Create an Azure AI Foundry hub and project

Download content

Extract information from invoice documents

Extract information from a slide image

Extract information from a voicemail audio recording

Extract information from a video conference recording

Clean up

Extract information from multimodal content

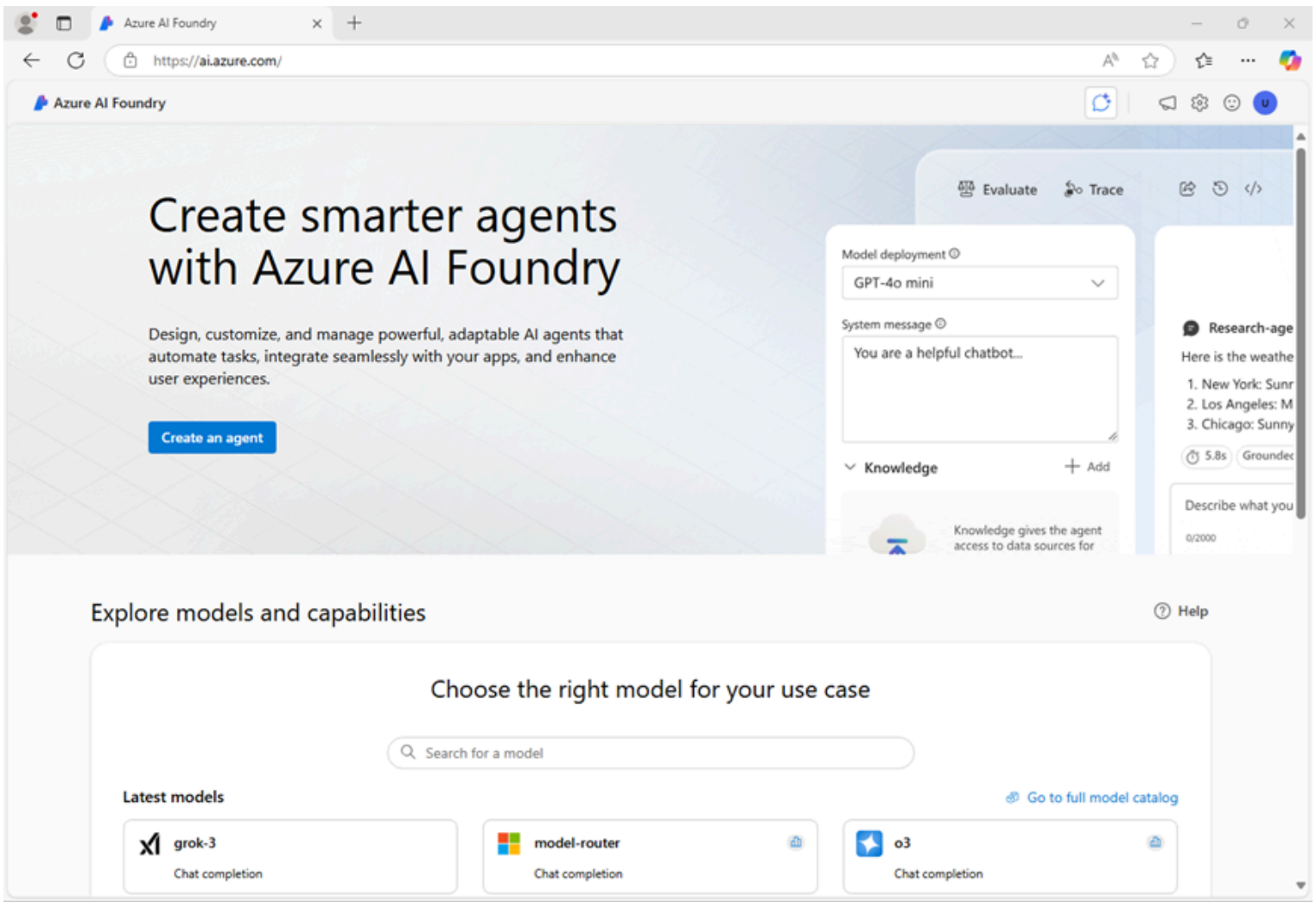
In this exercise, you use Azure Content Understanding to extract information from a variety of content types; including an invoice, an images of a slide containing charts, an audio recording of a voice messages, and a video recording of a conference call.

This exercise takes approximately **40** minutes.

Create an Azure AI Foundry hub and project

The features of Azure AI Foundry we're going to use in this exercise require a project that is based on an Azure AI Foundry *hub* resource.

1. In a web browser, open the Azure AI 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 **Azure AI 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):



2. In the browser, navigate to `https://ai.azure.com/managementCenter/allResources` and select **Create new**. Then choose the option to create a new **AI hub resource**.
3. In the **Create a project** wizard, enter a valid name for your project, and select the option to create a new hub. Then use the **Rename hub** link to specify a valid name for your new hub, expand **Advanced options**, and specify the following settings for your project:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: Select one of the following locations (*At the time of writing, Azure AI Content understanding is only available in these regions*):
        - Australia East
        - Sweden Central
        - West US

> **Note**: If you're working in an Azure subscription in which policies are used to restrict allowable resource names, you may need to use the link at the bottom of the **Create a new project** dialog box to create the hub using the Azure portal.

> **Tip**: If the **Create** button is still disabled, be sure to rename your hub to a unique alphanumeric value.

4. Wait for your project to be created.

Download content

The content you're going to analyze is in a .zip archive. Download it and extract it in a local folder.

1. In a new browser tab, download content.zip from
   ```
   https://github.com/microsoftlearning/mslearn-ai-information-extraction/raw/main/Labfiles/content/content.zip
   ```
   and save it in a local folder.
2. Extract the downloaded *content.zip* file and view the files it contains. You'll use these files to build various Content Understanding analyzers in this exercise.

> **Note**: If you're only interested in exploring analysis of a specific modality (documents, images, video, or audio), you can skip to the relevant task below. For the best experience, go through each task to learn how to extract information from different types of content.

Extract information from invoice documents

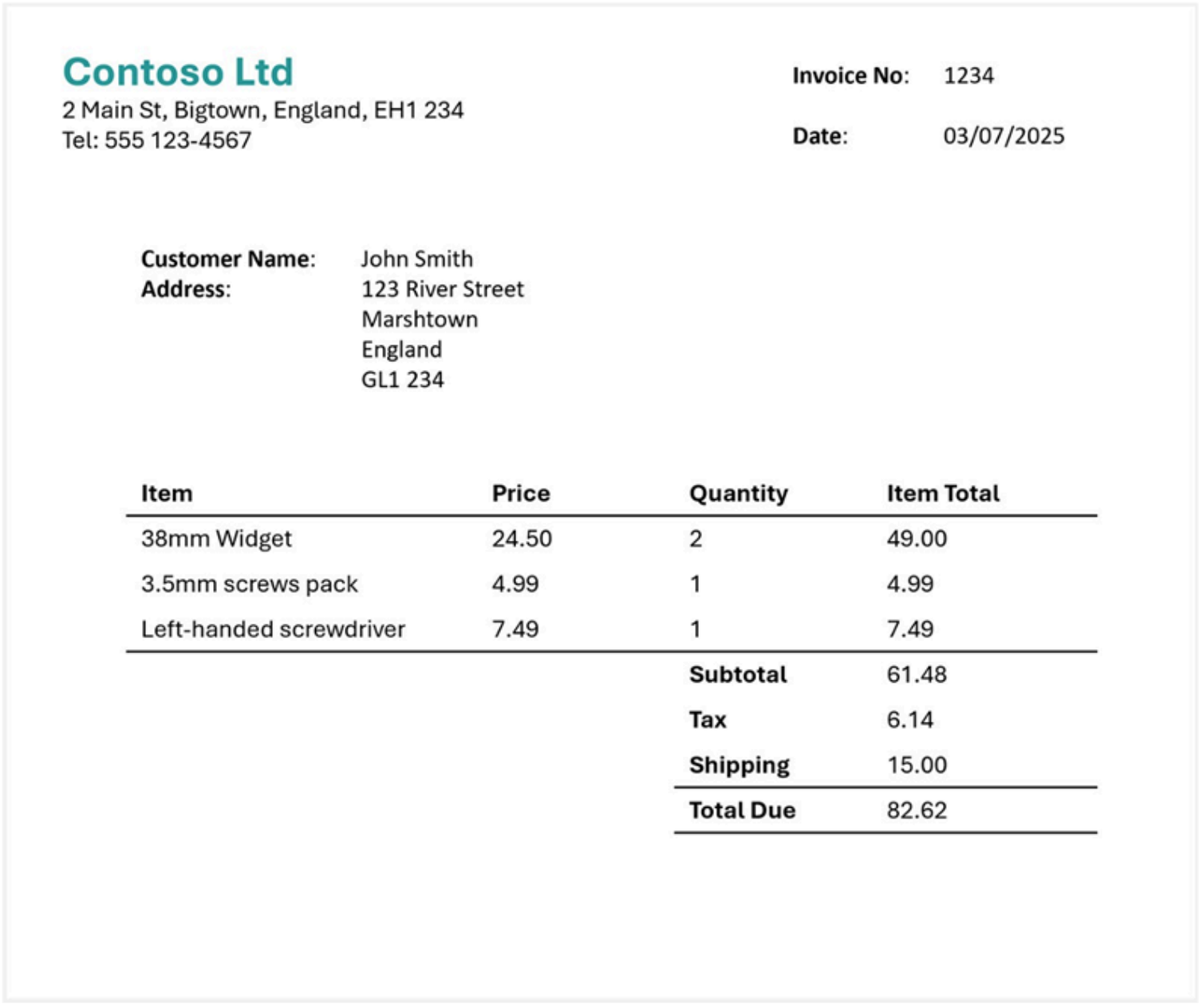
You are going to build an Azure AI Content Understanding analyzer that can extract information from invoices. You'll start by defining a schema based on a sample invoice.

Define a schema for invoice analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
   - **Task name**: `Invoice analysis`
   - **Description**: `Extract data from an invoice`
   - **Single file content analysis**: *Selected*
   - **Advanced settings**:
     - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
     - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

> **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **invoice-1234.pdf** file from the folder where you extracted content files. This file contains the following invoice:



Contoso Ltd
2 Main St, Bigtown, England, EH1 234
Tel: 555 123-4567

Invoice No: 1234
Date: 03/07/2025

Customer Name: John Smith
Address: 123 River Street
Marshtown
England
GL1 234

Item	Price	Quantity	Item Total
38mm Widget	24.50	2	49.00
3.5mm screws pack	4.99	1	4.99
Left-handed screwdriver	7.49	1	7.49
		Subtotal	61.48
		Tax	6.14
		Shipping	15.00
		Total Due	82.62

6. On the **Define schema** page, after uploading the invoice file, select the **Invoice data extraction** template and select **Create**.

   The *Invoice analysis* template includes common fields that are found in invoices. You can use the schema editor to delete any of the suggested fields that you don't need, and add any custom fields that you do.
7. In the list of suggested fields, select **BillingAddress**. This field is not needed for the invoice format you have uploaded, so use the **Delete field** (🗑) icon that appears in the selected field row to delete it.
8. Now delete the following suggested fields, which aren't needed for your invoice schema:

