sleep_quality'],

    mode='lines+markers',

    name='Sleep Quality',

    line=dict(color='#F18F01', width=2)

),

row=2, col=1

)

```

```

fig.add_trace(

    go.Scatter(

        x=df['date'],

        y=10 - df['stress_level'], # Invert stress for better
visualization

        mode='lines+markers',

        name='Low Stress (Inverted)',

        line=dict(color='#C73E1D', width=2)

    ),

    row=2, col=1

```

```

    )

    fig.update_layout(

        height=600,

        showlegend=True,

        title_text="Mood and Wellness Tracking"

    )

    fig.update_yaxes(range=[0, 10], title_text="Score", row=1,
col=1)

    fig.update_yaxes(range=[0, 10], title_text="Score", row=2,
col=1)

    fig.update_xaxes(title_text="Date", row=2, col=1)

    return fig

class EmotionAnalyzer:

    def __init__(self):

        self.sentiment_analyzer = download_nltk_data()

```

```

# Load dataset if available, otherwise use fallback

try:

    dataset_path = "mental_health_dataset.csv"

    self.df = pd.read_csv(dataset_path)

    # Normalize

    self.df['Extracted Concern'] = self.df['Extracted
Concern'].str.lower().str.strip()

    self.df['Category'] =
self.df['Category'].str.lower().str.strip()

    # Build mappings

    self.concern_to_category = dict(zip(self.df['Extracted
Concern'], self.df['Category']))

    self.concern_to_intensity = defaultdict(list)

    for _, row in self.df.iterrows():

        self.concern_to_intensity[row['Extracted
Concern']].append(row['Intensity'])

except:

    st.info("Mental health dataset not found. Using built-in
emotion detection.")

```

```

        self.concern_to_category = {}

        self.concern_to_intensity = defaultdict(list)

        # Known emotion categories (fallback)

        self.emotion_keywords = {

            'anxiety': ['anxious', 'worried', 'nervous', 'panic',
                        'fear', 'stress', 'overwhelmed'],

            'depression': ['sad', 'depressed', 'hopeless', 'empty',
                           'worthless', 'lonely', 'down'],

            'anger': ['angry', 'frustrated', 'mad', 'irritated',
                      'furious', 'annoyed'],

            'joy': ['happy', 'excited', 'joyful', 'cheerful', 'glad',
                    'pleased', 'content'],

            'fear': ['scared', 'afraid', 'terrified', 'frightened'],

            'neutral': []

        }

```

```

def analyze_emotion(self, text: str) -> tuple:

```

```

    text_lower = text.lower()

```

```

    # Try to match known concerns

```

```

    if self.concern_to_category:

        possible_matches = get_close_matches(text_lower,
self.concern_to_category.keys(), n=1, cutoff=0.6)

        if possible_matches:

            matched = possible_matches[0]

            category = self.concern_to_category[matched]

            avg_intensity = sum(self.concern_to_intensity[matched])
/ len(self.concern_to_intensity[matched])

            return category, avg_intensity / 10 # Normalize
intensity to 0 - 1

```

```

# Fallback keyword-based emotion classification

emotion_scores = {}

for emotion, keywords in self.emotion_keywords.items():

    score = sum(1 for keyword in keywords if keyword in
text_lower)

    emotion_scores[emotion] = score

primary_emotion = max(emotion_scores, key=emotion_scores.get)

if emotion_scores[primary_emotion] == 0:

    primary_emotion = 'neutral'

```



```

# Use TextBlob or VADER for backup polarity intensity

blob = TextBlob(text)

polarity = blob.sentiment.polarity

if self.sentiment_analyzer:

    scores = self.sentiment_analyzer.polarity_scores(text)

    intensity = max(abs(scores['pos']), abs(scores['neg']),
abs(scores['neu']))

else:

    intensity = abs(polarity)

return primary_emotion, intensity

```

```

class ResponseGenerator:

    def __init__(self):

        # Keywords to detect specific situations in user input

        self.situation_keywords = {

            'work': ['work', 'job', 'boss', 'colleague', 'office',
'career', 'deadline', 'meeting'],

            'relationship': ['boyfriend', 'girlfriend', 'partner',
'husband', 'wife', 'relationship', 'dating', 'breakup', 'marriage'],

```

```

        'family': ['family', 'parents', 'mom', 'dad', 'mother',
'father', 'sibling', 'brother', 'sister', 'children', 'kids'],

        'school': ['school', 'college', 'university', 'exam',
'test', 'study', 'homework', 'grades', 'teacher', 'professor'],

        'health': ['sick', 'illness', 'doctor', 'hospital', 'pain',
'health', 'medical', 'symptoms'],

        'financial': ['money', 'debt', 'bills', 'financial', 'job
loss', 'unemployed', 'broke', 'expensive'],

        'loss': ['died', 'death', 'funeral', 'lost', 'grief',
'goodbye', 'miss them', 'passed away'],

        'change': ['moving', 'new job', 'change', 'transition',
'different', 'unfamiliar', 'starting'],

        'social': ['friends', 'social', 'party', 'people', 'crowd',
'talking', 'conversation', 'shy'],

        'body': ['weight', 'appearance', 'body', 'looks', 'eating',
'food', 'diet', 'mirror', 'ugly']

    }

```

```

# Coping strategies

```

```

self.coping_strategies = {

```

```

    'breathing': "Here's a technique that can help right now:
breathe in slowly for 4 counts, hold for 7, then exhale for 8. This
activates your body's relaxation response.",

```

'grounding': "Let's ground you in the present moment. Can you name 5 things you can see, 4 things you can touch, 3 things you can hear, 2 things you can smell, and 1 thing you can taste?",

'journaling': "Sometimes writing down our thoughts can help untangle them. Even jotting down a few sentences about how you're feeling can provide clarity and relief.",

'connection': "Human connection can be incredibly healing. Is there someone you trust - a friend, family member, or counselor - who you could reach out to?",

'movement': "Physical movement can help process emotions. Even a short walk, some stretching, or dancing to a favorite song can shift your energy.",

'self-compassion': "Try speaking to yourself the way you'd speak to a dear friend going through the same thing. You deserve the same kindness you'd show others.",

'mindfulness': "When emotions feel overwhelming, try focusing on your breath or the sensations in your body. This can help anchor you in the present moment."

}

```
def detect_situation(self, user_input: str) -> str:
```

```
    """Detect the specific situation the user is talking about"""
```

```
    user_input_lower = user_input.lower()
```

```

        for situation, keywords in self.situation_keywords.items():

            if any(keyword in user_input_lower for keyword in
keywords):

                return situation

        return 'general'

def generate_contextual_response(self, emotion: str, intensity:
float, user_input: str, situation: str) -> str:

    """Generate a contextual response based on emotion, situation,
and user input"""

    user_input_lower = user_input.lower()

    # Start with acknowledgment of their specific situation

    if situation == 'work':

        if 'stress' in user_input_lower or 'overwhelm' in
user_input_lower:

            base_response = "Work stress can feel all-consuming,
especially when it seems like there's no escape from the pressure. It
sounds like your job is really weighing on you right now."

            elif 'boss' in user_input_lower:

```

```
        base_response = "Dealing with difficult workplace  
dynamics, especially with supervisors, can be incredibly draining and  
affect your whole sense of well-being."
```

```
    else:
```

```
        base_response = "Work challenges can spill over into  
every aspect of our lives. It sounds like you're carrying a heavy load  
right now."
```

```
    elif situation == 'relationship':
```

```
        if 'breakup' in user_input_lower or 'broke up' in  
user_input_lower:
```

```
            base_response = "The end of a relationship can feel  
like losing a piece of yourself. The pain you're feeling is real and  
valid - heartbreak is one of the most intense emotional experiences we  
can have."
```

```
            elif 'fight' in user_input_lower or 'argument' in  
user_input_lower:
```

```
                base_response = "Relationship conflicts can leave us  
feeling disconnected and hurt. It's painful when someone we care about  
feels distant or upset with us."
```

```
            else:
```

```
                base_response = "Relationships can bring us our  
greatest joys and deepest challenges. It sounds like you're navigating  
some difficult waters with someone important to you."
```

```
elif situation == 'family':
```

```
    base_response = "Family dynamics can be some of the most  
complex and emotionally charged relationships we have. It sounds like  
something at home is really affecting you."
```

