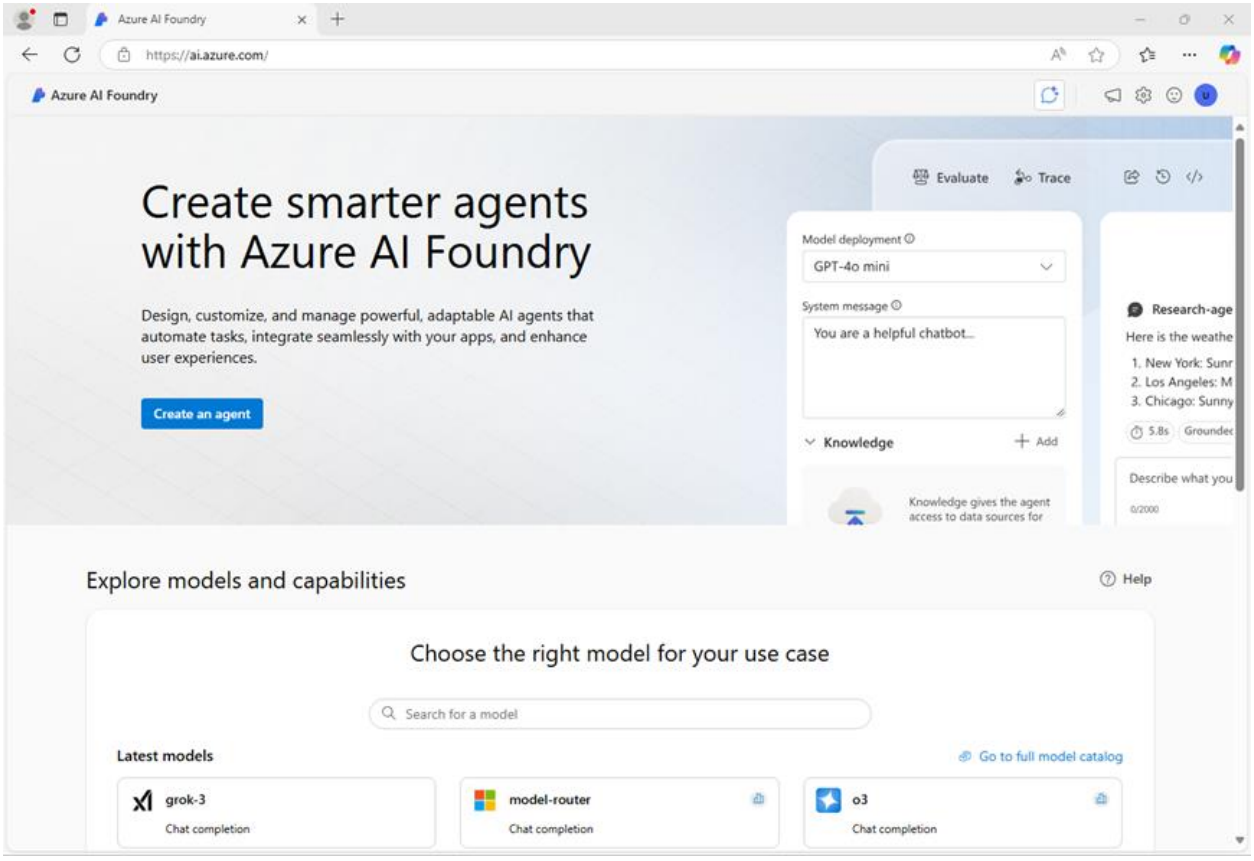
Develop an AI agent

In this exercise, you'll use Azure AI Agent Service to create a simple agent that analyzes data and creates charts. The agent can use the built-in *Code Interpreter* tool to dynamically generate any code required to analyze data.

Create a Foundry project

Let's start by creating a Foundry project.

1. In a web browser, open the Foundry portal at `https://ai.azure.com` and sign in using your Azure credentials. Close any tips or quick start panes that are opened the first time you sign in, and if necessary use the **Foundry** logo at the top left to navigate to the home page, which looks similar to the following image (close the **Help** pane if it's open):



Important: Make sure the **New Foundry** toggle is *Off* for this lab.

