Aleeza Aslam
Intern - Tech Horizon Technologies

AI Agents Tasks

Task 1: Smart Email Sorting Agent

Objective:

The goal of this task was to build a Python program that automatically categorizes incoming emails into groups like Work, Spam, Social, and Promotions. This simulates how email providers filter and organize messages.

How the Program Works

- **Streamlit Setup**
  - When you run streamlit run file.py, it opens a web page in your browser.
  - The page shows a text box for entering/pasting an email message.
- **Email Categories**
  - For each category, a list of keywords was created.
    - Work → ["project", "meeting", "client"]
    - Spam → ["win", "free", "lottery"]
    - Social → ["party", "friend", "hangout"]
    - Promotions → ["sale", "discount", "offer"]
- **Categorizing Emails**
  - Each email from the inbox is passed through the function.
  - The result (category) is stored in a dictionary for neat display.
- **Optional Sentiment Tagging**
  - If the subject contains strong words like "urgent" or "immediately", the program tags the email as Urgent.
  - Prints the subject along with its category and urgency tag if applicable.

Code:

```python
import streamlit as st

# --------------------------
# Title
# --------------------------
st.title("📧 Simple Email Sorter")

# --------------------------
# Step 1: Input emails
# --------------------------
st.write("Enter email subjects (one per line):")
user_input = st.text_area("Your Inbox", height=200)

# --------------------------
# Step 2: Define keyword rules
# --------------------------
work_keywords = ["meeting", "project", "deadline", "salary", "job", "update", "interview", "appointment", "client"]
spam_keywords = ["win", "free", "lottery", "prize", "claim", "reward", "selected"]
social_keywords = ["friend", "invitation", "dinner", "party", "coffee", "reunion", "comment"]
promotions_keywords = ["offer", "sale", "deal", "discount", "purchase", "travel", "arrivals", "exclusive", "buy one"]
urgent_keywords = ["urgent", "asap", "immediately", "important"]

# --------------------------
# Step 3: Categorize function
# --------------------------
def categorize_email(email):
    subject = email.lower()
    if any(word in subject for word in work_keywords):
        return "Work"
    elif any(word in subject for word in spam_keywords):
        return "Spam"
    elif any(word in subject for word in social_keywords):
        return "Social"
    elif any(word in subject for word in promotions_keywords):
        return "Promotions"
    else:
        return "Other"

def check_urgent(email):
    subject = email.lower()
    return any(word in subject for word in urgent_keywords)

# --------------------------
# Step 4: Run the classifier
# --------------------------
if st.button("Sort Emails"):
    emails = user_input.split("\n")
    for mail in emails:
        if mail.strip() == "":
            continue
        category = categorize_email(mail)
        urgent_tag = " 🚨URGENT" if check_urgent(mail) else ""
        st.write(f"**{mail}** → {category}{urgent_tag}")
```

Task 2: AI Life Coach Agent

Objective:

The goal of this task was to build a simple AI-based Life Coach that responds to the user's mood with motivational suggestions or activity ideas. The program asks the user how they feel, interprets the mood, and then gives suitable advice. It also remembers recent moods to avoid repeating the same suggestions.

How the Program Works:

- **Mood–Response Dictionary**
    - The code starts with a dictionary (responses) where each mood (like *tired*, *stressed*, *happy*, etc.) has a list of 3 motivational suggestions.
    - When you type your mood, the program randomly picks one suggestion from the corresponding list.
- **Short-Term Memory**
    - To avoid repeating the same advice, the program keeps track of the last 3 moods entered in a list called recent_moods.
    - If you type the same mood again too soon, it warns you instead of repeating.
- **Life Coach Function**
    - The life_coach(mood) function checks if your mood exists in the dictionary.
    - If yes → returns a random motivational tip.
    - If not → replies with a default "I don't have advice for that mood yet" message.
- **Main Loop**
    - Runs continuously, asking: "How are you feeling right now?"
    - You can type moods like happy, sad, etc. → program gives advice.
    - If you type quit, the loop ends and says goodbye.