```
elif situation == 'school':
```

```
    if 'exam' in user_input_lower or 'test' in  
user_input_lower:
```

```
        base_response = "Academic pressure can feel  
overwhelming, especially when it feels like your entire future depends  
on performance. The stress you're feeling is completely  
understandable."
```

```
    else:
```

```
        base_response = "School can bring up so many emotions -  
pressure, comparison, uncertainty about the future. It sounds like  
you're dealing with some significant academic stress."
```

```
elif situation == 'health':
```

```
    base_response = "Health concerns can be terrifying because  
they make us confront our vulnerability. It's natural to feel scared  
when your body isn't feeling right or when facing medical uncertainty."
```

```
elif situation == 'financial':
```

```
    base_response = "Financial stress affects every aspect of  
life and can make you feel trapped and anxious about the future. Money  
worries can be incredibly isolating and overwhelming."
```

```
elif situation == 'loss':
```

```
    base_response = "Loss and grief are some of the most  
profound experiences we face as humans. There's no timeline for  
healing, and whatever you're feeling right now is exactly what you need  
to feel."
```

```
elif situation == 'change':
```

```
    base_response = "Major life changes, even positive ones,  
can trigger anxiety and uncertainty. It's completely normal to feel  
unsettled when your familiar world is shifting."
```

```
elif situation == 'social':
```

```
    base_response = "Social situations can feel incredibly  
overwhelming, especially when anxiety makes every interaction feel like  
it's under a microscope. Your feelings about social settings are  
valid."
```

```
elif situation == 'body':
```

```
base_response = "Our relationship with our bodies and  
appearance can be so complicated and painful. The thoughts you're  
having about yourself are causing real distress."
```

```
else:
```

```
# General emotional acknowledgment
```

```
if emotion == 'anxiety':
```

```
base_response = "I can hear the worry and tension in  
what you're sharing. Anxiety can make everything feel urgent and  
overwhelming."
```

```
elif emotion == 'depression':
```

```
base_response = "It sounds like you're carrying some  
heavy feelings right now. Depression can make even simple things feel  
impossible."
```

```
elif emotion == 'anger':
```

```
base_response = "I can sense the frustration and anger  
in your words. Those feelings are telling you that something important  
to you has been threatened or hurt."
```

```
elif emotion == 'fear':
```

```
base_response = "Fear can be paralyzing, making us feel  
small and vulnerable. What you're experiencing sounds really  
frightening."
```

```
elif emotion == 'joy':
```



```
        base_response = "I love hearing the happiness in your words! It sounds like something wonderful is happening in your life."
```

```
    else:
```

```
        base_response = "Thank you for sharing what's on your heart. I can tell this is important to you."
```

```
    # Add emotion-specific support
```

```
    if emotion in ['anxiety', 'fear'] and intensity > 0.5:
```

```
        emotional_support = " Right now, your nervous system is on high alert, which is exhausting. You're not broken - you're human, and you're responding to something that feels threatening."
```

```
        elif emotion == 'depression' and intensity > 0.5:
```

```
            emotional_support = " Depression can make you feel like you're drowning in your own thoughts. Please know that what you're experiencing is real, but it's not permanent."
```

```
            elif emotion == 'anger' and intensity > 0.5:
```

```
                emotional_support = " Anger often protects other vulnerable feelings underneath. It makes sense that you'd feel this way given what you're going through."
```

```
        else:
```

```
            emotional_support = " Your feelings make complete sense given your situation."
```

```
# Add personalized questions to continue the conversation

if situation == 'work':

    questions = " Have you been able to talk to anyone about
the pressure you're under? What would make tomorrow at work feel even
slightly more manageable?"

elif situation == 'relationship':

    questions = " How are you taking care of yourself through
this relationship challenge? What do you need most right now - space,
support, or clarity?"

elif situation == 'family':

    questions = " Family situations can feel especially stuck
because we can't just walk away. What boundaries might help protect
your emotional well-being?"

elif situation == 'loss':

    questions = " Grief comes in waves and there's no right way
to do it. How are you honoring both your pain and your love for what
you've lost?"

else:

    questions = " What would feel most supportive for you right
now? Sometimes just being heard is enough, and sometimes we need more
active help."
```

```

response = base_response + emotional_support + questions

# Add coping strategy if intensity is high

if intensity > 0.6:

    if emotion in ['anxiety', 'fear']:

        strategy = self.coping_strategies['breathing']

    elif emotion == 'anger':

        strategy = self.coping_strategies['movement']

    elif emotion == 'depression':

        strategy = self.coping_strategies['self-compassion']

    else:

        strategy =
random.choice(list(self.coping_strategies.values()))

response += f"\n\nHere's something that might help right
now: {strategy}"

return response

def generate_response(self, emotion: str, intensity: float,
user_input: str) -> str:

```

```
        """Generate an appropriate response based on emotion category
and user input"""
```

```
        situation = self.detect_situation(user_input)
```

```
        return self.generate_contextual_response(emotion, intensity,
user_input, situation)
```

```
class VoiceInputHandler:
```

```
    def __init__(self):
```

```
        self.recognizer = sr.Recognizer()
```

```
        self.microphone = sr.Microphone()
```

```
        # Adjust for ambient noise on initialization
```

```
        with self.microphone as source:
```

```
            self.recognizer.adjust_for_ambient_noise(source,
duration=1)
```

```
    def listen_for_speech(self, timeout=10, phrase_time_limit=30):
```

```
        """
```

```
        Listen for speech input from microphone
```

```
        Args:
```

```
            timeout: Maximum time to wait for speech to start
```

phrase_time_limit: Maximum time for a single phrase

Returns:

tuple: (success: bool, text: str, error_message: str)

"""

try:

with self.microphone as source:

st.info("🎤 Listening... Please speak now!")

audio = self.recognizer.listen(

source,

timeout=timeout,

phrase_time_limit=phrase_time_limit

)

st.info("🔄 Processing speech...")

text = self.recognizer.recognize_google(audio)

return True, text, ""

except sr.WaitTimeoutError:

return False, "", "Listening timeout - no speech detected"

```

except sr.UnknownValueError:

    return False, "", "Could not understand the audio"

except sr.RequestError as e:

    return False, "", f"Could not request results; {e}"

except Exception as e:

    return False, "", f"An error occurred: {e}"


def process_audio_file(self, audio_file):
    """
    Process uploaded audio file

    Args:

        audio_file: Streamlit uploaded file object

    Returns:

        tuple: (success: bool, text: str, error_message: str)
    """
    try:

        # Save uploaded file temporarily

```

```

        with tempfile.NamedTemporaryFile(delete=False,
suffix='.wav') as tmp_file:

            tmp_file.write(audio_file.read())

            tmp_path = tmp_file.name

# Convert to WAV if necessary

try:

    audio = AudioSegment.from_file(tmp_path)

    wav_path = tmp_path.replace('.wav', '_converted.wav')

    audio.export(wav_path, format="wav")

# Recognize speech

with sr.AudioFile(wav_path) as source:

    audio_data = self.recognizer.record(source)

    text = self.recognizer.recognize_google(audio_data)

# Cleanup

os.unlink(tmp_path)

os.unlink(wav_path)

```

```

        return True, text, ""

    except Exception as e:

        # Cleanup on error

        if os.path.exists(tmp_path):

            os.unlink(tmp_path)

        return False, "", f"Error processing audio file: {e}"

    except Exception as e:

        return False, "", f"Error handling audio file: {e}"

class VoiceOutputHandler:

    def __init__(self):

        """Initialize voice output with both offline and online TTS
options"""

        self.offline_engine = None

        self.online_available = True

        # Initialize offline TTS (pyttsx3)

        try:

            self.offline_engine = pyttsx3.init()

```



```

        self.setup_offline_voice()

except Exception as e:

    st.warning(f"Offline TTS initialization failed: {e}")

    self.offline_engine = None


# Initialize pygame for audio playback

try:

    pygame.mixer.init()

except Exception as e:

    st.warning(f"Audio mixer initialization failed: {e}")


def setup_offline_voice(self):

    """Configure offline voice settings"""

    if self.offline_engine:

        # Get available voices

        voices = self.offline_engine.getProperty('voices')


        # Set voice properties

        self.offline_engine.setProperty('rate', 150) # Speech rate

```

```

        self.offline_engine.setProperty('volume', 0.8) # Volume
(0.0 to 1.0)

