Lesson 3 Demo 1

Build Your First AutoGen Agent Using Azure API

Objective:

To demonstrate how to build and configure AutoGen agents for automated customer support interactions using an Azure-hosted OpenAI model.

You are a customer support representative for an online store and frequently receive a high volume of repetitive queries about order tracking. To improve efficiency and reduce human workload, you deploy an AI-powered support agent using Azure OpenAI services. This agent will automatically handle common inquiries, allowing human agents to focus on complex customer issues.

Prerequisites:

- Create a virtual environment.
- Install required dependencies (autogen, dotenv).
- Obtain an Azure OpenAI deployment endpoint, API key, and model configuration.

Steps to be followed:

Step 1: Set up the environment

Step 2: Define customer and support agents

Step 3: Configure the support agent

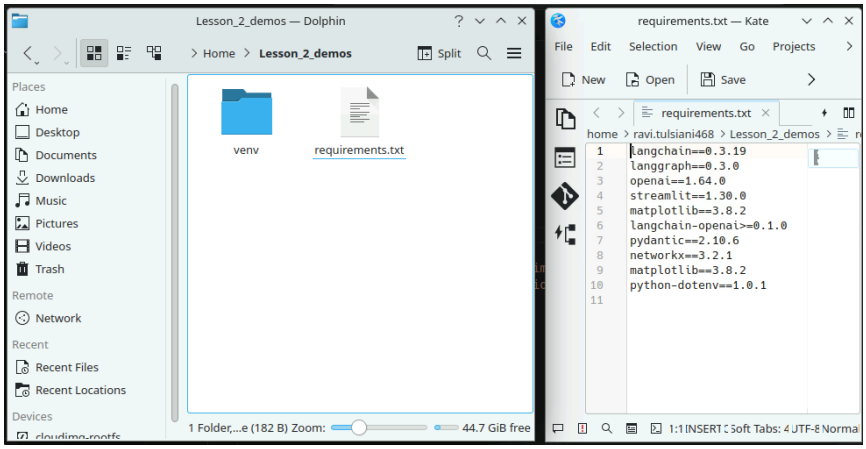
Step 4: Run the code

Step 1: Set up the environment

1.1 Open command prompt and create a new folder named "**Lesson_3_demos**" and go to the respective folder using the command given below:
mkdir Lesson_3_demos
cd Lesson_3_demos

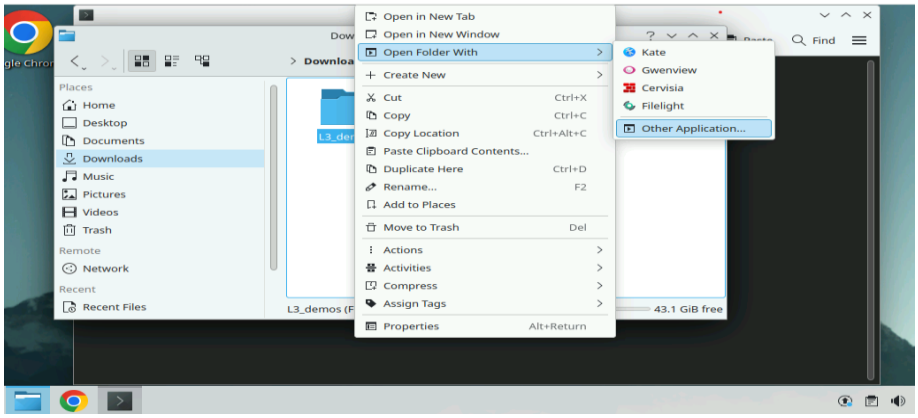
1.2 After this, install and activate the virtual environment using the command below:
python3 -m venv venv
source venv/bin/activate

1.3 Now, create a `requirements.txt` file inside the folder with required libraries:



1.4 Install all the required libraries using the command below:
pip install -r requirements.txt

1.5 Now, open the folder using VS Code editor:



After this, open a new Python file using the "**New File**" option and name it as "**Demo1**".