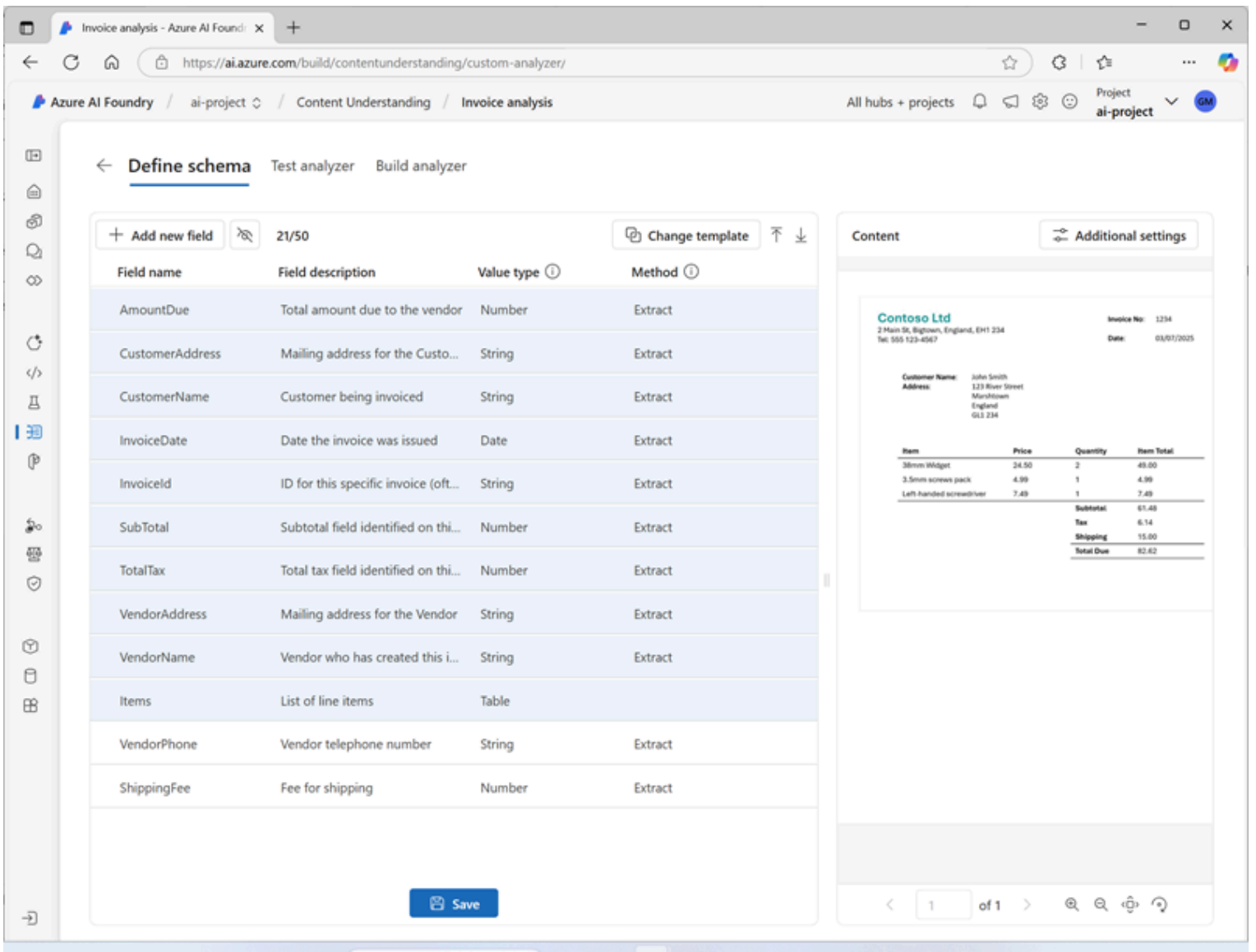
   - BillingAddressRecipient
   - CustomerAddressRecipient
   - CustomerId
   - CustomerTaxId
   - DueDate
   - InvoiceTotal
   - PaymentTerm
   - PreviousUnpaidBalance
   - PurchaseOrder
   - RemittanceAddress
   - RemittanceAddressRecipient
   - ServiceAddress
   - ServiceAddressRecipient
   - ShippingAddress
   - ShippingAddressRecipient
   - TotalDiscount
   - VendorAddressRecipient
   - VendorTaxId
   - TaxDetails

9. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

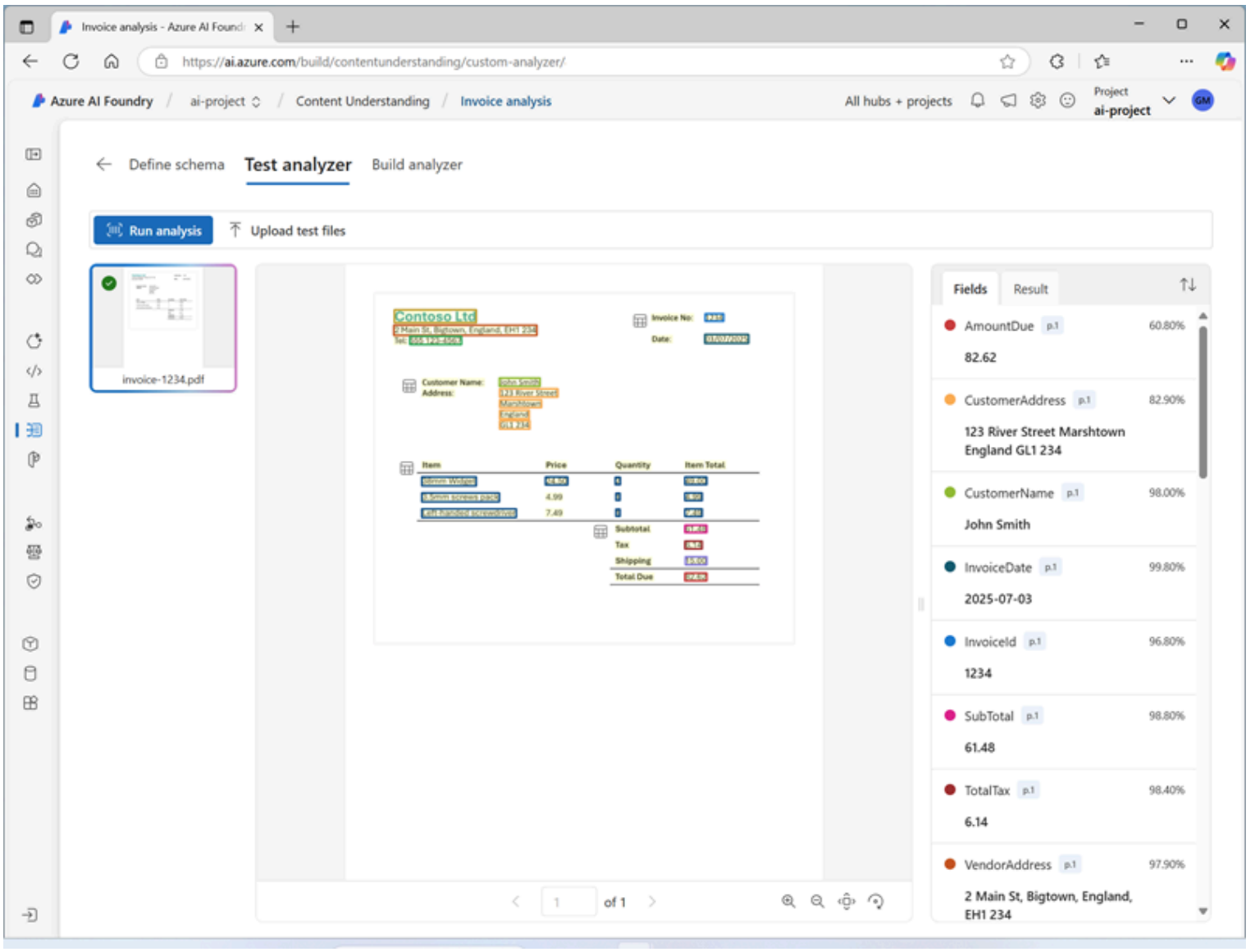
Field name	Field description	Value type	Method
`VendorPhone`	`Vendor telephone number`	String	Extract

Field name	Field description	Value type	Method
`ShippingFee`	`Fee for shipping`	Number	Extract

10. In the row for the **Items** field, note that this field is a *table* (it contains the collection of items in the invoice). Select it's **Edit** (▦) icon to open a new page with its subfields.
11. Remove the following subfields from the **Items** table:
    - Date
    - ProductCode
    - Unit
    - TaxAmount
    - TaxRate
12. Use the **OK** button to confirm the changes and return to the top-level of the invoice schema.
13. Verify that your completed schema looks like this, and select **Save**.



14. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.
15. Review the analysis results, which should look similar to this:



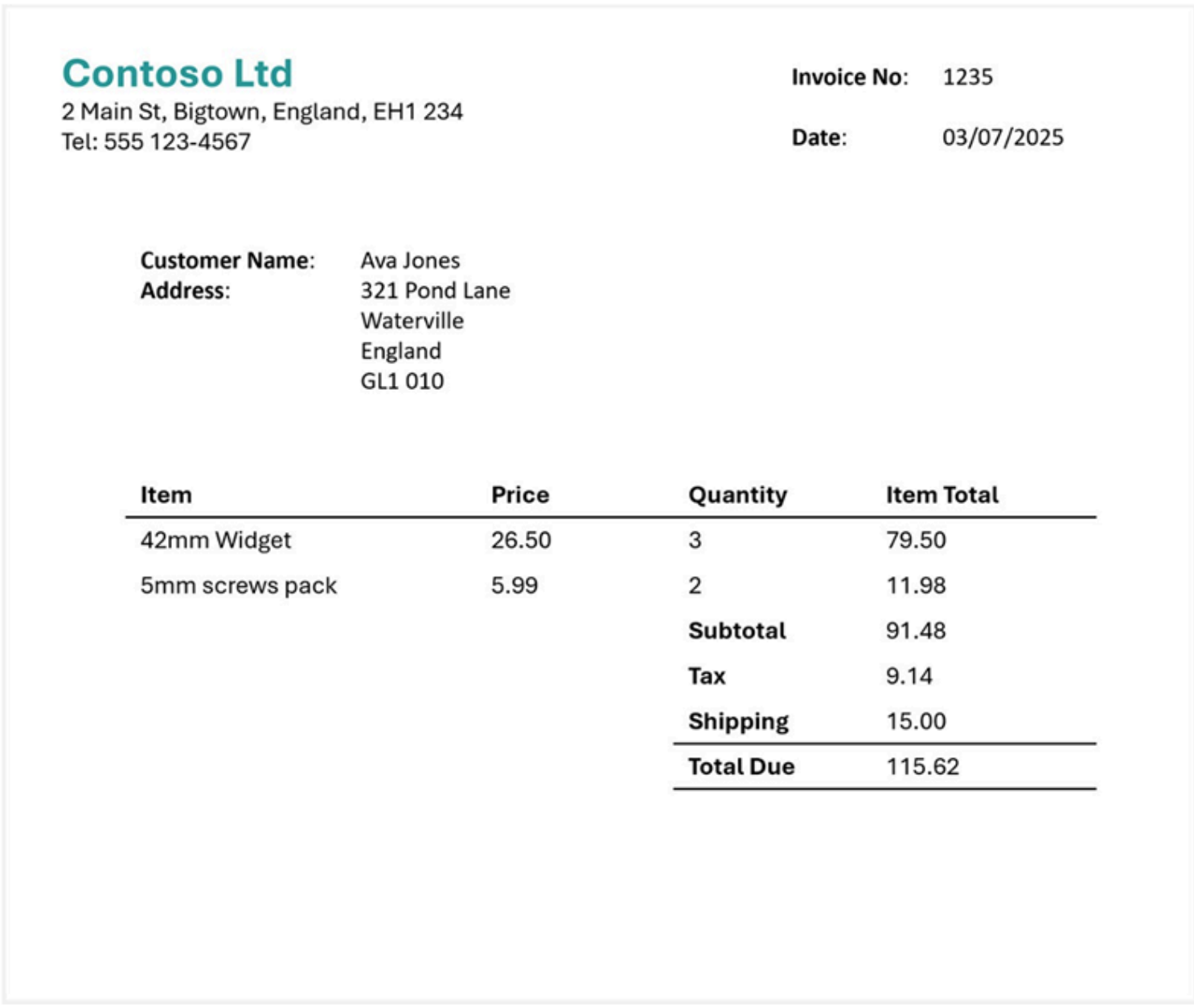
16. View the details of the fields that were identified in the **Fields** pane.

Build and test an analyzer for invoices

Now that you have trained a model to extract fields from invoices, you can build an analyzer to use with similar documents.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `invoice-analyzer`
    - **Description**: `Invoice analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **invoice-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **invoice-analyzer** page, select the **Test** tab.
5. Use the **+ Upload test files** button to upload **invoice-1235.pdf** from the folder where you extracted the content files, and click on **Run analysis** to extract field data from the invoice.

The invoice being analyzed looks like this:



Contoso Ltd
2 Main St, Bigtown, England, EH1 234
Tel: 555 123-4567

Invoice No: 1235
Date: 03/07/2025

Customer Name: Ava Jones
Address: 321 Pond Lane
Waterville
England
GL1 010

Item	Price	Quantity	Item Total
42mm Widget	26.50	3	79.50
5mm screws pack	5.99	2	11.98
		Subtotal	91.48
		Tax	9.14
		Shipping	15.00
		Total Due	115.62

6. Review the **Fields** pane, and verify that the analyzer extracted the correct fields from the test invoice.
7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. On the **Code example** tab, view the sample code that you could use to develop a client application that uses the Content Understanding REST interface to call your analyzer.
9. Close the **invoice-analyzer** page.

Extract information from a slide image

You are going to build an Azure AI Content Understanding analyzer that can extract information from a slide containing charts.

Define a schema for image analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
    - **Task name**: `Slide analysis`
    - **Description**: `Extract data from an image of a slide`
    - **Single file content analysis**: *Selected*
    - **Advanced settings**:
        - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
        - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

> **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **slide-1.jpg** file from the folder where you extracted content files. Then select the **Image analysis** template and select **Create**.

The *Image analysis* template doesn't include any predefined fields. You must define fields to describe the information you want to extract.

6. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

Field name	Field description	Value type	Method
`Title`	`Slide title`	String	Generate
`Summary`	`Summary of the slide`	String	Generate
`Charts`	`Number of charts on the slide`	Integer	Generate

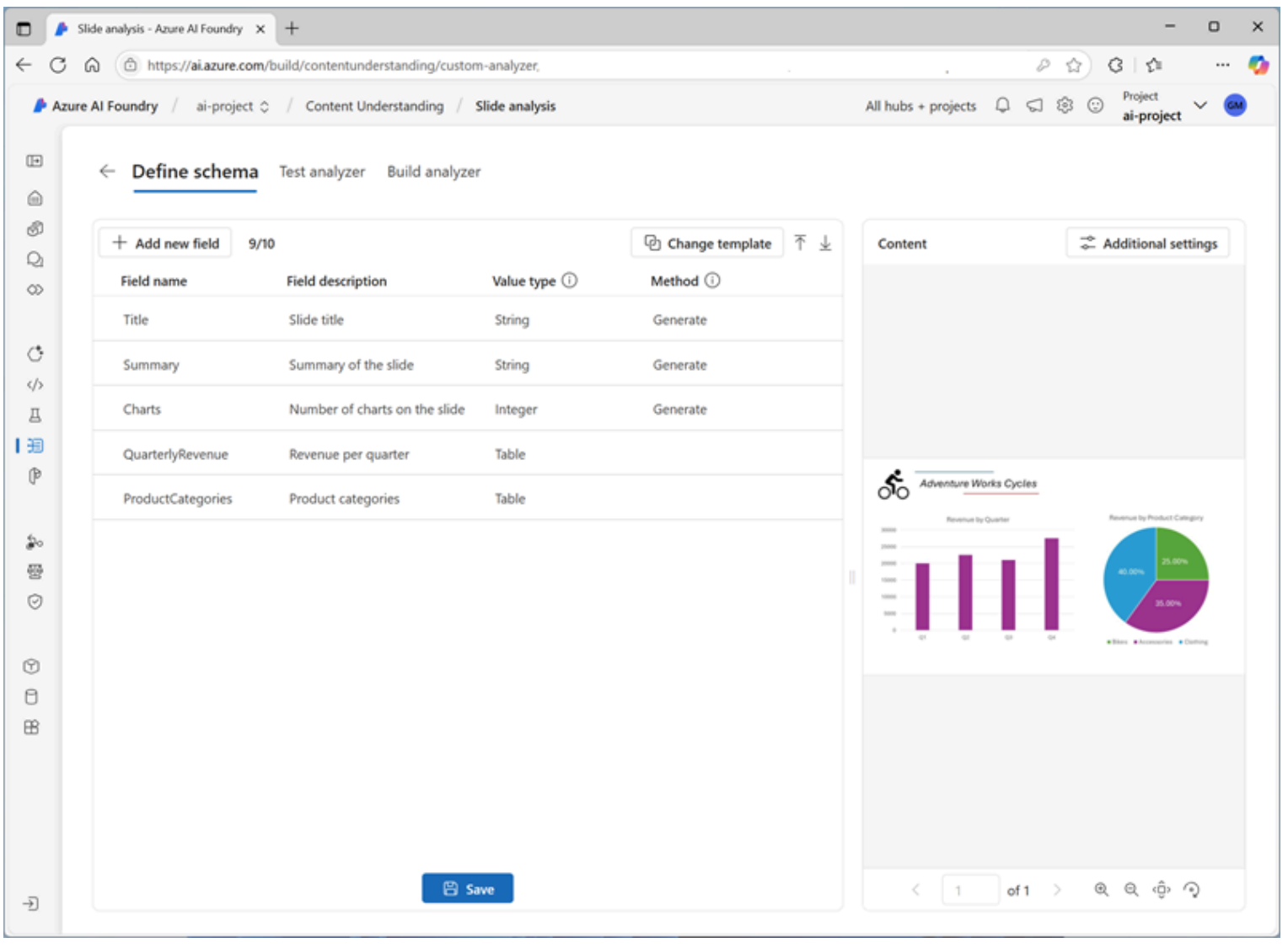
7. Use **+ Add new field** button to add a new field named `QuarterlyRevenue` with the description `Revenue per quarter` with the value type **Table**, and save the new field (✓). Then, in the new page for the table subfields that opens, add the following subfields:

Field name	Field description	Value type	Method
`Quarter`	`Which quarter?`	String	Generate
`Revenue`	`Revenue for the quarter`	Number	Generate

8. Select **Back** (the arrow icon near the **Add new subfield** button) or ✓ **OK** to return to the top level of your schema, and use **+ Add new field** button to add a new field named `ProductCategories` with the description `Product categories` with the value type **Table**, and save the new field (✓). Then, in the new page for the table subfields that opens, add the following subfields:

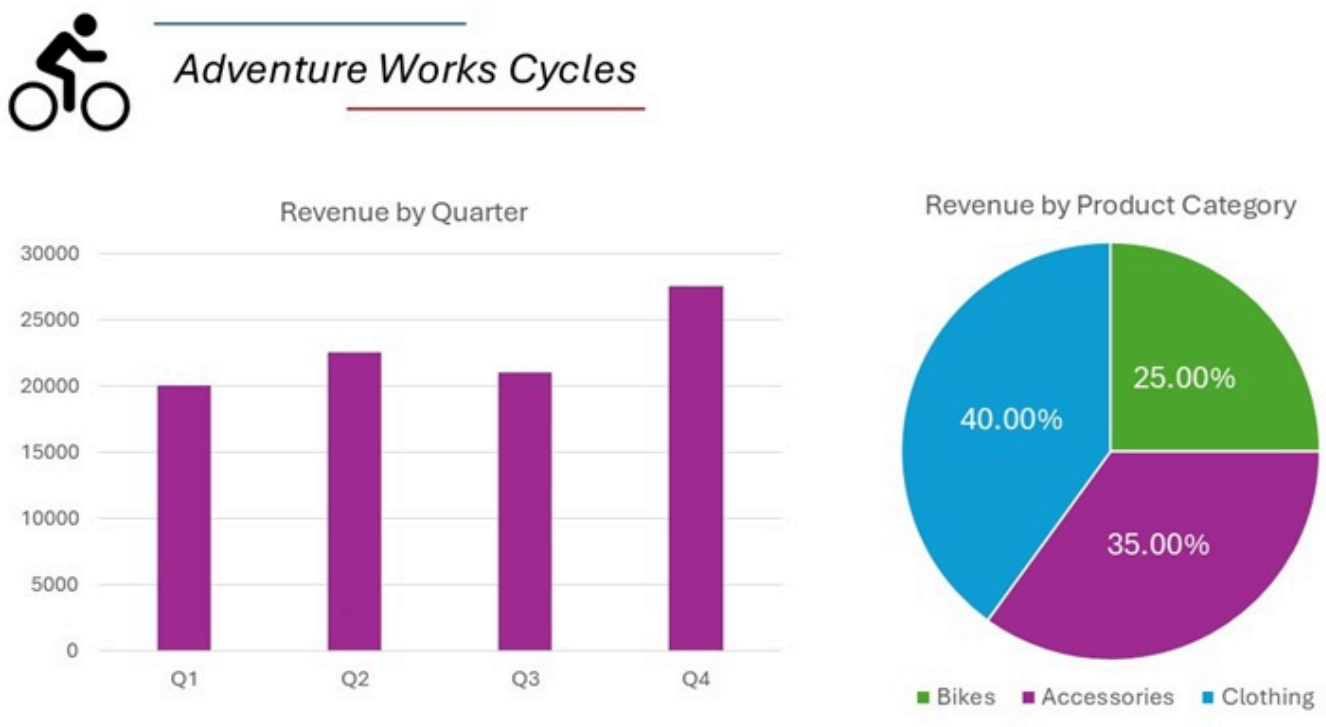
Field name	Field description	Value type	Method
`ProductCategory`	`Product category name`	String	Generate
`RevenuePercentage`	`Percentage of revenue`	Number	Generate

9. Select **Back** (the arrow icon near the **Add new subfield** button) or ✓ **OK** to return to the top level of your schema, and verify that it looks like this. Then select **Save**.

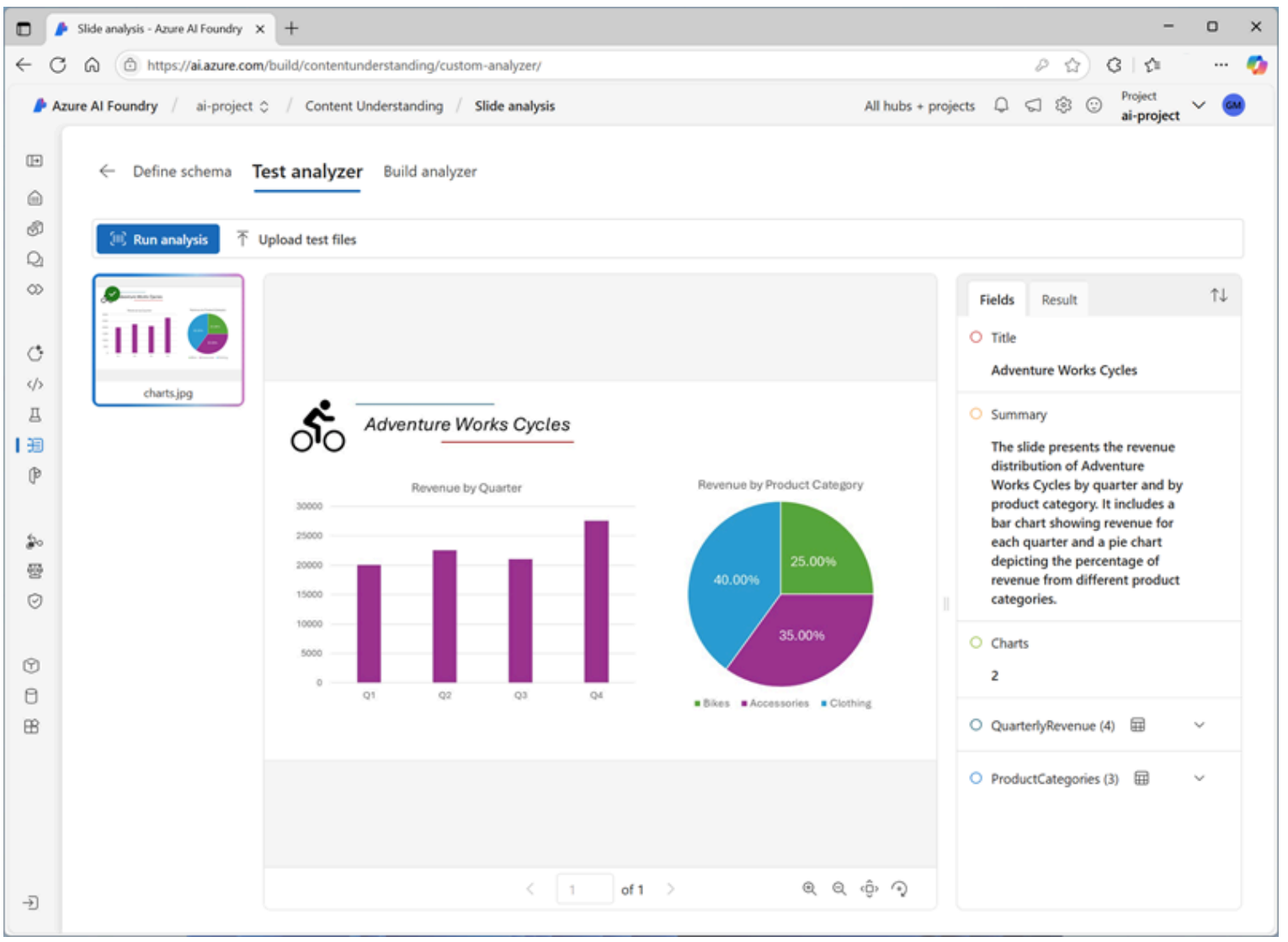


10. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.

The slide being analyzed looks like this:



11. Review the analysis results, which should look similar to this:



12. View the details of the fields that were identified in the **Fields** pane, expanding the **QuarterlyRevenue** and **ProductCategories** fields to see the subfield values.

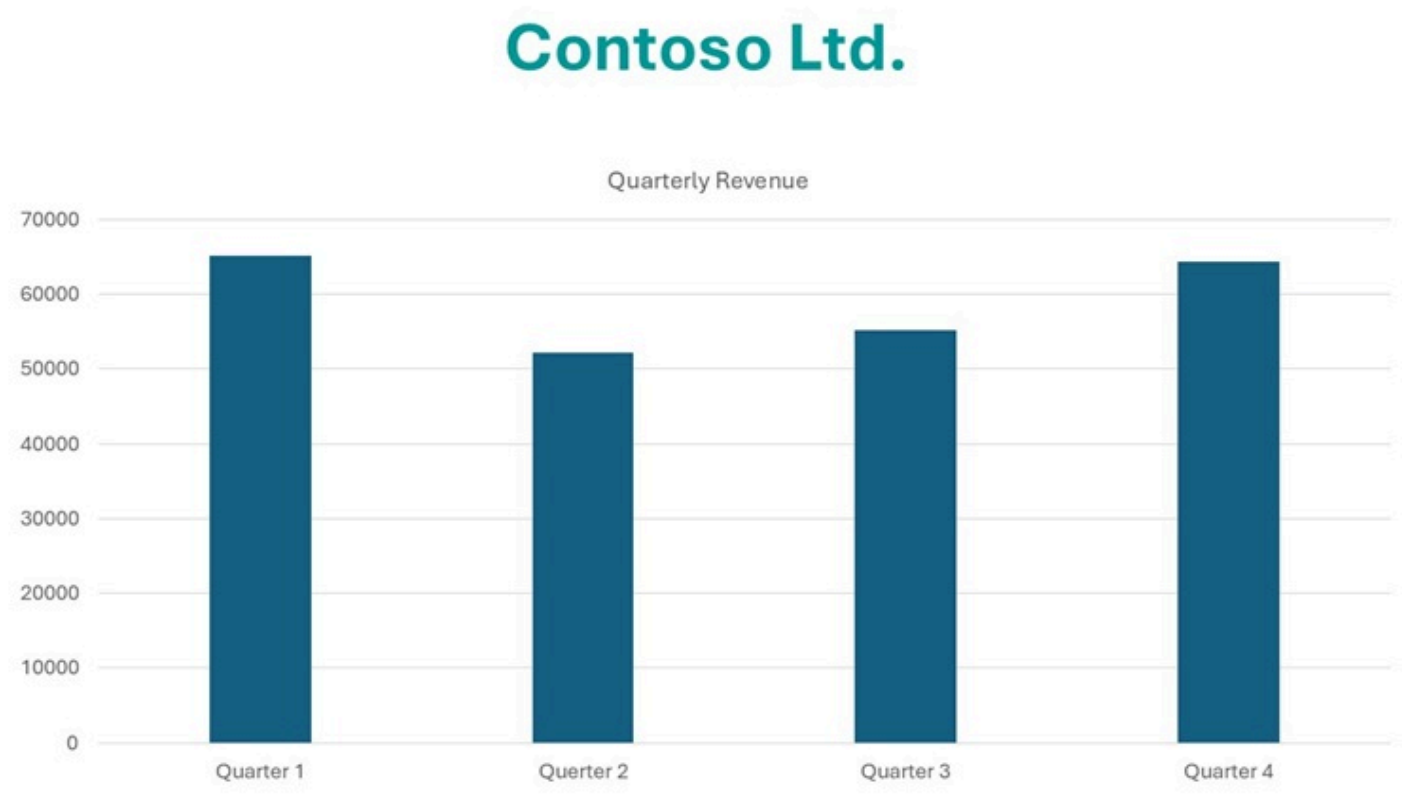
Build and test an analyzer

Now that you have trained a model to extract fields from slides, you can build an analyzer to use with similar slide images.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `slide-analyzer`
    - **Description**: `Slide image analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **slide-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **slide-analyzer** page, select the **Test** tab.