        # Try to set a female voice if available

        for voice in voices:

            if 'female' in voice.name.lower() or 'zira' in
voice.name.lower():

                self.offline_engine.setProperty('voice', voice.id)

                break

def speak_offline(self, text: str) -> bool:

    """Use offline TTS (pyttsx3) to speak text"""

    try:

        if self.offline_engine:

            self.offline_engine.say(text)

            self.offline_engine.runAndWait()

            return True

        return False

    except Exception as e:

        st.error(f"Offline TTS error: {e}")

```

```

        return False

def speak_online(self, text: str, language='en') -> bool:
    """Use online TTS (gTTS) to speak text"""
    try:
        # Create gTTS object
        tts = gTTS(text=text, lang=language, slow=False)

        # Save to temporary file
        with tempfile.NamedTemporaryFile(delete=False,
suffix='.mp3') as tmp_file:

            tts.save(tmp_file.name)

        # Play audio using pygame
        pygame.mixer.music.load(tmp_file.name)

        pygame.mixer.music.play()

        # Wait for playback to complete
        while pygame.mixer.music.get_busy():
            time.sleep(0.1)

```

```

        # Clean up

        os.unlink(tmp_file.name)

        return True

except Exception as e:

    st.error(f"Online TTS error: {e}")

    return False

def speak_text(self, text: str, method='auto') -> bool:

    """

    Speak text using specified method

    Args:

        text: Text to speak

        method: 'offline', 'online', or 'auto'

    """

    # Clean text for better speech

    clean_text = self.clean_text_for_speech(text)

```

```

        if method == 'offline' or (method == 'auto' and
self.offline_engine):

            return self.speak_offline(clean_text)

        elif method == 'online' or method == 'auto':

            return self.speak_online(clean_text)

        else:

            st.error("No TTS method available")

            return False


def clean_text_for_speech(self, text):

    """Clean text for better text-to-speech output"""

    # Remove markdown formatting

    text = re.sub(r'\*(.*?)\*', r'\1', text) # Bold - **text**

    text = re.sub(r'\*(.*?)\*', r'\1', text) # Italic - *text*

    text = re.sub(r'\`(.*)\`', r'\1', text) # Code - `text`

    text = re.sub(r'\[(.*?)\]\((.*?)\)', r'\1', text) # Links -
[text](url)

    # Remove special characters that don't read well

    text = re.sub(r'[\# \-]', '', text)

```

```

# Clean up extra whitespace

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


return text


def create_audio_player_component(self, text: str) -> str:

    """Create HTML audio player component for web playback"""

    try:

        # Generate audio using gTTS

        tts = gTTS(text=self.clean_text_for_speech(text),
lang='en', slow=False)

        # Save to bytes

        audio_buffer = io.BytesIO()

        tts.write_to_fp(audio_buffer)

        audio_buffer.seek(0)

        # Encode to base64 for embedding

```

```

        audio_base64 =
base64.b64encode(audio_buffer.read()).decode()

        # Create HTML audio player

        audio_html = f"""

        <audio controls autoplay style="width: 100%; margin: 10px
0;">

                <source src="data:audio/mp3;base64,{audio_base64}"
type="audio/mp3">

                Your browser does not support the audio element.

        </audio>

        """

        return audio_html

    except Exception as e:

        st.error(f"Audio player creation failed: {e}")

        return ""

class MentalHealthAssistant:

    def __init__(self):

```

```

self.emotion_analyzer = EmotionAnalyzer()

self.response_generator = ResponseGenerator()

self.mood_tracker = MoodTracker()

self.psychometric_tests = PsychometricTests()

self.voice_output_handler = VoiceOutputHandler()


# Initialize session state

if 'conversation_history' not in st.session_state:

    st.session_state.conversation_history = []

if 'session_start' not in st.session_state:

    st.session_state.session_start = datetime.datetime.now()

if 'test_results' not in st.session_state:

    st.session_state.test_results = []

if 'voice_enabled' not in st.session_state:

    st.session_state.voice_enabled = False

if 'voice_method' not in st.session_state:

    st.session_state.voice_method = 'auto'


def emergency_check(self, text: str) -> bool:

```



```
"""Check for emergency keywords that require immediate
attention"""
```

```
emergency_keywords = [

    'suicide', 'kill myself', 'end my life', 'want to die',

    'hurt myself', 'self harm', 'overdose'

]
```

```
text_lower = text.lower()

for keyword in emergency_keywords:

    if keyword in text_lower:

        return True

return False
```

```
def handle_emergency(self):
```

```
    """Provide emergency resources and support"""
```

```
    emergency_message = """
```

```
        🚨 *I'm very concerned about what you've shared. Your life has
value and you deserve support.*
```

```
        *Please reach out to professional help immediately:*
```

- *National Suicide Prevention Lifeline: 988* (US)
- *Crisis Text Line: Text HOME to 741741*
- *Emergency Services: 911*

You don't have to face this alone. Professional counselors are available 24/7.

```
"""
```

```
    return emergency_message
```

```
# Continuation from where the code was cut off
```

```
def save_conversation(self):
```

```
    """Save conversation history to downloadable format"""
```

```
    if st.session_state.conversation_history:
```

```
        conversations_dict = [
```

```
            {
```

```
                'timestamp': entry.timestamp,
```

```
                'user_input': entry.user_input,
```

```
                'emotion_score': entry.emotion_score,
```

```
                'emotion_label': entry.emotion_label,
```

```

        'assistant_response': entry.assistant_response
    }

    for entry in st.session_state.conversation_history
]

df = pd.DataFrame(conversations_dict)

return df.to_csv(index=False)

return None

def process_user_input(self, user_input: str) -> str:

    """Process user input and generate appropriate response"""

    # Check for emergency situations first

    if self.emergency_check(user_input):

        return self.handle_emergency()

    # Analyze emotion

    emotion, intensity =
self.emotion_analyzer.analyze_emotion(user_input)

    # Generate response

```

```
        response = self.response_generator.generate_response(emotion,
intensity, user_input)
```

```
# Save to conversation history
```

```
conversation_entry = ConversationEntry(

    timestamp=datetime.datetime.now().isoformat(),

    user_input=user_input,

    emotion_score=intensity,

    emotion_label=emotion,

    assistant_response=response

)
```

```
st.session_state.conversation_history.append(conversation_entry)
```

```
return response
```

```
def run_streamlit_app(self):
```

```
    """Main Streamlit application interface"""
```

```
    st.set_page_config(
```

```
        page_title="AI Mental Health Assistant",
```

```

    page_icon="🧠",

    layout="wide",

    initial_sidebar_state="expanded"

)

# Initialize session state variables if they don't exist

if 'conversation_history' not in st.session_state:

    st.session_state.conversation_history = []

if 'session_start' not in st.session_state:

    st.session_state.session_start = datetime.datetime.now()

if 'test_results' not in st.session_state:

    st.session_state.test_results = []

if 'mood_entries' not in st.session_state:

    st.session_state.mood_entries = []


# Custom CSS for better styling

st.markdown("""

<style>

.main-header {

    font-size: 2.5rem;

```

```

        color: #2E86AB;

        text-align: center;

        margin-bottom: 2rem;
    }

    .emotion-badge {

        display: inline-block;

        padding: 0.25rem 0.75rem;

        margin: 0.25rem;

        border-radius: 20px;

        font-size: 0.8rem;

        font-weight: bold;
    }

    .anxiety { background-color: #FFE5E5; color: #D63384; }

    .depression { background-color: #E5E5FF; color: #6610F2; }

    .anger { background-color: #FFE5CC; color: #FD7E14; }

    .joy { background-color: #E5F5E5; color: #198754; }

    .fear { background-color: #FFF3CD; color: #FFCA2C; }

    .neutral { background-color: #F8F9FA; color: #6C757D; }

    .voice-input-section {

        background-color: #f0f8ff;
    }

```

```

padding: 1rem;

border-radius: 10px;

border-left: 4px solid #2E86AB;

margin: 1rem 0;
}

.voice-status {

font-weight: bold;


color: #2E86AB;
}

```

```
</style>
```

```
""", unsafe_allow_html=True)
```

```
# Main header
```

```
st.markdown('<h1 class="main-header"> AI Mental Health
Assistant</h1>', unsafe_allow_html=True)
```