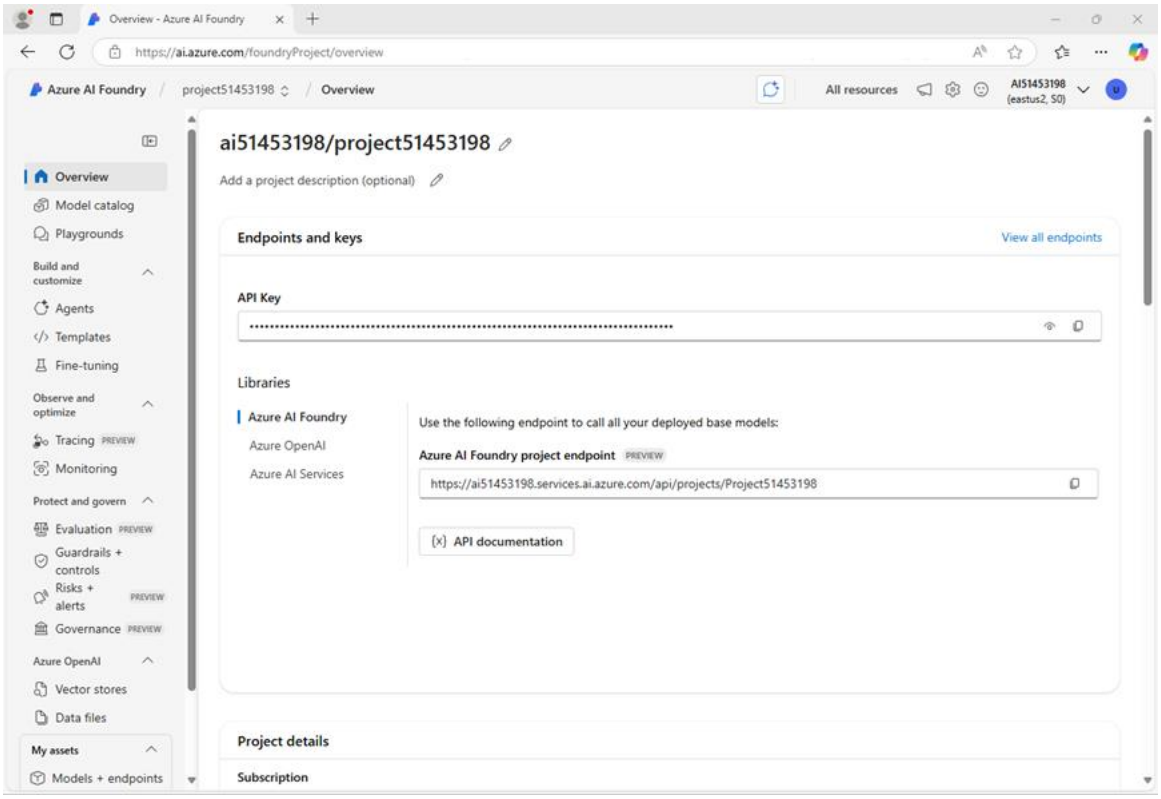
2. In the home page, select **Create an agent**.
3. When prompted to create a project, enter a valid name for your project and expand **Advanced options**.
4. Confirm the following settings for your project:
    a. **Foundry resource**: *A valid name for your Foundry resource*
    b. **Subscription**: *Your Azure subscription*
    c. **Resource group**: *Create or select a resource group*
    d. **Region**: *Select any **AI Foundry recommended****

* Some Azure AI resources are constrained by regional model quotas. In the event of a quota limit being exceeded later in the exercise, there's a possibility you may need to create another resource in a different region.

5. Select **Create** and wait for your project to be created.
6. If prompted, deploy a **gpt-4o** model using either the *Global Standard* or *Standard* deployment option (depending on your quota availability).

Note: If quota is available, a GPT-4o base model may be deployed automatically when creating your Agent and project.

7. When your project is created, the Agents playground will be opened.
8. In the navigation pane on the left, select **Overview** to see the main page for your project; which looks like this:



9. Copy the **Foundry project endpoint** values to a notepad, as you'll use them to connect to your project in a client application.

Create an agent client app

Now you're ready to create a client app that uses an agent. Some code has been provided for you in a GitHub repository.