5. Use the **+ Upload test files** button to upload **slide-2.jpg** from the folder where you extracted the content files, and click on **Run analysis** to extract field data from the image.

   The slide being analyzed looks like this:



6. Review the **Fields** pane, and verify that the analyzer extracted the correct fields from the slide image.

   > **Note**: Slide 2 doesn't include a breakfown by product category, so the product category revenue data is not found.

7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. On the **Code example** tab, view the sample code that you could use to develop a client application that uses the Content understanding REST interface to call your analyzer.
9. Close the **slide-analyzer** page.

Extract information from a voicemail audio recording

You are going to build an Azure AI Content Understanding analyzer that can extract information from an audio recording of a voicemail message.

Define a schema for audio analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
   - **Task name**: `Voicemail analysis`
   - **Description**: `Extract data from a voicemail recording`
   - **Single file content analysis**: *Selected*
   - **Advanced settings**:
     - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
     - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

   > **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **call-1.mp3** file from the folder where you extracted content files. Then select the **Speech transcript analysis** template and select **Create**.

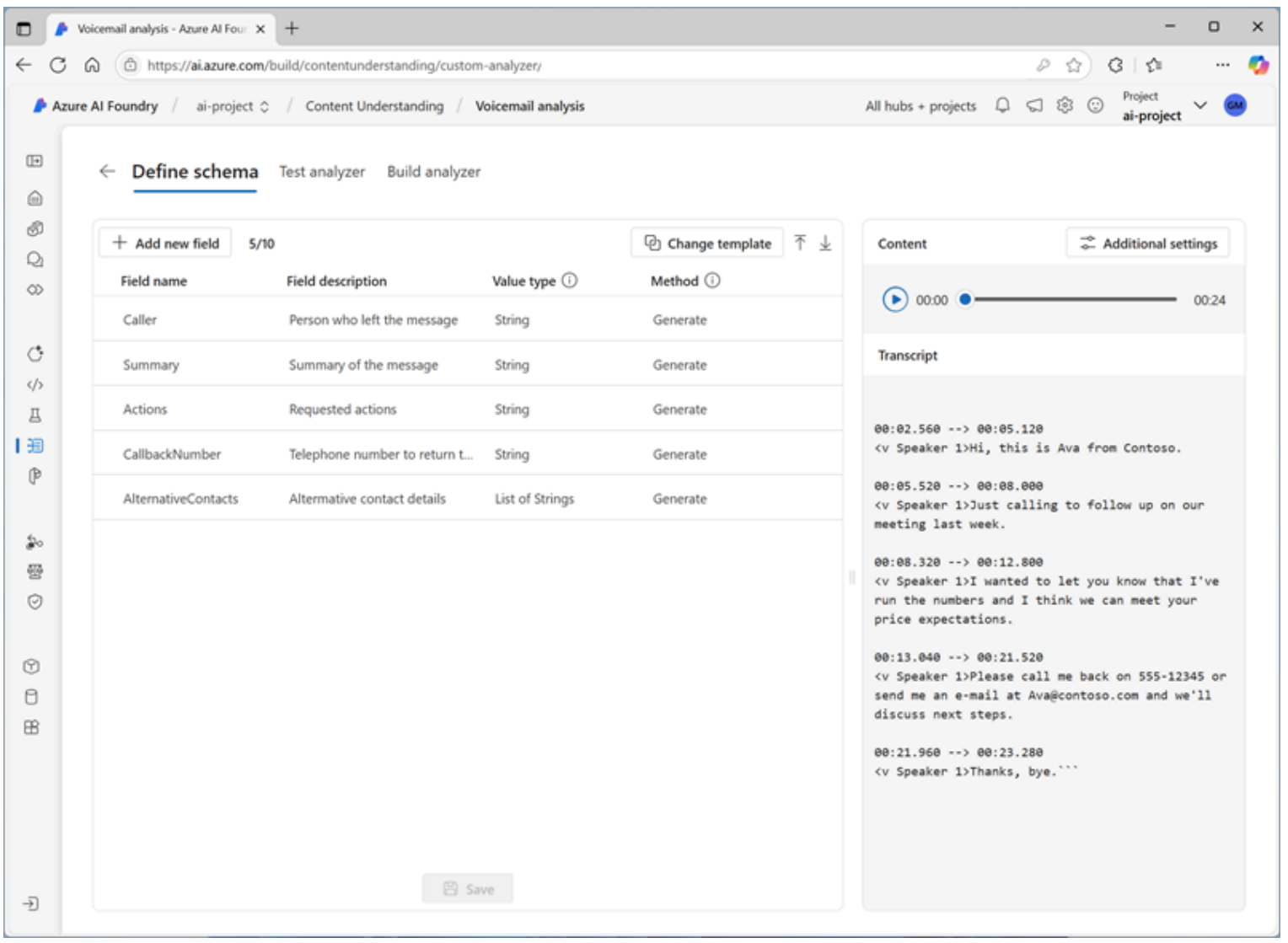
6. In the **Content** pane on the right, select **Get transcription preview** to see a transcription of the recorded message.

   The *Speech transcript analysis* template doesn't include any predefined fields. You must define fields to describe the information you want to extract.

7. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

Field name	Field description	Value type	Method
`Caller`	`Person who left the message`	String	Generate
`Summary`	`Summary of the message`	String	Generate
`Actions`	`Requested actions`	String	Generate
`CallbackNumber`	`Telephone number to return the call`	String	Generate
`AlternativeContacts`	`Alternative contact details`	List of Strings	Generate

8. Verify that your schema looks like this. Then select **Save**.



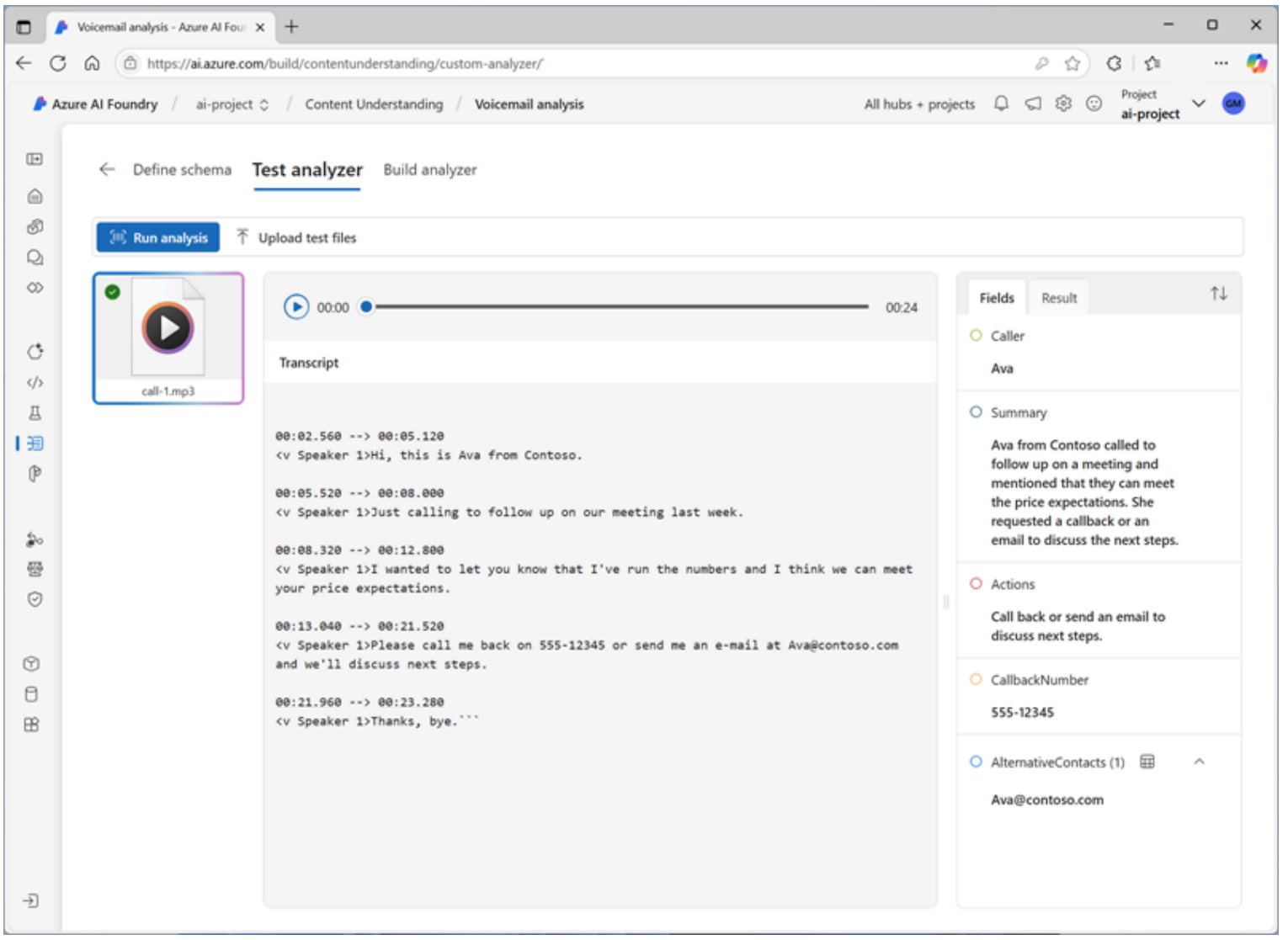
9. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.

   Audio analysis can take some time. While you're waiting, you can play the audio file below:

   0:00

   Note: This audio was generated using AI.

10. Review the analysis results, which should look similar to this:



11. View the details of the fields that were identified in the **Fields** pane, expanding the **AlternativeContacts** field to see the listed values.

Build and test an analyzer

Now that you have trained a model to extract fields from voice messages, you can build an analyzer to use with similar audio recordings.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `voicemail-analyzer`
    - **Description**: `Voicemail audio analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **voicemail-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **voicemail-analyzer** page, select the **Test** tab.
5. Use the **+ Upload test files** button to upload **call-2.mp3** from the folder where you extracted the content files, and click on **Run analysis** to extract field data from the audio file.

   Audio analysis can take some time. While you're waiting, you can play the audio file below:

   0:00

   Note: This audio was generated using AI.
6. Review the **Fields** pane, and verify that the analyzer extracted the correct fields from the voice message.
7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. On the **Code example** tab, view the sample code that you could use to develop a client application that uses the Content understanding REST interface to call your analyzer.
9. Close the **voicemail-analyzer** page.

Extract information from a video conference recording

You are going to build an Azure AI Content Understanding analyzer that can extract information from an video recording of a conference call.

Define a schema for video analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
    - **Task name**: `Conference call video analysis`
    - **Description**: `Extract data from a video conference recording`
    - **Single file content analysis**: *Selected*
    - **Advanced settings**:
        - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
        - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

> **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **meeting-1.mp4** file from the folder where you extracted content files. Then select the **Video analysis** template and select **Create**.
6. In the **Content** pane on the right, select **Get transcription preview** to see a transcription of the recorded message.

   The *Video analysis* template extracts data for the video. It doesn't include any predefined fields. You must define fields to describe the information you want to extract.

7. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

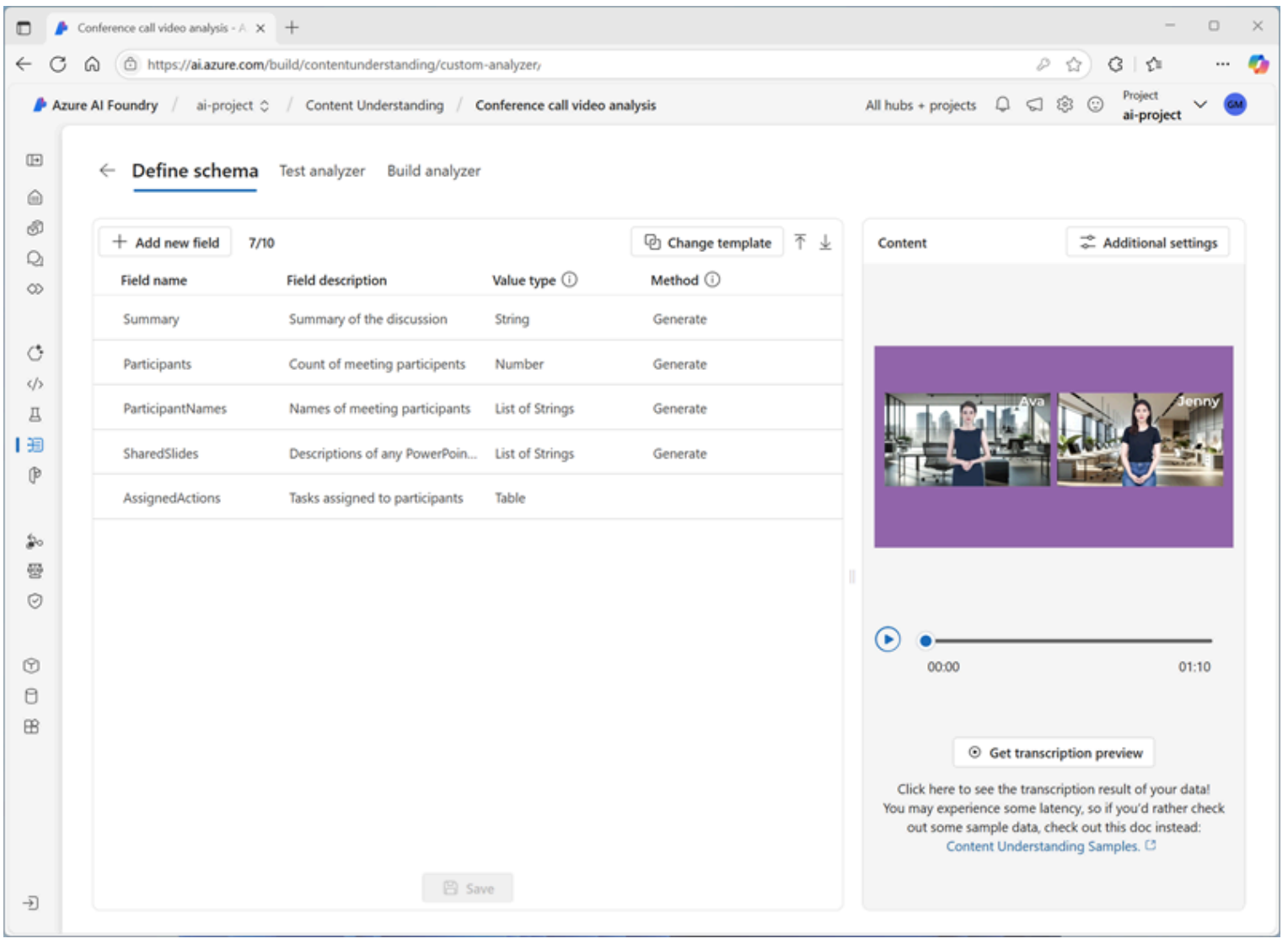
Field name	Field description	Value type	Method
`Summary`	`Summary of the discussion`	String	Generate
`Participants`	`Count of meeting participants`	Integer	Generate
`ParticipantNames`	`Names of meeting participants`	List of Strings	Generate
`SharedSlides`	`Descriptions of any PowerPoint slides presented`	List of Strings	Generate
`AssignedActions`	`Tasks assigned to participants`	Table	

8. When you enter the **AssignedActions** field, in the table of subfields that appears, create the following subfields:

Field name	Field description	Value type	Method
`Task`	`Description of the task`	String	Generate
`AssignedTo`	`Who the task is assigned to`	String	Generate

9. Select **Back** (the arrow icon near the **Add new subfield** button) or ✓ **OK** to return to the top level of your schema, and verify that it looks like this. Then select **Save**.

10. Verify that your schema looks like this. Then select **Save**.



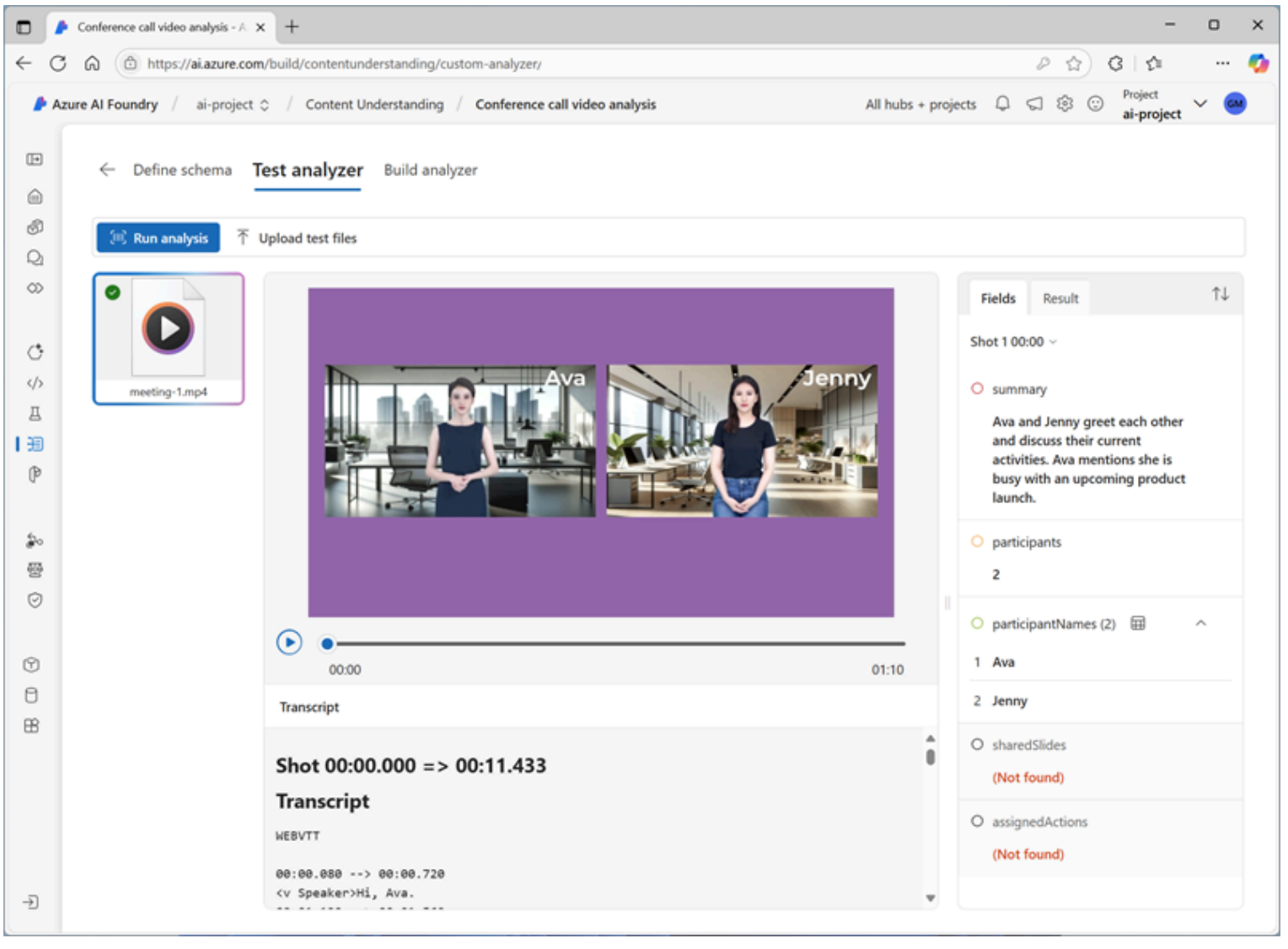
11. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.

    Video analysis can take some time. While you're waiting, you can view the video below:

    0:00

    Note: This video was generated using AI.

12. When analysis is complete, review the results, which should look similar to this:



13. In the **Fields** pane, view the extracted data for the video, including the fields you added. View the field values that were generated, expanding list and table fields as necessary.

Build and test an analyzer

Now that you have trained a model to extract fields from conference call recordings, you can build an analyzer to use with similar videos.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `conference-call-analyzer`
    - **Description**: `Conference call video analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **conference-call-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **conference-call-analyzer** page, select the **Test** tab.
5. Use the **Upload test files** button to upload **meeting-2.mp4** from the folder where you extracted the content files, and run the analysis to extract field data from the audio file.

    Video analysis can take some time. While you're waiting, you can view the video below:

    0:00

    Note: This video was generated using AI.
6. Review the **Fields** pane, and view the fields that the analyzer extracted for the conference call video.
7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. Close the **conference-call-analyzer** page.

Clean up

If you've finished working with the Content Understanding service, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. In the Azure AI Foundry portal, navigate to your hub, in the overview page, select your project and delete it.
2. In the Azure portal, delete the resource group you created in this exercise.

Develop a Content Understanding client application

Create an Azure AI Foundry hub and project

Use the REST API to create a Content Understanding analyzer

Use the REST API to analyze content

Clean up

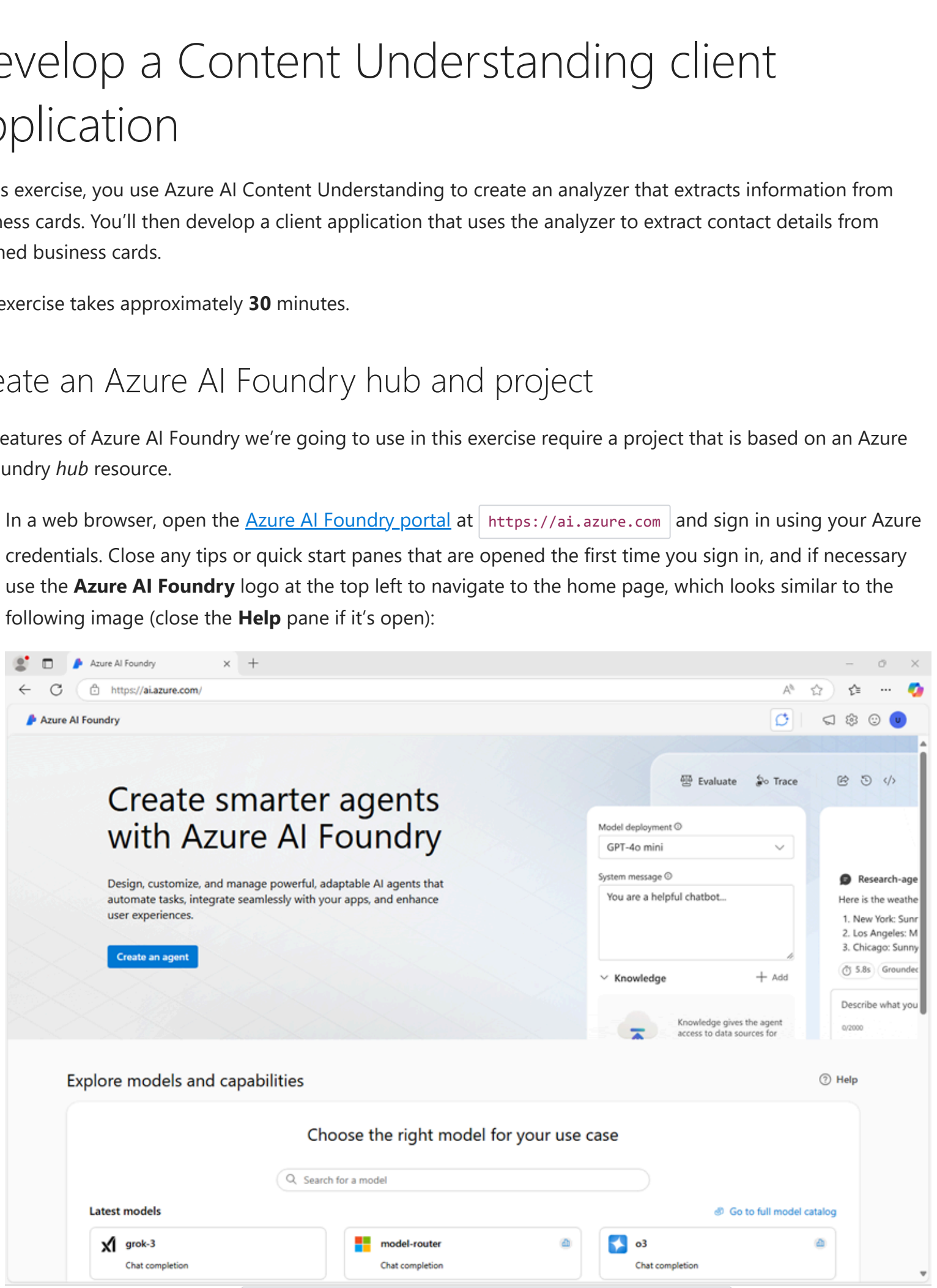
In this exercise, you use Azure AI Content Understanding to create an analyzer that extracts information from business cards. You'll then develop a client application that uses the analyzer to extract contact details from scanned business cards.