```
# Sidebar navigation
```

```
st.sidebar.title("Dashboard")
```

```
page = st.sidebar.selectbox(
```

```
    "Choose a section:",
```

```

        ["Voice Assistant", "Mood Tracker", "Psychometric Tests",
"Analytics & Insights", "Resources"]

    )

```

```

if page == "Voice Assistant":

    self.chat_interface()

elif page == "Mood Tracker":

    self.mood_tracker_interface()

elif page == "Psychometric Tests":

    self.psychometric_tests_interface()

elif page == "Analytics & Insights":

    self.analytics_interface()

elif page == "Resources":

    self.resources_interface()

```

```

def chat_interface(self):

    """Chat interface for emotional support with voice input and
output"""

    st.header("🗣️ Emotional Support Chat")

```



```

# ADD VOICE OUTPUT SETTINGS

with st.expander("🔊 Voice Settings"):

    col1, col2 = st.columns(2)

    with col1:

        st.session_state.voice_enabled = st.checkbox(

            "Enable Voice Output",

            value=st.session_state.voice_enabled,

            help="Enable text-to-speech for assistant
responses"

        )

    with col2:

        st.session_state.voice_method = st.selectbox(

            "Voice Method",

            ['auto', 'offline', 'online'],

            index=['auto', 'offline',
'online'].index(st.session_state.voice_method),

            help="Auto: Try offline first, then online |
Offline: Computer voice | Online: Natural voice (requires internet)"

        )

# Initialize voice handler if not exists

```

```

if not hasattr(self, 'voice_handler'):

    self.voice_handler = VoiceInputHandler()

# ... (keep existing voice input code)

# Display conversation history (MODIFY THIS SECTION around line
900)

if st.session_state.conversation_history:

    st.subheader("Conversation History")

    for entry in st.session_state.conversation_history[-5:]:

        with st.container():

            col1, col2 = st.columns([3, 1])

            with col1:

                st.write(f"*You:* {entry.user_input}")

                st.write(f"*Assistant:*
{entry.assistant_response}")

# ADD VOICE OUTPUT BUTTON

if st.session_state.voice_enabled:

    col_a, col_b = st.columns([1, 4])

```

```

        with col_a:

            if st.button(f" Play",
key=f"play_{entry.timestamp}"):

                success =

self.voice_output_handler.speak_text(

                    entry.assistant_response,

                    st.session_state.voice_method

                )

                if success:

                    st.success("Playing
response...")

                else:

                    st.error("Voice output failed")

# Alternative: Web audio player (works
better in some environments)

        with col_b:

            if st.button(f" Web Player",
key=f"web_{entry.timestamp}"):

                audio_html =

self.voice_output_handler.create_audio_player_component(

                    entry.assistant_response

```

```

        )

        if audio_html:

            st.markdown(audio_html,
unsafe_allow_html=True)

        with col2:

            emotion_class = entry.emotion_label.lower()

            st.markdown(f'<span class="emotion-badge
{emotion_class}">{entry.emotion_label.title()}</span>',
                        unsafe_allow_html=True)

            st.write(f"Intensity:
{entry.emotion_score:.2f}")

            st.divider()

# Text input section (MODIFY the button action around line 950)

st.subheader("💬 Text Input")

user_input = st.text_area(

    "Share what's on your mind...",

    placeholder="I'm here to listen and support you. Feel free
to share whatever you're experiencing.",

    height=100

```

```

)

col1, col2, col3 = st.columns([1, 1, 2])

with col1:

    if st.button("Send Message", type="primary"):

        if user_input.strip():

            with st.spinner("Processing your message..."):

                response = self.process_user_input(user_input)

            st.success("Response generated!")

        # ADD AUTOMATIC VOICE OUTPUT

        if st.session_state.voice_enabled:

            with st.spinner("Generating voice
response..."):

                success =
self.voice_output_handler.speak_text(

                    response,

                    st.session_state.voice_method

```

```

        )

        if success:

            st.info("🔊 Playing voice response...")

        else:

            st.warning("Voice output failed, but
you can use the play buttons above")

            st.rerun()

        else:

            st.warning("Please enter a message before
sending.")

```

```

# ... (keep rest of the method unchanged)

```

```

def mood_tracker_interface(self):

    """Mood tracking interface"""

    st.header("📊 Daily Mood Tracker")

    col1, col2 = st.columns([1, 1])

```

```

with col1:

    st.subheader("Log Today's Mood")

    mood_score = st.slider("Overall Mood", 1, 10, 5,
                            help="1 = Terrible, 10 = Amazing")

    energy_level = st.slider("Energy Level", 1, 10, 5)
    sleep_quality = st.slider("Sleep Quality", 1, 10, 5)
    stress_level = st.slider("Stress Level", 1, 10, 5,
                             help="1 = No stress, 10 = Extremely
stressed")

    activities = st.multiselect(
        "Activities Today",
        self.mood_tracker.activity_options,
        help="Select all activities you did today"
    )

    notes = st.text_area(

```

```

        "Additional Notes",

        placeholder="Any thoughts, events, or observations
about your day..."

    )

    if st.button("Save Mood Entry", type="primary"):

        self.mood_tracker.add_mood_entry(

            mood_score, energy_level, sleep_quality,

            stress_level, notes, activities

        )

        st.success("Mood entry saved successfully!")

        st.rerun()

with col2:

    st.subheader("Mood Trends")

    if st.session_state.mood_entries:

        days_to_show = st.selectbox("Show data for:", [7, 14,
30, 60], index=2)

```



```

df = self.mood_tracker.get_mood_trends(days_to_show)

if df is not None and not df.empty:

    fig =

self.mood_tracker.create_mood_visualization(df)

    if fig:

        st.plotly_chart(fig, use_container_width=True)


# Summary statistics

st.subheader("Summary Statistics")

avg_mood = df['mood_score'].mean()

avg_energy = df['energy_level'].mean()

avg_sleep = df['sleep_quality'].mean()

avg_stress = df['stress_level'].mean()


col1, col2, col3, col4 = st.columns(4)

col1.metric("Avg Mood", f"{avg_mood:.1f}")

col2.metric("Avg Energy", f"{avg_energy:.1f}")

col3.metric("Avg Sleep", f"{avg_sleep:.1f}")

col4.metric("Avg Stress", f"{avg_stress:.1f}")

else:

```

```
        st.info("No mood data available for the selected  
period.")
```

```
    else:
```

```
        st.info("Start logging your daily mood to see trends  
and insights!")
```

```
def psychometric_tests_interface(self):
```

```
    """Interface for psychometric tests"""
```

```
    st.header("🧠 Mental Health Assessments")
```

```
        st.info("These assessments are screening tools and not  
diagnostic. Please consult a mental health professional for proper  
evaluation.")
```

```
        test_type = st.selectbox(  
            "Choose an assessment:",  
            ["PHQ-9 (Depression)", "GAD-7 (Anxiety)", "Stress  
Assessment", "Self-Esteem Scale"]  
        )
```

```
        if test_type == "PHQ-9 (Depression)":
```

```

        self.run_phq9_test()

    elif test_type == "GAD-7 (Anxiety)":

        self.run_gad7_test()

    elif test_type == "Stress Assessment":

        self.run_stress_test()

    elif test_type == "Self-Esteem Scale":

        self.run_self_esteem_test()


def run_phq9_test(self):

    """Run PHQ-9 depression screening test"""

    st.subheader("PHQ-9 Depression Screening")

    st.write("Over the last 2 weeks, how often have you been
bothered by any of the following problems?")


    responses = []

    options = ["Not at all", "Several days", "More than half the
days", "Nearly every day"]


    for i, question in
enumerate(self.psychometric_tests.phq9_questions):

        response = st.radio(

```

```

        f"{i+1}. {question}",

        options,

        key=f"phq9_{i}",

        horizontal=True

    )

    responses.append(options.index(response))


if st.button("Calculate PHQ-9 Score", type="primary"):

    score, category, recommendations =
self.psychometric_tests.calculate_phq9_score(responses)


# Save result

test_result = TestResult(

    test_name="PHQ-9",

    score=score,

    max_score=27,

    result_category=category,

    recommendations=recommendations,

    timestamp=datetime.datetime.now().isoformat()

)

```

```

st.session_state.test_results.append(test_result)


# Display results

st.success("Assessment completed!")


col1, col2 = st.columns([1, 2])

with col1:

    st.metric("PHQ-9 Score", f"{score}/27")

    st.metric("Category", category)


with col2:

    st.subheader("Recommendations")

    for rec in recommendations:

        st.write(f"• {rec}")


def run_gad7_test(self):