Code:

```python
import streamlit as st
import random

# ---------------------------
# Step 1: Define mood-based responses
# ---------------------------
responses = {
    "tired": [
        "Take a short 20-minute nap to recharge.",
        "Go for a quick walk to refresh your mind.",
        "Drink some water and stretch your body."
    ],
    "stressed": [
        "Take a deep breath. Inhale… exhale…",
        "Try writing down your thoughts to clear your head.",
        "Listen to calming music for 10 minutes."
    ],
    "happy": [
        "That's wonderful! Keep spreading positivity.",
        "Celebrate by doing something you love today.",
        "Share your happiness with a friend or family member."
    ],
    "sad": [
        "It's okay to feel sad. Talk to someone you trust.",
        "Watch a feel-good movie or listen to uplifting music.",
        "Remember: tough times don't last, but tough people do."
    ],
    "angry": [
        "Try a short workout to release the tension.",
        "Count to 10 slowly before reacting.",
        "Write down what's making you angry and then tear the paper."
    ]
}
```

```python
# Step 2: Function to get advice
# ---------------------------
def life_coach(mood):
    mood = mood.lower()
    if mood in responses:
        return random.choice(responses[mood])
    else:
        return "I don't have advice for that mood yet, but stay positive!"

# ---------------------------
# Streamlit UI
# ---------------------------
st.title("🌟 AI Life Coach")
st.write("Tell me how you feel (e.g., tired, stressed, happy, sad, angry).")

mood_input = st.text_input("How are you feeling right now?")

if st.button("Get Advice"):
    if mood_input:
        advice = life_coach(mood_input)
        st.success(f"💡 Suggestion: {advice}")
    else:
        st.warning("Please enter a mood.")
```

Task 3: AI Study Planner Agent

Objective:

The objective of this program is to help students plan their study time effectively based on the total time available and the relative difficulty of each subject. The program ensures that harder subjects get more time while easier ones get less. It also converts time into hours and minutes for clarity and ends with a motivational quote to keep the student inspired.

How the Program Works:

- **User Inputs**
    - First, it asks: “Enter total study time (in minutes)”.
    - Then it asks: “Enter subjects separated by commas” (e.g., Math, Physics, English).
    - For each subject, it asks you to rate difficulty from 1 (easy) to 5 (hard).
- **Weighting by Difficulty**
    - Each subject’s difficulty acts like a weight.
    - Example: If Physics is 5 and Math is 3, Physics gets more study time than Math.
- **Time Allocation Formula**
    - The code calculates proportion:

      Subject proportion = difficulty / total of all difficulties
    - Then it multiplies this by total study time to get minutes for each subject.
    - Converts minutes into hours + minutes for clarity.
- **Output Study Plan**
    - Prints a structured plan with each subject’s allocated time.
    - Example:
        - Math: 1 hr 15 min
        - Physics: 2 hr 45 min
- **Motivational Quote**
    - A random quote is chosen from a list using random.choice(quotes) and displayed at the end.

Code:

```python
import streamlit as st
import random

st.title("📘 Simple Study Planner 📘")

# Step 1: Ask total time and subjects
total_time = st.number_input("Enter total study time available (in minutes):", min_value=1, step=1)
subjects_input = st.text_area("Enter subjects separated by commas (e.g., Math, Physics, English):")

# Use session_state to store subjects
if "subjects" not in st.session_state:
    st.session_state.subjects = []

if st.button("Next Step ➡️"):
    if total_time > 0 and subjects_input:
        st.session_state.subjects = [s.strip() for s in subjects_input.split(",")]
    else:
        st.warning("Please enter both total time and subjects.")
```

```python
# Step 2: Show difficulty inputs only if subjects exist
if st.session_state.subjects:
    st.subheader("📊 Assign Difficulty Levels (1=easy, 5=hard):")
    difficulties = []
    for subject in st.session_state.subjects:
        level = st.slider(f"{subject}", min_value=1, max_value=5, value=3, key=subject)
        difficulties.append(level)

    if st.button("Generate Study Plan"):
        total_weight = sum(difficulties)

        st.subheader("📝 Your Study Plan:")
        for i in range(len(st.session_state.subjects)):
            proportion = difficulties[i] / total_weight
            allocated_time = int(proportion * total_time)
            hours = allocated_time // 60
            mins = allocated_time % 60

            if hours > 0:
                st.write(f"- {st.session_state.subjects[i]}: {hours} hr {mins} min")
            else:
                st.write(f"- {st.session_state.subjects[i]}: {mins} min")

        quotes = [
            "Keep going, you are doing great!",
            "Small progress is still progress.",
            "Stay focused, stay determined.",
            "One step at a time leads to success."
        ]
        st.success("💡 Motivation: " + random.choice(quotes))
```