Clone the repo containing the application code

1. Open a new browser tab (keeping the Foundry portal open in the existing tab). Then in the new tab, browse to the Azure portal at `https://portal.azure.com`; signing in with your Azure credentials if prompted.

Close any welcome notifications to see the Azure portal home page.

2. Use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment with no storage in your subscription.

The cloud shell provides a command-line interface in a pane at the bottom of the Azure portal. You can resize or maximize this pane to make it easier to work in.

Note: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

3. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

Ensure you've switched to the classic version of the cloud shell before continuing.

4. In the cloud shell pane, enter the following commands to clone the GitHub repo containing the code files for this exercise (type the command, or copy it to the clipboard and then right-click in the command line and paste as plain text):

```
rm -r ai-agents -f
git clone https://github.com/KiranAvulasetty/ai-agents  ai-agents
```

Tip: As you enter commands into the cloudshell, the output may take up a large amount of the screen buffer and the cursor on the current line may be obscured. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

5. Enter the following command to change the working directory to the folder containing the code files and list them all.

```
cd ai-agents/Labfiles/02-build-ai-agent/Python
ls -a -l
```

The provided files include application code, configuration settings, and data.

Configure the application settings

1. In the cloud shell command-line pane, enter the following command to install the libraries you'll use:

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-ai-projects azure-ai-agents
```

2. Enter the following command to edit the configuration file that has been provided:

```
code .env
```

The file is opened in a code editor.

3. In the code file, replace the **your_project_endpoint** placeholder with the endpoint for your project (copied from the project **Overview** page in the Foundry portal) and ensure that the MODEL_DEPLOYMENT_NAME variable is set to your model deployment name (which should be *gpt-4o*).
4. After you've replaced the placeholder, use the **CTRL+S** command to save your changes and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.

Write code for an agent app

Tip: As you add code, be sure to maintain the correct indentation. Use the comment indentation levels as a guide.

1. Enter the following command to edit the code file that has been provided:

```
code agent.py
```

2. Review the existing code, which retrieves the application configuration settings and loads data from *data.txt* to be analyzed. The rest of the file includes comments where you'll add the necessary code to implement your data analysis agent.
3. Find the comment **Add references** and add the following code to import the classes you'll need to build an Azure AI agent that uses the built-in code interpreter tool:

```
# Add references
from azure.identity import DefaultAzureCredential
from azure.ai.agents import AgentsClient
from azure.ai.agents.models import FilePurpose, CodeInterpreterTool, ListSortOrder,
MessageRole
```

4. Find the comment **Connect to the Agent client** and add the following code to connect to the Azure AI project.

Tip: Be careful to maintain the correct indentation level.

```
# Connect to the Agent client
agent_client = AgentsClient(
   endpoint=project_endpoint,
   credential=DefaultAzureCredential
       (exclude_environment_credential=True,
        exclude_managed_identity_credential=True)
)
with agent_client:
```

The code connects to the Foundry project using the current Azure credentials. The final *with agent_client* statement starts a code block that defines the scope of the client, ensuring it's cleaned up when the code within the block is finished.

5. Find the comment **Upload the data file and create a CodeInterpreterTool**, within the *with agent_client* block, and add the following code to upload the data file to the project and create a CodeInterpreterTool that can access the data in it:

```
# Upload the data file and create a CodeInterpreterTool
file = agent_client.files.upload_and_poll(
    file_path=file_path, purpose=FilePurpose.AGENTS
)
print(f"Uploaded {file.filename}")

code_interpreter = CodeInterpreterTool(file_ids=[file.id])
```

6. Find the comment **Define an agent that uses the CodeInterpreterTool** and add the following code to define an AI agent that analyzes data and can use the code interpreter tool you defined previously:

```
# Define an agent that uses the CodeInterpreterTool
agent = agent_client.create_agent(
    model=model_deployment,
    name="data-agent",
    instructions="You are an AI agent that analyzes the data in the file that has
been uploaded. Use Python to calculate statistical metrics as necessary.",
    tools=code_interpreter.definitions,
    tool_resources=code_interpreter.resources,
)
print(f"Using agent: {agent.name}")
```

7. Find the comment **Create a thread for the conversation** and add the following code to start a thread on which the chat session with the agent will run:

```
# Create a thread for the conversation
thread = agent_client.threads.create()
```

8. Note that the next section of code sets up a loop for a user to enter a prompt, ending when the user enters "quit".
9. Find the comment **Send a prompt to the agent** and add the following code to add a user message to the prompt (along with the data from the file that was loaded previously), and then run thread with the agent.