    """Run GAD-7 anxiety screening test"""

    st.subheader("GAD-7 Anxiety Screening")

    st.write("Over the last 2 weeks, how often have you been
bothered by the following problems?")

```

```

responses = []

options = ["Not at all", "Several days", "More than half the
days", "Nearly every day"]

for i, question in
enumerate(self.psychometric_tests.gad7_questions):

    response = st.radio(

        f"{i+1}. {question}",

        options,

        key=f"gad7_{i}",

        horizontal=True

    )

    responses.append(options.index(response))

if st.button("Calculate GAD-7 Score", type="primary"):

    score, category, recommendations =

self.psychometric_tests.calculate_gad7_score(responses)

# Save result

test_result = TestResult(

```

```

        test_name="GAD-7",

        score=score,

        max_score=21,

        result_category=category,

        recommendations=recommendations,

        timestamp=datetime.datetime.now().isoformat()
    )

    st.session_state.test_results.append(test_result)


# Display results

st.success("Assessment completed!")


col1, col2 = st.columns([1, 2])

with col1:

    st.metric("GAD-7 Score", f"{score}/21")

    st.metric("Category", category)


with col2:

    st.subheader("Recommendations")

    for rec in recommendations:

```

```

        st.write(f" • {rec}")

def run_stress_test(self):

    """Run stress assessment test"""

    st.subheader("Stress Level Assessment")

    st.write("Rate how much each statement applies to you recently:")

    responses = []

    options = ["Never", "Rarely", "Sometimes", "Often", "Always"]

    for i, question in
enumerate(self.psychometric_tests.stress_questions):

        response = st.radio(

            f"{i+1}. {question}",

            options,

            key=f"stress_{i}",

            horizontal=True

        )

        responses.append(options.index(response))

```



```

if st.button("Calculate Stress Score", type="primary"):

    score, category, recommendations =
self.psychometric_tests.calculate_stress_score(responses)


# Save result

test_result = TestResult(

    test_name="Stress Assessment",

    score=score,

    max_score=len(responses) * 4,

    result_category=category,

    recommendations=recommendations,

    timestamp=datetime.datetime.now().isoformat()

)

st.session_state.test_results.append(test_result)


# Display results

st.success("Assessment completed!")


col1, col2 = st.columns([1, 2])

```

```

        with col1:

            st.metric("Stress Score", f"{score}/{len(responses) *
4}")

            st.metric("Category", category)

        with col2:

            st.subheader("Recommendations")

            for rec in recommendations:

                st.write(f" • {rec}")

def run_self_esteem_test(self):

    """Run self-esteem assessment test"""

    st.subheader("Self-Esteem Scale")

    st.write("Rate how much you agree with each statement:")

    responses = []

    options = ["Strongly Disagree", "Disagree", "Neutral", "Agree",
"Strongly Agree"]

    for i, question in
enumerate(self.psychometric_tests.self_esteem_questions):

```

```

response = st.radio(

    f"{i+1}. {question}",

    options,

    key=f"self_esteem_{i}",

    horizontal=True

)

responses.append(options.index(response))


if st.button("Calculate Self-Esteem Score", type="primary"):

    score, category, recommendations =
self.psychometric_tests.calculate_self_esteem_score(responses)


# Save result

test_result = TestResult(

    test_name="Self-Esteem Scale",

    score=score,

    max_score=40,

    result_category=category,

    recommendations=recommendations,

    timestamp=datetime.datetime.now().isoformat()

```

```

    )

    st.session_state.test_results.append(test_result)

    # Display results

    st.success("Assessment completed!")

    col1, col2 = st.columns([1, 2])

    with col1:

        st.metric("Self-Esteem Score", f"{score}/40")

        st.metric("Category", category)

    with col2:

        st.subheader("Recommendations")

        for rec in recommendations:

            st.write(f" • {rec}")

def analytics_interface(self):

    """Analytics and insights interface"""

    st.header("📊 Analytics & Insights")

```

```
        if not st.session_state.conversation_history and not
st.session_state.mood_entries and not st.session_state.test_results:

            st.info("Start using the chat, mood tracker, or assessments
to see your analytics!")

            return
```

```
# Conversation analytics
```

```
if st.session_state.conversation_history:
```

```
    st.subheader("Conversation Insights")
```

```
# Emotion distribution
```

```
    emotions = [entry.emotion_label for entry in
st.session_state.conversation_history]
```

```
    emotion_counts = pd.Series(emotions).value_counts()
```

```
    col1, col2 = st.columns(2)
```

```
    with col1:
```

```
        fig_pie = px.pie(
```

```
            values=emotion_counts.values,
```

```
            names=emotion_counts.index,
```

```

        title="Emotion Distribution in Conversations"

    )

    st.plotly_chart(fig_pie, use_container_width=True)

with col2:

    # Emotion intensity over time

    df_conv = pd.DataFrame([

        {

            'timestamp': entry.timestamp,

            'emotion': entry.emotion_label,

            'intensity': entry.emotion_score

        }

        for entry in st.session_state.conversation_history

    ])

    df_conv['timestamp'] =

pd.to_datetime(df_conv['timestamp'])

    fig_line = px.line(

        df_conv, x='timestamp', y='intensity',

```

```

        color='emotion', title="Emotion Intensity Over
Time"

    )

    st.plotly_chart(fig_line, use_container_width=True)


# Test results analytics

if st.session_state.test_results:

    st.subheader("Assessment History")


    for result in st.session_state.test_results[-5:]: # Show
last 5 results

        with st.expander(f"{result.test_name} -
{result.timestamp[:10]}"):

            col1, col2 = st.columns([1, 2])

            with col1:

                st.metric("Score",
f"{result.score}/{result.max_score}")

                st.metric("Category", result.result_category)

            with col2:

                st.write("*Recommendations:*")

                for rec in result.recommendations:

```

```
st.write(f" • {rec}")
```

```
def resources_interface(self):
```

```
    """Mental health resources interface"""
```

```
    st.header("🆘 Mental Health Resources")
```

```
    # Emergency resources
```

```
    st.subheader("🚨 Crisis Resources")
```

```
    st.error("""
```

```
    *If you're in immediate danger or having thoughts of  
self-harm:*
```

```
    - *National Suicide Prevention Lifeline: 988* (US)
```

```
    - *Crisis Text Line: Text HOME to 741741*
```

```
    - *Emergency Services: 911*
```

```
    """)
```

```
    # Professional help
```

```
    st.subheader("🏥 Professional Help")
```

```
    st.info("""
```

```
    *Finding Professional Support:*
```


- Psychology Today:
psychologytoday.com - Find therapists
near you

- SAMHSA Treatment Locator:
findtreatment.samhsa.gov

- Your insurance provider's mental health directory
- Your primary care physician for referrals

""")

Self-help resources

st.subheader("📖 Self-Help Resources")

st.write("""

Apps:

- Headspace: Meditation and mindfulness
- Calm: Sleep stories and relaxation
- Youper: AI emotional health assistant
- Sanvello: Anxiety and mood tracking

Books:

- "Feeling Good" by David D. Burns
- "The Anxiety and Worry Workbook" by David A. Clark

- "Mind Over Mood" by Dennis Greenberger

Websites:

- Mental Health America:

mhanational.org

- National Alliance on Mental Illness:

nami.org

- Anxiety and Depression Association:

adaa.org

""")

Coping strategies

st.subheader("🔧 Coping Strategies")

strategy_tabs = st.tabs(["Anxiety", "Depression", "Stress",
"General Wellbeing"])

with strategy_tabs[0]:

st.write("""

For Anxiety:

- Practice deep breathing exercises (4-7-8 technique)

- Use grounding techniques (5-4-3-2-1 method)
 - Progressive muscle relaxation
 - Limit caffeine intake
 - Challenge anxious thoughts with evidence
- """)

with strategy_tabs[1]:

```

st.write("""
*For Depression:*

- Maintain a regular sleep schedule

- Get sunlight exposure daily

- Exercise regularly, even light walking

- Connect with supportive people

- Practice self-compassion

- Set small, achievable daily goals

""")

```

with strategy_tabs[2]:

```

st.write("""
*For Stress:*

```

- Time management and prioritization
- Regular breaks during work
- Physical exercise or movement
- Mindfulness meditation
- Journaling thoughts and feelings
- Boundary setting

""")

with strategy_tabs[3]:

st.write("""

General Wellbeing:

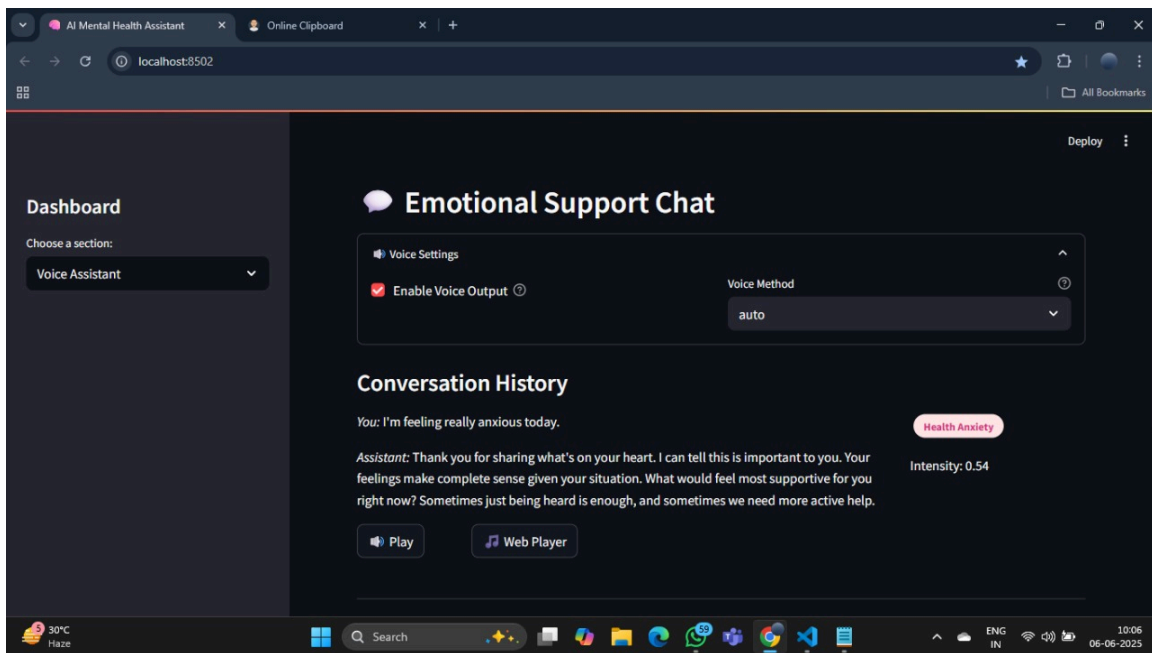
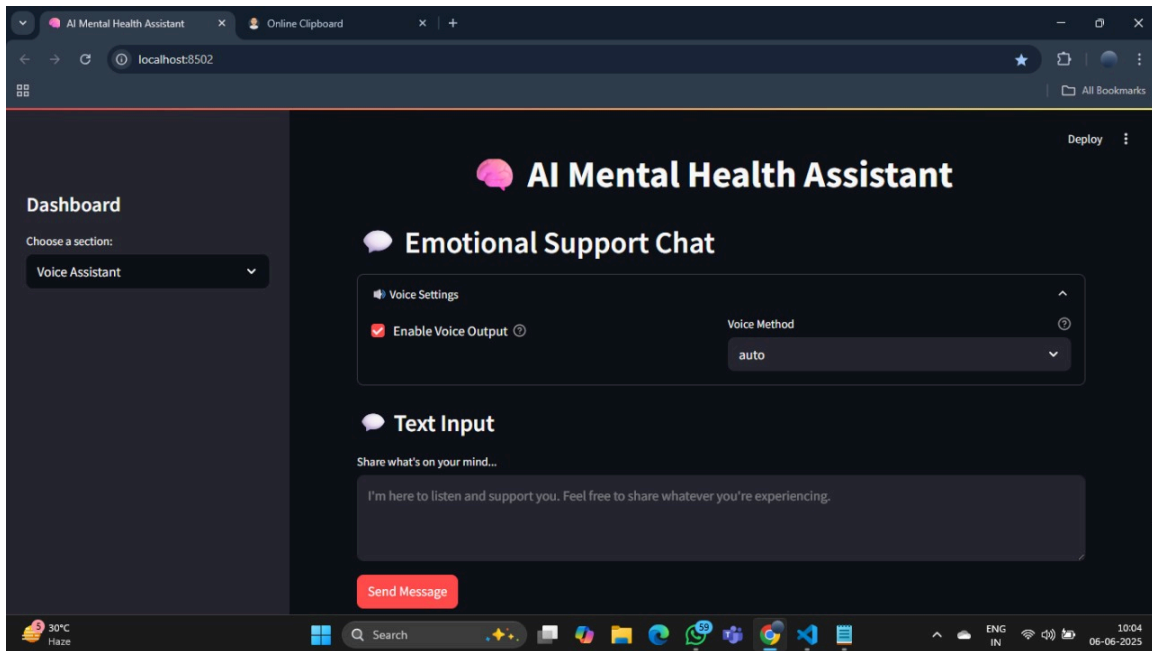
- Maintain social connections
- Practice gratitude daily
- Engage in hobbies and interests
- Limit social media consumption
- Get adequate sleep (7-9 hours)
- Eat nutritious meals regularly

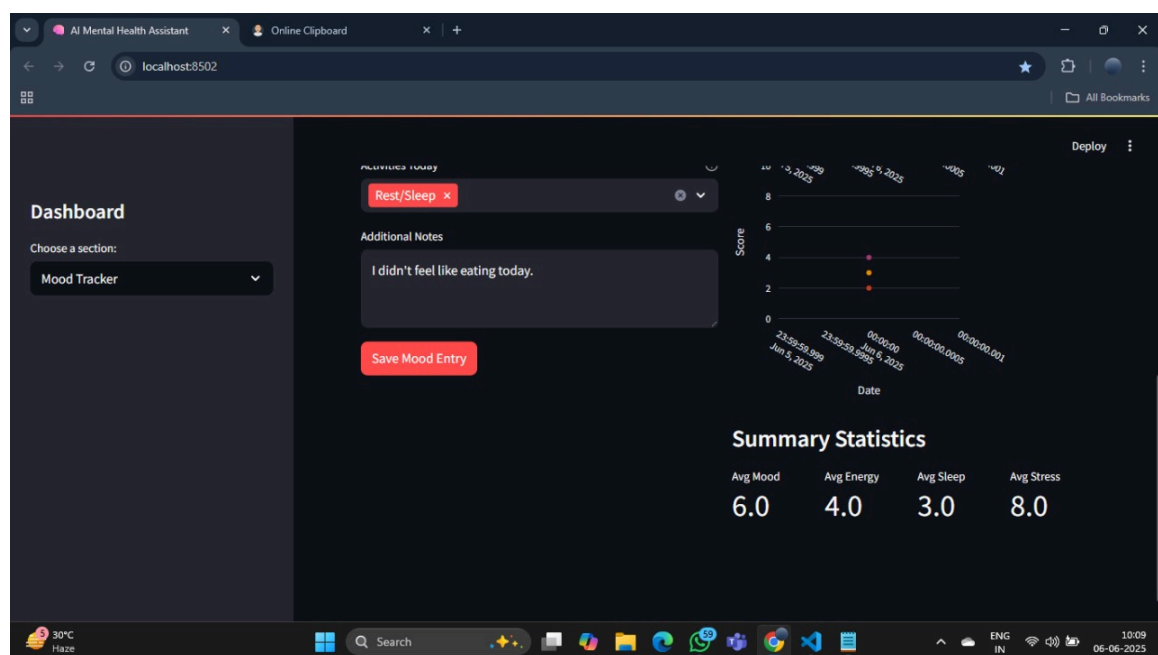
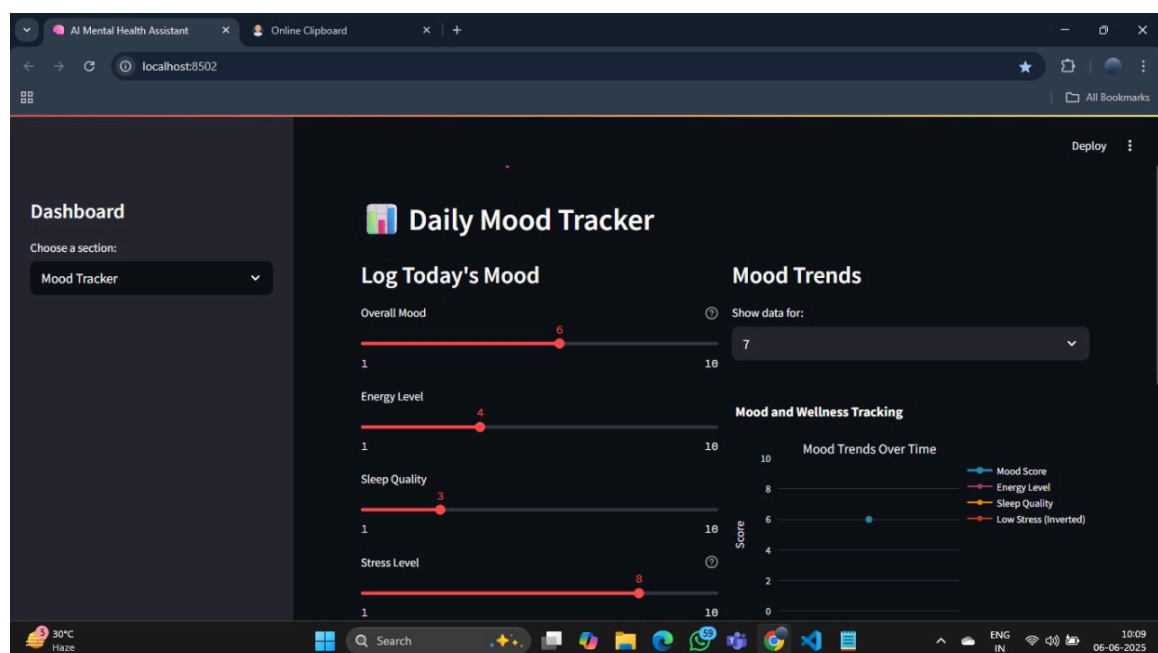
""")

Main application entry point