This exercise takes approximately **30** minutes.

Create an Azure AI Foundry hub and project

The features of Azure AI Foundry we're going to use in this exercise require a project that is based on an Azure AI Foundry *hub* resource.

1. In a web browser, open the Azure AI 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 **Azure AI 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):



2. In the browser, navigate to `https://ai.azure.com/managementCenter/allResources` and select **Create new**. Then choose the option to create a new **AI hub resource**.
3. In the **Create a project** wizard, enter a valid name for your project, and select the option to create a new hub. Then use the **Rename hub** link to specify a valid name for your new hub, expand **Advanced options**, and specify the following settings for your project:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Hub name**: A valid name for your hub
    - **Location**: Choose one of the following locations:*
        - Australia East
        - Sweden Central
        - West US

> *At the time of writing, Azure AI Content understanding is only available in these regions.

> **Tip**: If the **Create** button is still disabled, be sure to rename your hub to a unique alphanumeric value.

4. Wait for your project to be created, and then navigate to your project overview page.

Use the REST API to create a Content Understanding analyzer

You're going to use the REST API to create an analyzer that can extract information from images of business cards.

1. Open a new browser tab (keeping the Azure AI 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.

   > **Tip**: Resize the pane so you can work mostly in the cloud shell but still see the keys and endpoint in the Azure portal page - you'll need to copy them into your code.

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):

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

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

5. After the repo has been cloned, navigate to the folder containing the code files for your app:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/content-app
   ls -a -l
   ```

   The folder contains two scanned business card images as well as the Python code files you need to build your app.

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

   Code Copy

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

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

Code Copy

```
code .env
```

The file is opened in a code editor.

8. In the code file, replace the **YOUR_ENDPOINT** and **YOUR_KEY** placeholders with the your Azure AI services endpoint and either of its keys (copied from the Azure portal), and ensure that **ANALYZER_NAME** is set to `business-card-analyzer`.

9. After you've replaced the placeholders, within the code editor, 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.

> **Tip**: You can maximize the cloud shell pane now.

10. In the cloud shell command line, enter the following command to view the **biz-card.json** JSON file that has been provided:

Code Copy

```
cat biz-card.json
```

Scroll the cloud shell pane to view the JSON in the file, which defines an analyzer schema for a business card.

11. When you've viewed the JSON file for the analyzer, enter the following command to edit the **create-analyzer.py** Python code file that has been provided:

Code Copy

```
code create-analyzer.py
```

The Python code file is opened in a code editor.

12. Review the code, which:
    - Loads the analyzer schema from **biz-card.json** file.
    - Retrieves the endpoint, key, and analyzer name from the environment configuration file.
    - Calls a function named **create_analyzer**, which is currently not implemented

13. In the **create_analyzer** function, find the comment **Create a Content Understanding analyzer** and add the following code (being careful to maintain the correct indentation):

Code Copy

```
# Create a Content Understanding analyzer
print (f"Creating {analyzer}")

# Set the API version
CU_VERSION = "2025-05-01-preview"

# initiate the analyzer creation operation
headers = {
     "Ocp-Apim-Subscription-Key": key,
     "Content-Type": "application/json"}

url = f"{endpoint}/contentunderstanding/analyzers/{analyzer}?api-version={CU_VERSION}"

# Delete the analyzer if it already exists
response = requests.delete(url, headers=headers)
print(response.status_code)
time.sleep(1)

# Now create it
response = requests.put(url, headers=headers, data=(schema))
print(response.status_code)

# Get the response and extract the callback URL
callback_url = response.headers["Operation-Location"]

# Check the status of the operation
time.sleep(1)
result_response = requests.get(callback_url, headers=headers)

# Keep polling until the operation is no longer running
status = result_response.json().get("status")
while status == "Running":
     time.sleep(1)
     result_response = requests.get(callback_url, headers=headers)
     status = result_response.json().get("status")

result = result_response.json().get("status")
print(result)
if result == "Succeeded":
     print(f"Analyzer '{analyzer}' created successfully.")
else:
     print("Analyzer creation failed.")
     print(result_response.json())
```

14. Review the code you added, which:
    - Creates appropriate headers for the REST requests
    - Submits an HTTP *DELETE* request to delete the analyzer if it already exists.
    - Submits an HTTP *PUT* request to initiate the creation of the analyzer.
    - Checks the response to retrieve the *Operation-Location* callback URL.
    - Repeatedly submits an HTTP *GET* request to the callback URL to check the operation status until it is no longer running.
    - Confirms success (or failure) of the operation to the user.

> **Note**: The code includes some deliberate time delays to avoid exceeding the request rate limit foe the service.

15. Use the **CTRL+S** command to save the code changes, but keep the code editor pane open in case you need to correct any errors in the code. Resize the panes so you can clearly see the command line pane.

16. In the cloud shell command line pane, enter the following command to run the Python code:

```
python create-analyzer.py
```

17. Review the output from the program, which should hopefully indicate that the analyzer has been created.

Use the REST API to analyze content

Now that you've created an analyzer, you can consume it from a client application through the Content Understanding REST API.

1. In the cloud shell command line, enter the following command to edit the **read-card.py** Python code file that has been provided:

```
code read-card.py
```

   The Python code file is opened in a code editor:

2. Review the code, which:
   - Identifies the image file to be analyzed, with a default of **biz-card-1.png**.
   - Retrieves the endpoint and key for your Azure AI Services resource from the project (using the Azure credentials from the current cloud shell session to authenticate).
   - Calls a function named **analyze_card**, which is currently not implemented

3. In the **analyze_card** function, find the comment **Use Content Understanding to analyze the image** and add the following code (being careful to maintain the correct indentation):

```
# Use Content Understanding to analyze the image
print (f"Analyzing {image_file}")

# Set the API version
CU_VERSION = "2025-05-01-preview"

# Read the image data
with open(image_file, "rb") as file:
     image_data = file.read()

## Use a POST request to submit the image data to the analyzer
print("Submitting request...")
headers = {
     "Ocp-Apim-Subscription-Key": key,
     "Content-Type": "application/octet-stream"}
url = f'{endpoint}/contentunderstanding/analyzers/{analyzer}:analyze?api-version=
{CU_VERSION}'
response = requests.post(url, headers=headers, data=image_data)

# Get the response and extract the ID assigned to the analysis operation
print(response.status_code)
response_json = response.json()
id_value = response_json.get("id")

# Use a GET request to check the status of the analysis operation
print ('Getting results...')
time.sleep(1)
result_url = f'{endpoint}/contentunderstanding/analyzerResults/{id_value}?api-version=
{CU_VERSION}'
result_response = requests.get(result_url, headers=headers)
print(result_response.status_code)

# Keep polling until the analysis is complete
status = result_response.json().get("status")
while status == "Running":
     time.sleep(1)
     result_response = requests.get(result_url, headers=headers)
     status = result_response.json().get("status")

# Process the analysis results
if status == "Succeeded":
     print("Analysis succeeded:\n")
     result_json = result_response.json()
     output_file = "results.json"
     with open(output_file, "w") as json_file:
         json.dump(result_json, json_file, indent=4)
         print(f"Response saved in {output_file}\n")

     # Iterate through the fields and extract the names and type-specific values
     contents = result_json["result"]["contents"]
     for content in contents:
         if "fields" in content:
             fields = content["fields"]
             for field_name, field_data in fields.items():
                 if field_data['type'] == "string":
                     print(f"{field_name}: {field_data['valueString']}")
                 elif field_data['type'] == "number":
                     print(f"{field_name}: {field_data['valueNumber']}")
```

```
        elif field_data['type'] == "integer":
            print(f"{field_name}: {field_data['valueInteger']}")
        elif field_data['type'] == "date":
            print(f"{field_name}: {field_data['valueDate']}")
        elif field_data['type'] == "time":
            print(f"{field_name}: {field_data['valueTime']}")
        elif field_data['type'] == "array":
            print(f"{field_name}: {field_data['valueArray']}")
```

4. Review the code you added, which:
    - Reads the contents of the image file
    - Sets the version of the Content Understanding REST API to be used
    - Submits an HTTP *POST* request to your Content Understanding endpoint, instructing the to analyze the image.
    - Checks the response from the operation to retrieve an ID for the analysis operation.
    - Repeatedly submits an HTTP *GET* request to your Content Understanding endpoint to check the operation status until it is no longer running.
    - If the operation has succeeded, saves the JSON response, and then parses the JSON and displays the values retrieved for each type-specific field.

> **Note**: In our simple business card schema, all of the fields are strings. The code here illustrates the need to check the type of each field so that you can extract values of different types from a more complex schema.

5. Use the **CTRL+S** command to save the code changes, but keep the code editor pane open in case you need to correct any errors in the code. Resize the panes so you can clearly see the command line pane.

6. In the cloud shell command line pane, enter the following command to run the Python code:

Code Copy

```
python read-card.py biz-card-1.png
```

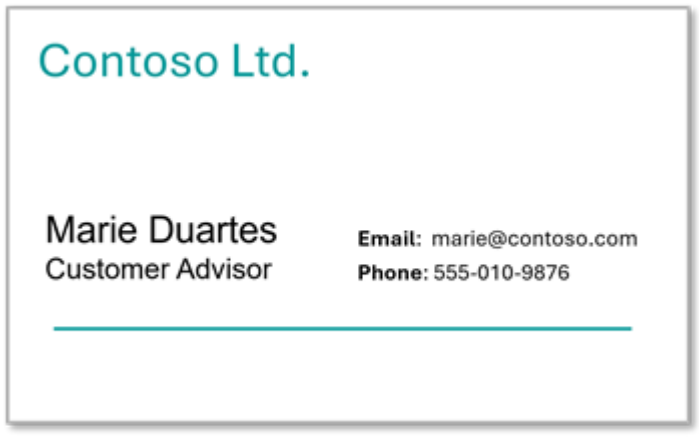
7. Review the output from the program, which should show the values for the fields in the following business card:

8. Use the following command to run the program with a different business card:

Code Copy

```
python read-card.py biz-card-2.png
```

9. Review the results, which should reflect the values in this business card:



10. In the cloud shell command line pane, use the following command to view the full JSON response that was returned:

```
cat results.json
```

Scroll to view the JSON.

Clean up

If you've finished working with the Content Understanding service, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. In the Azure portal, delete the resources you created in this exercise.

Create an Azure AI Foundry project

Use the Read model

Prepare to develop an app in Cloud Shell

Add code to use the Azure Document Intelligence service

Clean up

Analyze forms with prebuilt Azure AI Document Intelligence models

In this exercise, you'll set up an Azure AI Foundry project with all the necessary resources for document analysis. You'll use both the Azure AI Foundry portal and the Python SDK to submit forms to that resource for analysis.

While this exercise is based on Python, you can develop similar applications using multiple language-specific SDKs; including:

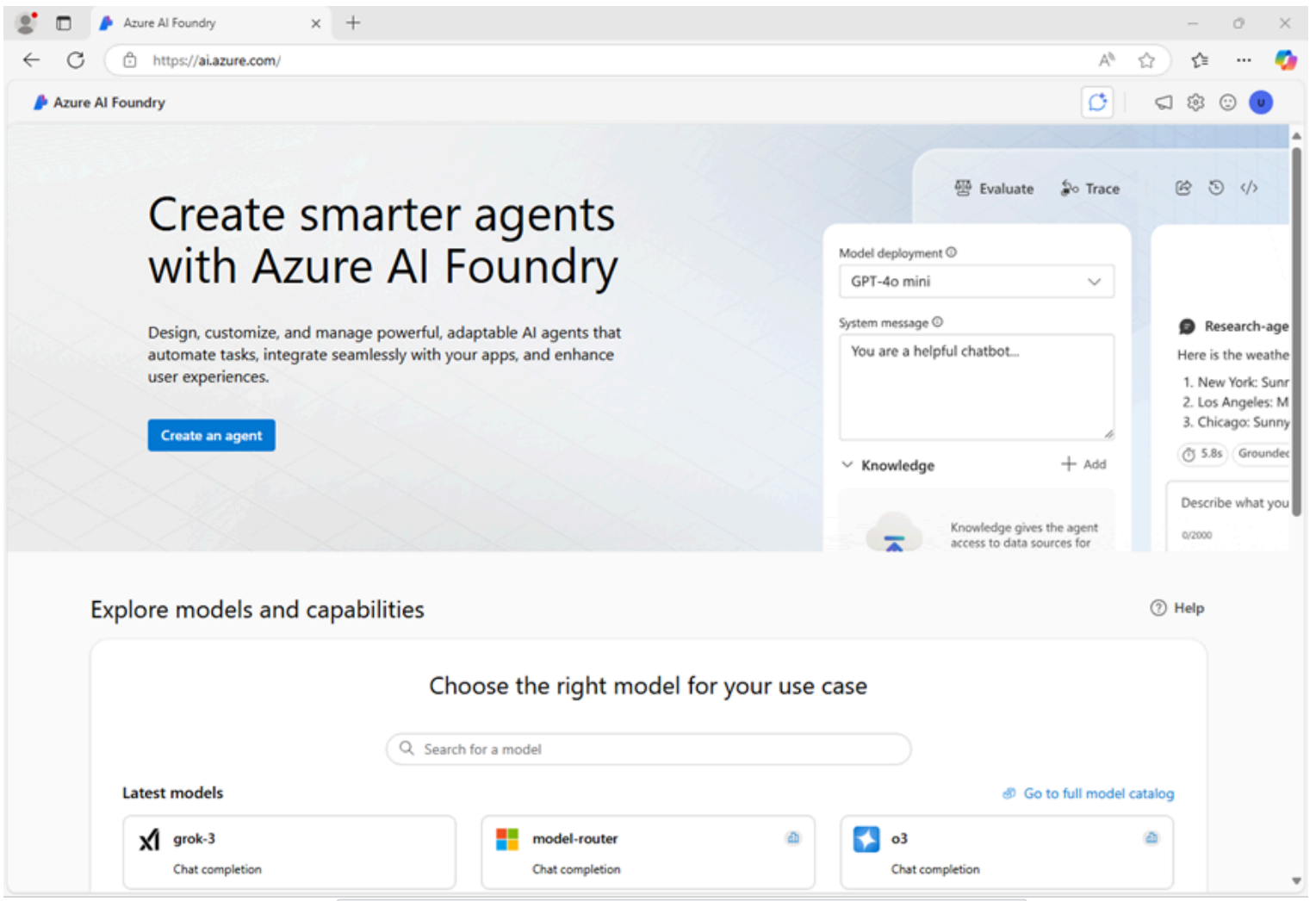
- Azure AI Document Intelligence client library for Python
- Azure AI Document Intelligence client library for Microsoft .NET
- Azure AI Document Intelligence client library for JavaScript

This exercise takes approximately **30** minutes.

Create an Azure AI Foundry project

Let's start by creating an Azure AI Foundry project.

1. In a web browser, open the Azure AI 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 **Azure AI 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):

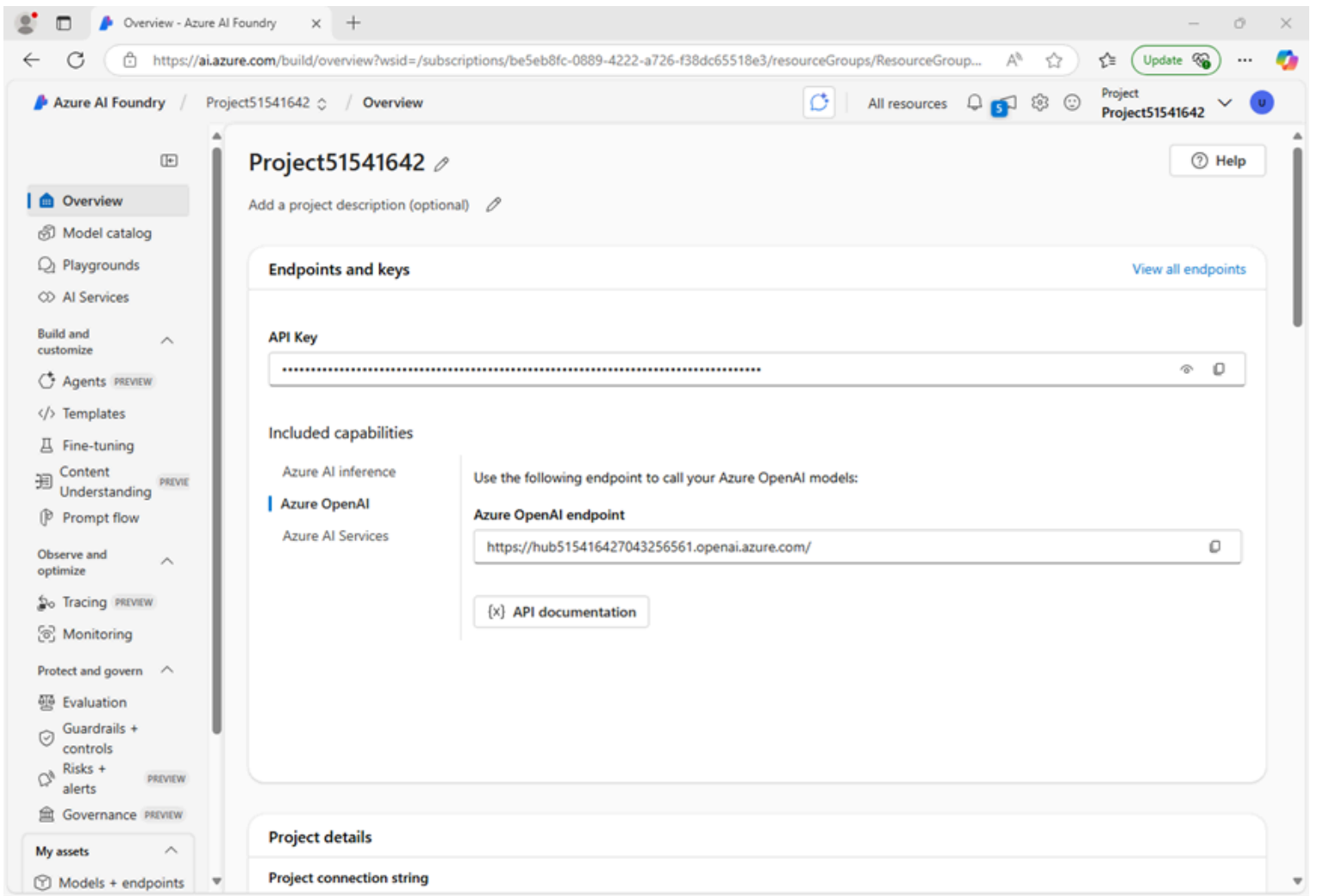


2. In the browser, navigate to `https://ai.azure.com/managementCenter/allResources` and select **Create new**. Then choose the option to create a new **AI hub resource**.
3. In the **Create a project** wizard, enter a valid name for your project, and select the option to create a new hub. Then use the **Rename hub** link to specify a valid name for your new hub, expand **Advanced options**, and specify the following settings for your project:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Any available region*

> **Note**: If you're working in an Azure subscription in which policies are used to restrict allowable resource names, you may need to use the link at the bottom of the **Create a new project** dialog box to create the hub using the Azure portal.

> **Tip**: If the **Create** button is still disabled, be sure to rename your hub to a unique alphanumeric value.

4. Wait for your project to be created.
5. When your project is created, close any tips that are displayed and review the project page in Azure AI Foundry portal, which should look similar to the following image:

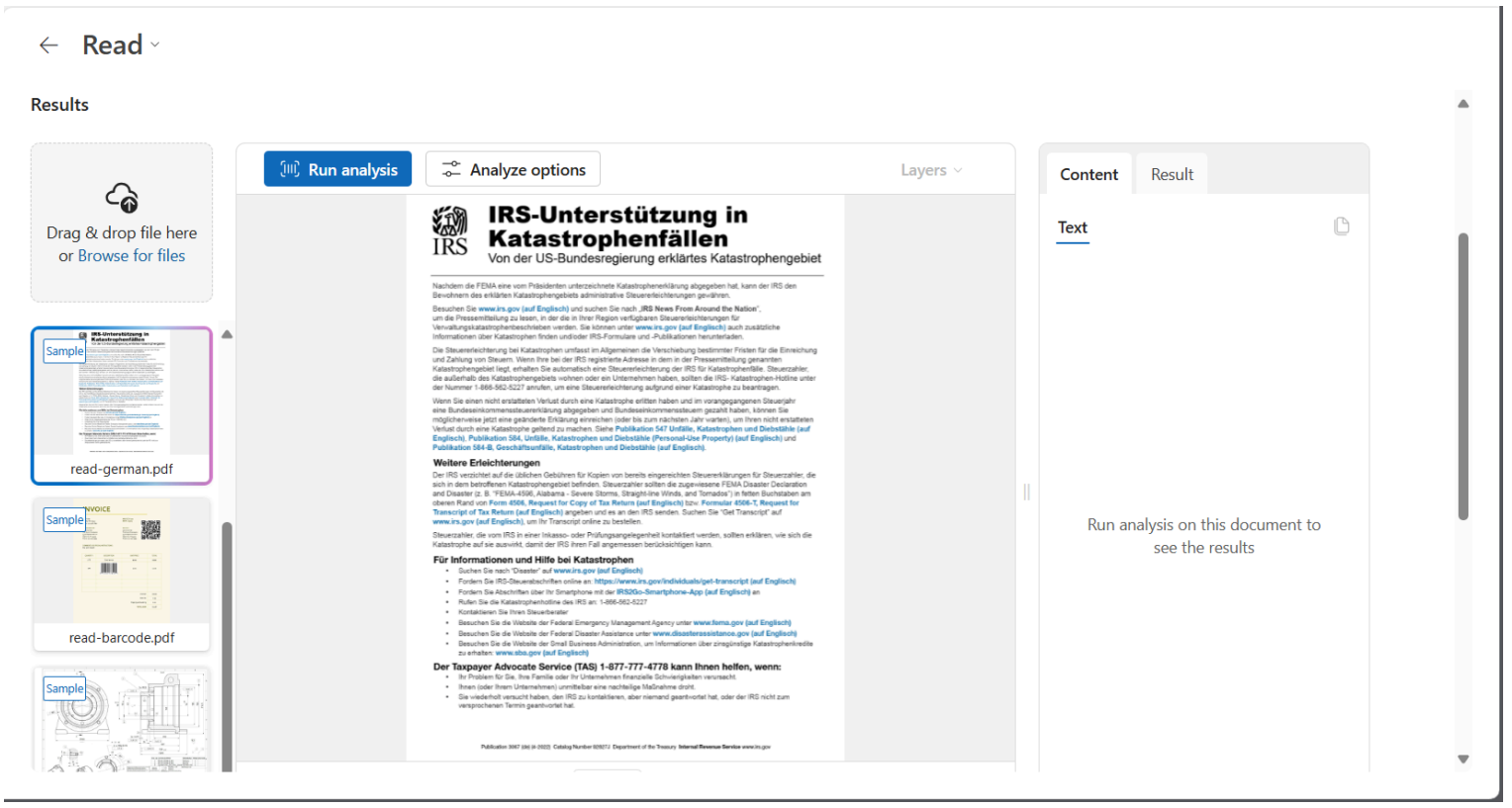


Use the Read model

Let's start by using the **Azure AI Foundry** portal and the Read model to analyze a document with multiple languages:

1. In the navigation panel on the left, select **AI Services**.
2. In the **Azure AI Services** page, select the **Vision + Document** tile.
3. In the **Vision + Document** page, verify that the **Document** tab is selected, then select the **OCR/Read** tile.

   In the **Read** page, the Azure AI Services resource created with your project should already be connected.
4. In the list of documents on the left, select **read-german.pdf**.



5. At the top toolbar, select **Analyze options**, then enable the **Language** check-box (under **Optional detection**) in the **Analyze options** pane and select **Save**.
6. At the top-left, select **Run Analysis**.
7. When the analysis is complete, the text extracted from the image is shown on the right in the **Content** tab. Review this text and compare it to the text in the original image for accuracy.
8. Select the **Result** tab. This tab displays the extracted JSON code.

Prepare to develop an app in Cloud Shell

Now let's explore the app that uses the Azure Document Intelligence service SDK. You'll develop your app using Cloud Shell. The code files for your app have been provided in a GitHub repo.

This is the invoice that your code will analyze.

CONTOSO LTD.

INVOICE

Contoso Headquarters
123 456th St
New York, NY, 10001

INVOICE: INV-100
DATE: 11/15/2019
DUE DATE: 12/15/2019
CUSTOMER NAME: MICROSOFT CORPORATION
CUSTOMER ID: CID-12345

Microsoft Corp
123 Other St,
Redmond WA, 98052

BILL TO:
Microsoft Finance
123 Bill St,
Redmond WA, 98052

SHIP TO:
Microsoft Delivery
123 Ship St,
Redmond WA, 98052

SERVICE ADDRESS:
Microsoft Services
123 Service St,
Redmond WA, 98052

SALESPERSON	P.O. NUMBER	REQUISITIONER	SHIPPED VIA	F.O.B. POINT	TERMS
	PO-3333				

QUANTITY	DESCRIPTION	UNIT PRICE	TOTAL
1	Test for 23 fields	1	$100.00
		SUBTOTAL	$100.00
		SALES TAX	$10.00
		TOTAL	$110.00

1. In the Azure AI Foundry portal, view the **Overview** page for your project.

2. In the **Endpoints and keys** area, select the **Azure AI Services** tab, and note the **API Key** and **Azure AI Services endpoint**. You'll use these credentials to connect to your Azure AI Services in a client application.
3. Open a new browser tab (keeping the Azure AI 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.
4. 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. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

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

5. 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.

6. In the PowerShell pane, enter the following commands to clone the GitHub repo for this exercise:

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

   > **Tip**: As you paste commands into the cloudshell, the ouput may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

   Now follow the steps for your chosen programming language.

7. After the repo has been cloned, navigate to the folder containing the code files:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/prebuilt-doc-intelligence/Python
   ```