Step 2: Define customer and support agents

To simulate a customer-agent interaction, define two agents:

2.1 CustomerAgent: Represents a customer seeking support.

2.2 SupportAgent: Acts as an AI-powered customer support assistant connected to Azure OpenAI.

```
import os

import autogen

from dotenv import load_dotenv

# Creating a customer agent that represents a user seeking support.

customer_agent = autogen.UserProxyAgent(

    name="customer",

    human_input_mode="ALWAYS",  # Allows manual input for demonstration purposes.

    code_execution_config = {"use_docker": False},

    max_consecutive_auto_reply= 5

)

# Creating a support agent that will respond to customer queries.

support_agent = autogen.AssistantAgent(

    name="support_agent",

    llm_config={

        "config_list": [

            {

                "api_type": "azure",

                "azure_endpoint": "https://openai-api-management-gw.azure-
api.net/deployments/gpt-4o-mini/chat/completions?api-version=2023-12-01-preview",

                "api_version": "2023-12-01-preview",

                "api_key":
"2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAABACOGi3sC",

                # "deployment_name": "gpt-4o-mini",  # corrected key name

                "model": "gpt-4o-mini",  # optional, but good to keep

            }

        ],

        "temperature": 0.7,

    },

    code_execution_config = {"use_docker": False},

    max_consecutive_auto_reply= 5

)
```

Step 3: Configure the support agent

Configure the behavior of the support agent using the following settings:

3.1 llm_config: Provides Azure endpoint details, API key, model name, and temperature settings to control response style.

3.2 system_message: Adds a structured instruction for the agent to ensure professional and clear customer support interactions.

This setup ensures the support agent delivers helpful, consistent, and appropriately guided responses during chats.

```
# Step 2: Configuring the Support Agent

# Modifying the support agent to follow a structured approach with clear and professional
responses.

support_agent = autogen.AssistantAgent(

    name="support_agent",

    llm_config={

        "config_list": [

            {

                "api_type": "azure",

                "azure_endpoint": "https://openai-api-management-gw.azure-
api.net/deployments/gpt-4o-mini/chat/completions?api-version=2023-12-01-preview",

                "api_version": "2023-12-01-preview",

                "api_key":
"2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAABACOGi3sC",

                # "deployment_name": "gpt-4o-mini",  # corrected key name

                "model": "gpt-4o-mini",  # optional, but good to keep

            }

        ],

        "temperature": 0.7,

    },

    system_message="You are a helpful AI support agent. Answer customer queries clearly
and professionally.",

    code_execution_config = {'use_docker': False},

    max_consecutive_auto_reply= 5

)

# Step 3: Running a Simulated Customer Interaction

# The customer initiates a conversation with the support agent.

customer_agent.initiate_chat(support_agent, message="I need help tracking my order.")
```

Note:
Setting a **system_message** refines the AI's behavior and improves the overall customer experience.

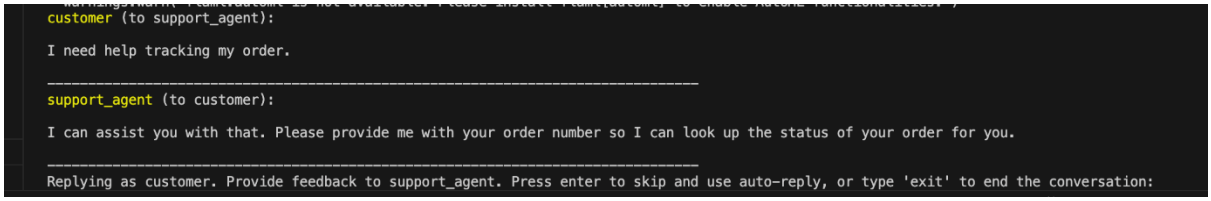
Step 4: Run the code

4.1 Save the file and then run the streamlit webapp from command prompt using the command given below:

streamlit run Demo1.py

Output:

- A simulated customer-agent chat interface.
- Text interaction displayed with AI-generated responses.
- **Caution**: AI-generated content may occasionally be incorrect and should be validated if used in real scenarios.



By following the above-mentioned steps, you have successfully showcased how AutoGen agents can automate customer support interactions, improving efficiency and response times. By configuring AI-driven agents, businesses can handle repetitive queries seamlessly, freeing human agents for complex issues and enhancing overall customer experience.