```
def main():  
  
    """Main function to run the Mental Health Assistant"""  
  
    assistant = MentalHealthAssistant()  
  
    assistant.run_streamlit_app()  
  
  
if __name__ == "__main__":  
  
    main()
```

12. Output Screens





AI Mental Health Assistant | Online Clipboard | localhost:8502

Mental Health Assessments

These assessments are screening tools and not diagnostic. Please consult a mental health professional for proper evaluation.

Choose an assessment:

PHQ-9 (Depression)

PHQ-9 (Depression)

GAD-7 (Anxiety)

Stress Assessment

Self-Esteem Scale

☒ Not at all ☐ Several days ☐ More than half the days ☐ Nearly every day

Feeling down, depressed, or hopeless

☒ Not at all ☐ Several days ☐ More than half the days ☐ Nearly every day

Trouble falling or staying asleep, or sleeping too much

30°C Haze | 10:10 06-06-2025

AI Mental Health Assistant | Online Clipboard | localhost:8502

Mental Health Assessments

Not at all ☒ Several days ☐ More than half the days ☐ Nearly every day

Thoughts that you would be better off dead, or of hurting yourself

☐ Not at all ☒ Several days ☐ More than half the days ☐ Nearly every day

Calculate PHQ-9 Score

Assessment completed!

PHQ-9 Score

16/27