```
# Send a prompt to the agent
message = agent_client.messages.create(
    thread_id=thread.id,
    role="user",
    content=user_prompt,
)

run = agent_client.runs.create_and_process(thread_id=thread.id, agent_id=agent.id)
```

10. Find the comment **Check the run status for failures** and add the following code to check for any errors.

```
# Check the run status for failures
if run.status == "failed":
    print(f"Run failed: {run.last_error}")
```

11. Find the comment **Show the latest response from the agent** and add the following code to retrieve the messages from the completed thread and display the last one that was sent by the agent.

```
# Show the latest response from the agent
last_msg = agent_client.messages.get_last_message_text_by_role(
   thread_id=thread.id,
   role=MessageRole.AGENT,
)
if last_msg:
   print(f"Last Message: {last_msg.text.value}")
```

12. Find the comment **Get the conversation history**, which is after the loop ends, and add the following code to print out the messages from the conversation thread; reversing the order to show them in chronological sequence

```
# Get the conversation history
print("\nConversation Log:\n")
messages = agent_client.messages.list(thread_id=thread.id,
order=ListSortOrder.ASCENDING)
for message in messages:
   if message.text_messages:
       last_msg = message.text_messages[-1]
       print(f"{message.role}: {last_msg.text.value}\n")
```

13. Find the comment **Clean up** and add the following code to delete the agent and thread when no longer needed.

```
# Clean up
agent_client.delete_agent(agent.id)
```

14. Review the code, using the comments to understand how it:
    a. Connects to the AI Foundry project.
    b. Uploads the data file and creates a code interpreter tool that can access it.
    c. Creates a new agent that uses the code interpreter tool and has explicit instructions to use Python as necessary for statistical analysis.
    d. Runs a thread with a prompt message from the user along with the data to be analyzed.
    e. Checks the status of the run in case there's a failure
    f. Retrieves the messages from the completed thread and displays the last one sent by the agent.
    g. Displays the conversation history
    h. Deletes the agent and thread when they're no longer required.
15. Save the code file (*CTRL+S*) when you have finished. You can also close the code editor (*CTRL+Q*); though you may want to keep it open in case you need to make any edits to the code you added. In either case, keep the cloud shell command-line pane open.

Sign into Azure and run the app

1. In the cloud shell command-line pane, enter the following command to sign into Azure.

```
az login
```

You must sign into Azure - even though the cloud shell session is already authenticated.

Note: In most scenarios, just using *az login* will be sufficient. However, if you have subscriptions in multiple tenants, you may need to specify the tenant by using the *--tenant* parameter. See Sign into Azure interactively using the Azure CLI for details.

2. When prompted, follow the instructions to open the sign-in page in a new tab and enter the authentication code provided and your Azure credentials. Then complete the sign in process in the command line, selecting the subscription containing your Foundry hub if prompted.
3. After you have signed in, enter the following command to run the application:

```
python agent.py
```

The application runs using the credentials for your authenticated Azure session to connect to your project and create and run the agent.

4. When prompted, view the data that the app has loaded from the *data.txt* text file. Then enter a prompt such as:

```
What's the category with the highest cost?
```

Tip: If the app fails because the rate limit is exceeded. Wait a few seconds and try again. If there is insufficient quota available in your subscription, the model may not be able to respond.

5. View the response. Then enter another prompt, this time requesting a visualization:

```
Create a text-based bar chart showing cost by category
```

6. View the response. Then enter another prompt, this time requesting a statistical metric:

```
What's the standard deviation of cost?
```

View the response.

7. You can continue the conversation if you like. The thread is *stateful*, so it retains the conversation history - meaning that the agent has the full context for each response. Enter `quit` when you're done.
8. Review the conversation messages that were retrieved from the thread - which may include messages the agent generated to explain its steps when using the code interpreter tool.

Summary

In this exercise, you used the Azure AI Agent Service SDK to create a client application that uses an AI agent. The agent can use the built-in Code Interpreter tool to run dynamic Python code to perform statistical analyses.