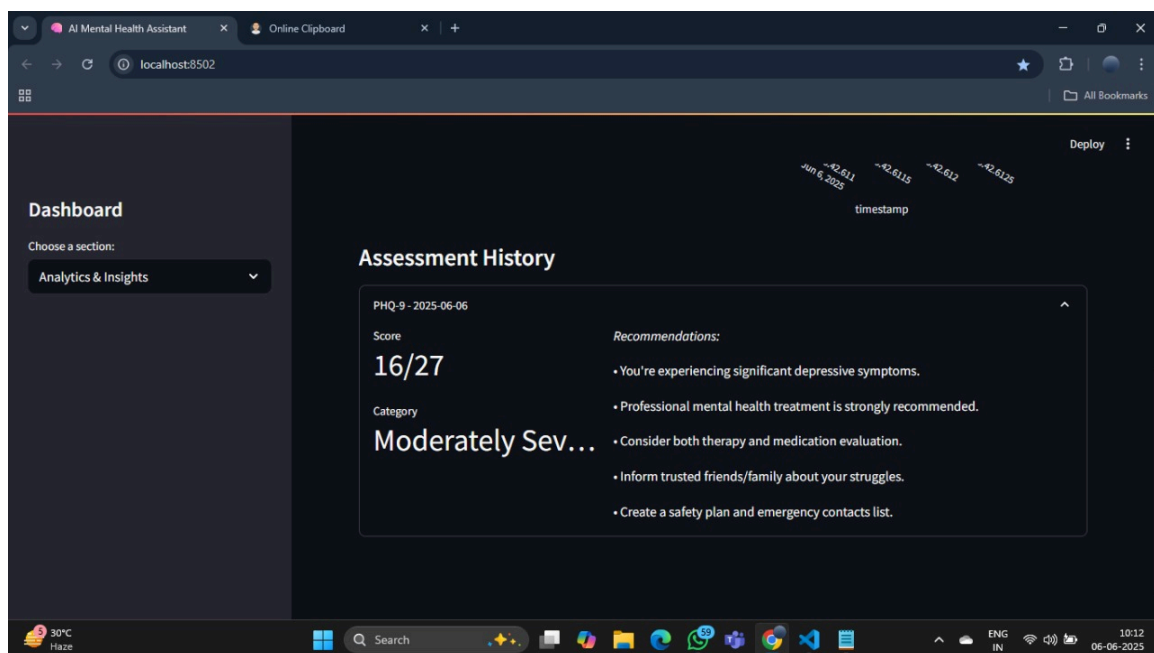
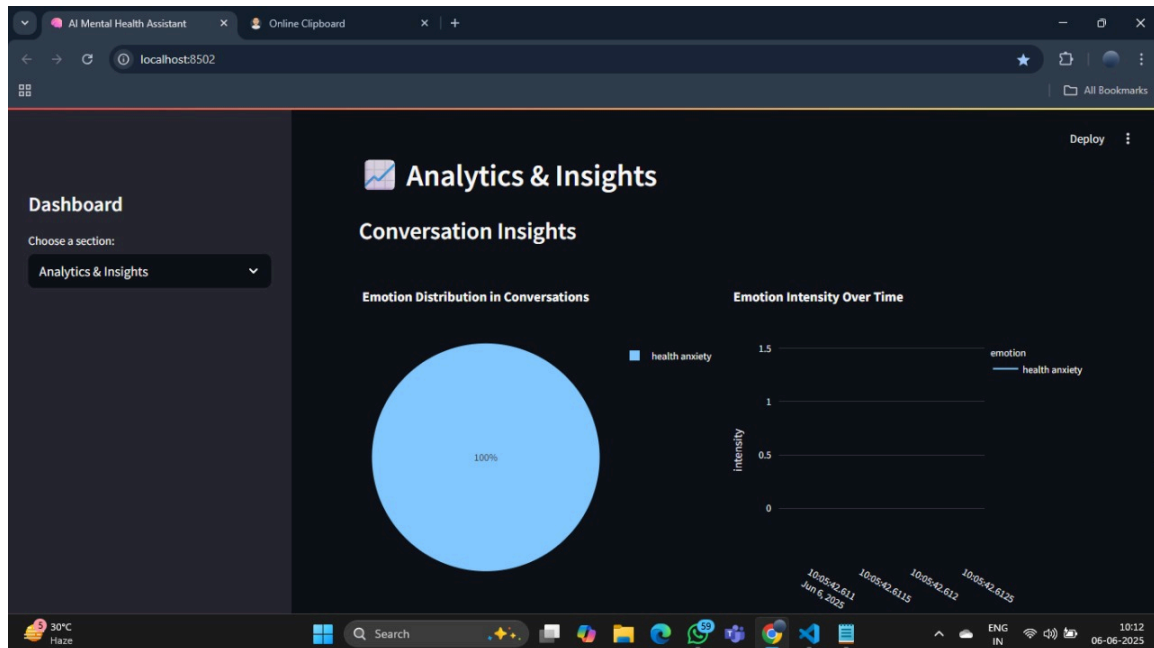
Category

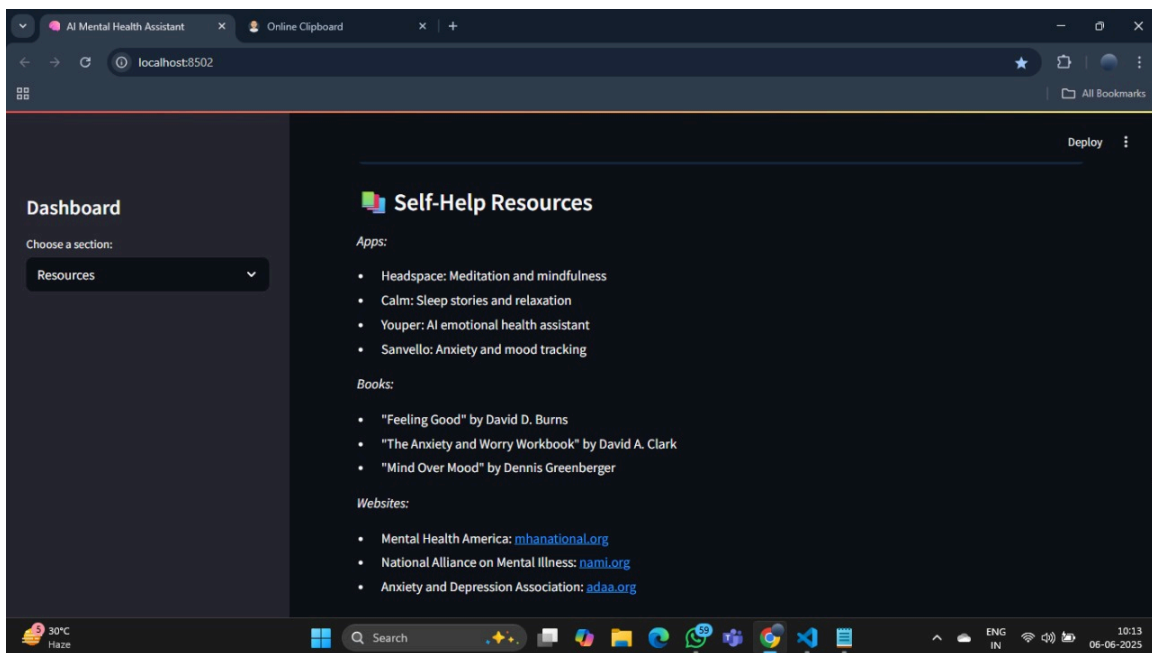
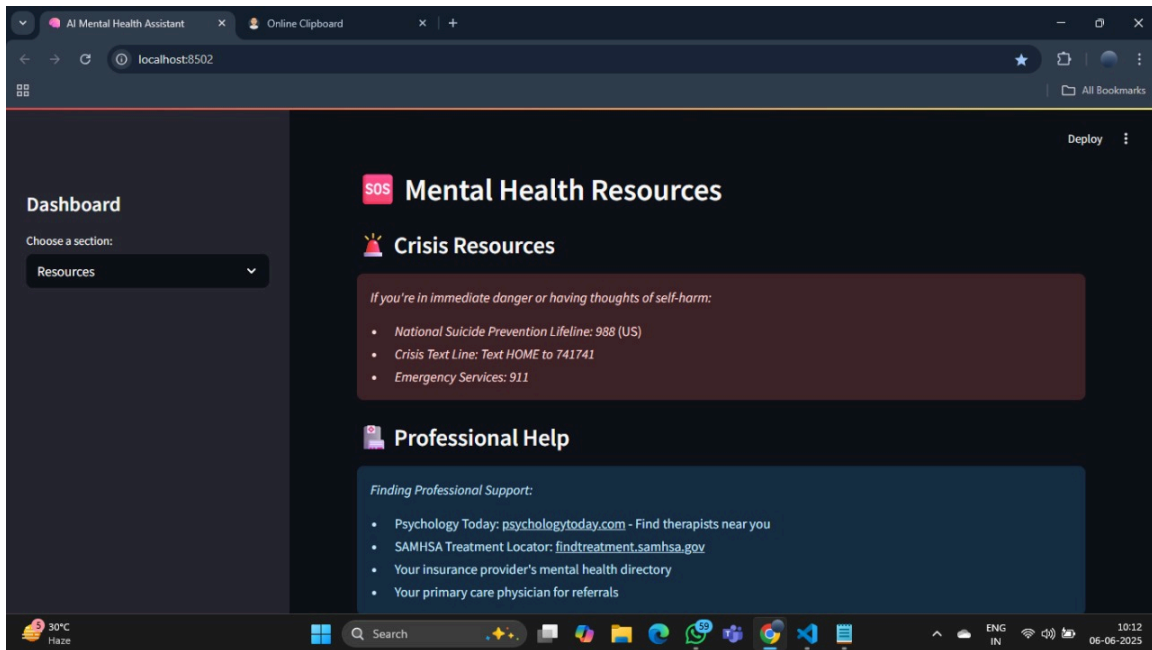
Moderately Sev...

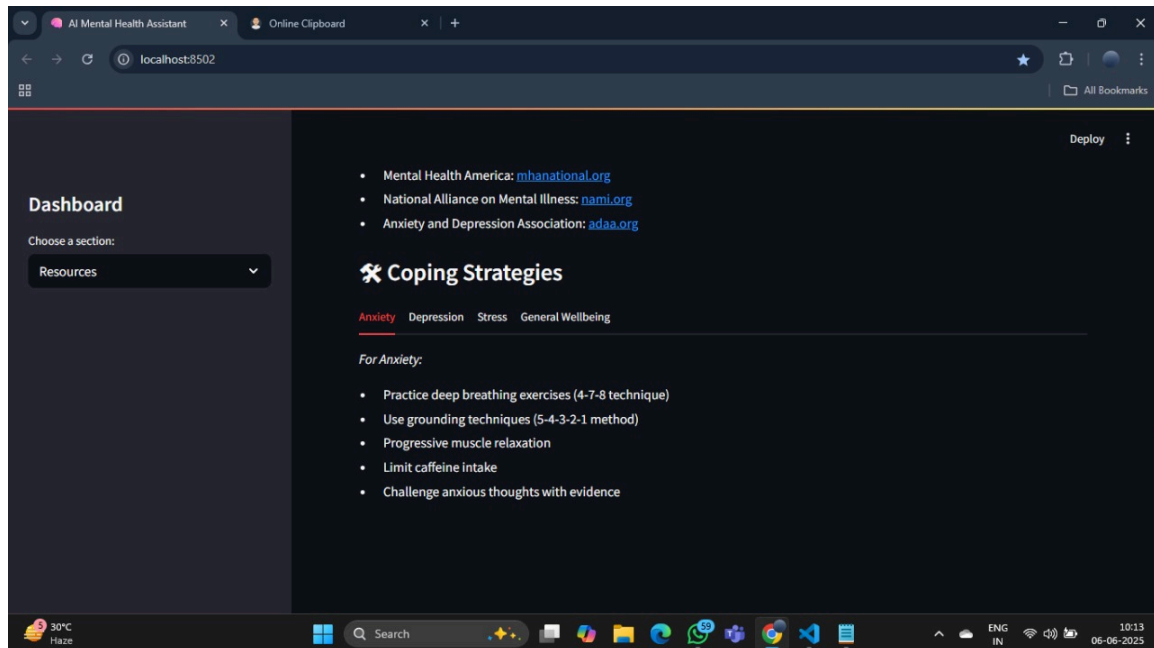
Recommendations

- You're experiencing significant depressive symptoms.
- Professional mental health treatment is strongly recommended.
- Consider both therapy and medication evaluation.
- Inform trusted friends/family about your struggles.
- Create a safety plan and emergency contacts list.

30°C Haze | 10:11 06-06-2025







13. Conclusion

The Voice-Based AI Mental Health Assistant successfully demonstrates the potential of AI in providing preliminary mental health support. While it is not a replacement for professional therapy, it serves as a helpful companion and emotional outlet.

14. Future Enhancements

- Multilingual support for regional accessibility.
- Integration with wearable devices for physiological monitoring.
- Expansion to include text-based chat option.
- Enhanced personalization using user data trends (with consent).

15. References

- WHO Mental Health Report 2022
- NLTK, TensorFlow, and TextBlob documentation
- Research papers on emotion recognition in speech