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

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt azure-ai-formrecognizer==3.3.3
   ```

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

   Code Copy

   ```
   code .env
   ```

   The file is opened in a code editor.
10. In the code file, replace the **YOUR_ENDPOINT** and **YOUR_KEY** placeholders with your Azure AI services endpoint and its API key (copied from the Azure AI Foundry portal).
11. After you've replaced the placeholders, within the code editor, 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.

Add code to use the Azure Document Intelligence service

Now you're ready to use the SDK to evaluate the pdf file.

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

```
code document-analysis.py
```

   The file is opened in a code editor.

2. In the code file, find the comment **Import the required libraries** and add the following code:

```
# Add references
from azure.core.credentials import AzureKeyCredential
from azure.ai.formrecognizer import DocumentAnalysisClient
```

3. Find the comment **Create the client** and add the following code (being careful to maintain the correct indentation level):

```
# Create the client
document_analysis_client = DocumentAnalysisClient(
     endpoint=endpoint, credential=AzureKeyCredential(key)
)
```

4. Find the comment **Analyze the invoice** and add the following code:

```
# Analyse the invoice
poller = document_analysis_client.begin_analyze_document_from_url(
     fileModelId, fileUri, locale=fileLocale
)
```

5. Find the comment **Display invoice information to the user**and add the following code:

```
# Display invoice information to the user
receipts = poller.result()

for idx, receipt in enumerate(receipts.documents):

     vendor_name = receipt.fields.get("VendorName")
     if vendor_name:
         print(f"\nVendor Name: {vendor_name.value}, with confidence
{vendor_name.confidence}.")

     customer_name = receipt.fields.get("CustomerName")
     if customer_name:
         print(f"Customer Name: '{customer_name.value}, with confidence
{customer_name.confidence}.")


     invoice_total = receipt.fields.get("InvoiceTotal")
     if invoice_total:
         print(f"Invoice Total: '{invoice_total.value.symbol}{invoice_total.value.amount},
 with confidence {invoice_total.confidence}.")
```

6. In the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes. Keep the code editor open in case you need to fix any errors in the code, but resize the panes so you can see the command line pane clearly.

7. In the command line pane, enter the following command to run the application.

Code Copy

```
python document-analysis.py
```

The program displays the vendor name, customer name, and invoice total with confidence levels. Compare the values it reports with the sample invoice you opened at the start of this section.

Clean up

If you’re done with your Azure resource, remember to delete the resource in the Azure portal (`https://portal.azure.com`) to avoid further charges.

Create a Azure AI Document Intelligence resource

Prepare to develop an app in Cloud Shell

Gather documents for training

Train the model using Document Intelligence Studio

Test your custom Document Intelligence model

Clean up

More information

Analyze forms with custom Azure AI Document Intelligence models

Suppose a company currently requires employees to manually purchase order sheets and enter the data into a database. They would like you to utilize AI services to improve the data entry process. You decide to build a machine learning model that will read the form and produce structured data that can be used to automatically update a database.

Azure AI Document Intelligence is an Azure AI service that enables users to build automated data processing software. This software can extract text, key/value pairs, and tables from form documents using optical character recognition (OCR). Azure AI Document Intelligence has pre-built models for recognizing invoices, receipts, and business cards. The service also provides the capability to train custom models. In this exercise, we will focus on building custom models.

While this exercise is based on Python, you can develop similar applications using multiple language-specific SDKs; including:

- Azure AI Document Intelligence client library for Python
- Azure AI Document Intelligence client library for Microsoft .NET
- Azure AI Document Intelligence client library for JavaScript

This exercise takes approximately **30** minutes.

Create a Azure AI Document Intelligence resource

To use the Azure AI Document Intelligence service, you need a Azure AI Document Intelligence or Azure AI Services resource in your Azure subscription. You'll use the Azure portal to create a resource.

1. In a browser tab, open the Azure portal at `https://portal.azure.com`, signing in with the Microsoft account associated with your Azure subscription.
2. On the Azure portal home page, navigate to the top search box and type **Document Intelligence** and then press **Enter**.
3. On the **Document Intelligence** page, select **Create**.
4. On the **Create Document Intelligence** page, create a new resource with the following settings:
    - **Subscription**: Your Azure subscription.
    - **Resource group**: Create or select a resource group
    - **Region**: Any available region
    - **Name**: A valid name for your Document Intelligence resource
    - **Pricing tier**: Free F0 (*if you don't have a Free tier available, select* Standard S0).
5. When the deployment is complete, select **Go to resource** to view the resource's **Overview** page.

Prepare to develop an app in Cloud Shell

You'll develop your text translation app using Cloud Shell. The code files for your app have been provided in a GitHub repo.

1. In the Azure Portal, 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. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

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

2. Size the cloud shell pane so you can see both the command line console and the Azure portal. You'll need to use the the split bar to switch as you switch between the two panes.

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 PowerShell pane, enter the following commands to clone the GitHub repo for this exercise:

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

   > **Tip**: As you paste commands into the cloudshell, the ouput may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

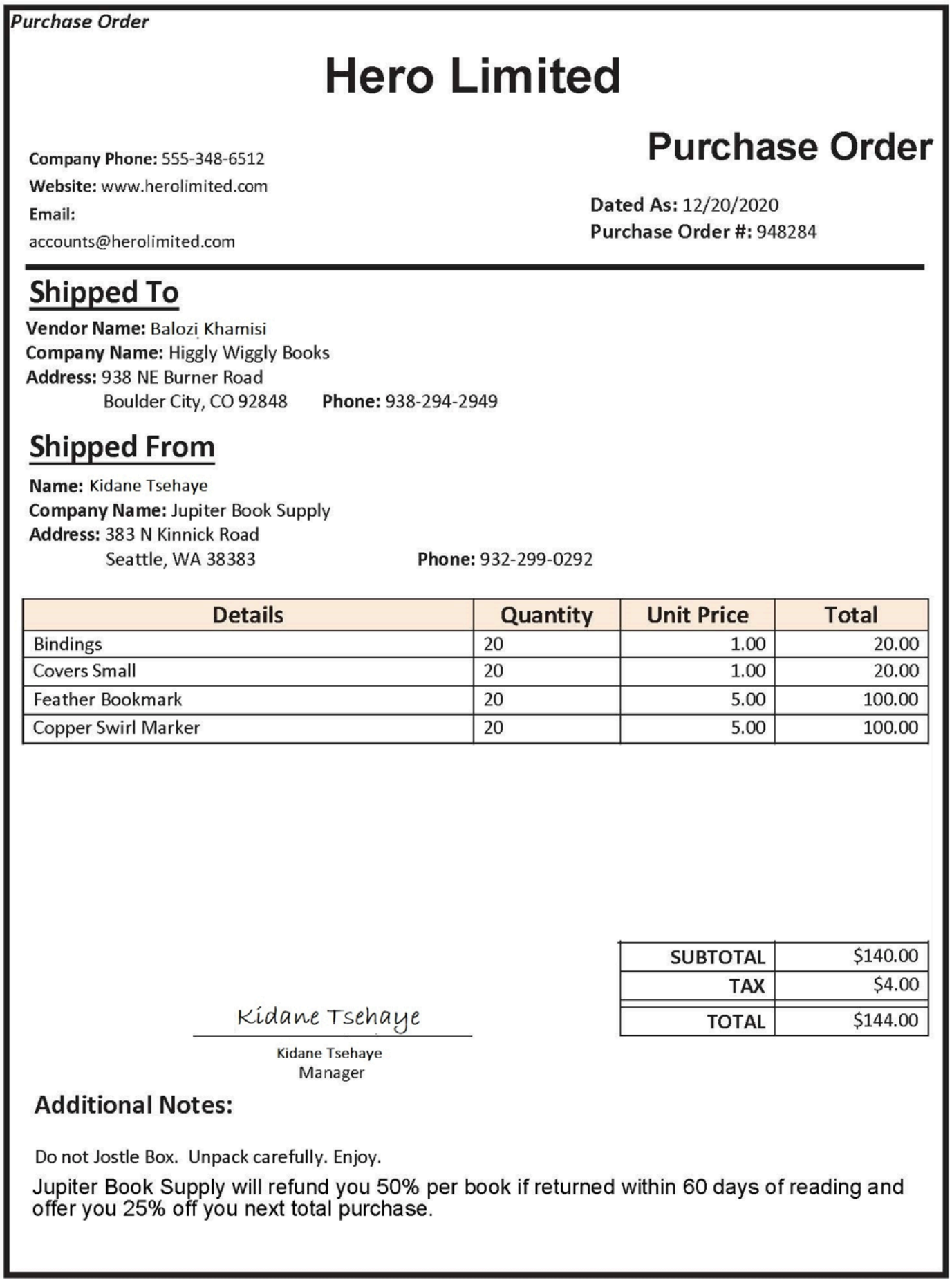
5. After the repo has been cloned, navigate to the folder containing the application code files:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/custom-doc-intelligence
   ```

Gather documents for training

You'll use the sample forms such as this one to train a test a model:



Purchase Order

Hero Limited

Company Phone: 555-348-6512
Website: www.herolimited.com
Email:
accounts@herolimited.com

Purchase Order

Dated As: 12/20/2020
Purchase Order #: 948284

Shipped To

Vendor Name: Balozi Khamisi
Company Name: Higgly Wiggly Books
Address: 938 NE Burner Road
Boulder City, CO 92848 **Phone:** 938-294-2949

Shipped From

Name: Kidane Tsehaye
Company Name: Jupiter Book Supply
Address: 383 N Kinnick Road
Seattle, WA 38383 **Phone:** 932-299-0292

Details	Quantity	Unit Price	Total
Bindings	20	1.00	20.00
Covers Small	20	1.00	20.00
Feather Bookmark	20	5.00	100.00
Copper Swirl Marker	20	5.00	100.00

SUBTOTAL	$140.00
TAX	$4.00
TOTAL	$144.00

Kidane Tsehaye

Kidane Tsehaye
Manager

Additional Notes:

Do not Jostle Box. Unpack carefully. Enjoy.

Jupiter Book Supply will refund you 50% per book if returned within 60 days of reading and offer you 25% off you next total purchase.

1. In the command line, run `ls ./sample-forms` to list the content in the **sample-forms** folder. Notice there are files ending in **.json** and **.jpg** in the folder.

   You will use the **.jpg** files to train your model.

   The **.json** files have been generated for you and contain label information. The files will be uploaded into your blob storage container alongside the forms.
2. In the **Azure portal** and navigate to your resource's **Overview** page if you're not already there. Under the *Essentials* section, note the **Resource group**, **Subscription ID**, and **Location**. You will need these values in subsequent steps.

3. Run the command `code setup.sh` to open **setup.sh** in a code editor. You will use this script to run the Azure command line interface (CLI) commands required to create the other Azure resources you need.
4. In the **setup.sh** script, review the commands. The program will:

   - Create a storage account in your Azure resource group
   - Upload files from your local *sampleforms* folder to a container called *sampleforms* in the storage account
   - Print a Shared Access Signature URI

5. Modify the **subscription_id**, **resource_group**, and **location** variable declarations with the appropriate values for the subscription, resource group, and location name where you deployed the Document Intelligence resource.

   > **Important**: For your **location** string, be sure to use the code version of your location. For example, if your location is "East US", the string in your script should be `eastus`. You can see that version is the **JSON View** button on the right side of the **Essentials** tab of your resource group in Azure portal.

   If the **expiry_date** variable is in the past, update it to a future date. This variable is used when generating the Shared Access Signature (SAS) URI. In practice, you will want to set an appropriate expiry date for your SAS. You can learn more about SAS here.

6. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes and then use the **CTRL+Q** command or **Right-click > Quit** to close the code editor while keeping the cloud shell command line open.

7. Enter the following commands to make the script executable and to run it:

   Code Copy

   ```
   chmod +x ./setup.sh
   ./setup.sh
   ```

8. When the script completes, review the displayed output.
9. In the Azure portal, refresh your resource group and verify that it contains the Azure Storage account just created. Open the storage account and in the pane on the left, select **Storage browser**. Then in Storage Browser, expand **Blob containers** and select the **sampleforms** container to verify that the files have been uploaded from your local **custom-doc-intelligence/sample-forms** folder.

Train the model using Document Intelligence Studio

Now you will train the model using the files uploaded to the storage account.

1. Open a new browser tab, and navigate to the Document Intelligence Studio at `https://documentintelligence.ai.azure.com/studio`.
2. Scroll down to the **Custom models** section and select the **Custom extraction model** tile.
3. If prompted, sign in with your Azure credentials.
4. If you are asked which Azure AI Document Intelligence resource to use, select the subscription and resource name you used when you created the Azure AI Document Intelligence resource.
5. Under **My Projects**, Create a new project with the following configuration:
   - **Enter project details**:
     - **Project name**: A valid name for your project
   - **Configure service resource**:
     - **Subscription**: Your Azure subscription
     - **Resource group**: The resource group where you deployed your Document Intelligence resource
     - **Document intelligence resource** Your Document Intelligence resource (select the *Set as default* option and use the default API version)
   - **Connect training data source**:
     - **Subscription**: Your Azure subscription
     - **Resource group**: The resource group where you deployed your Document Intelligence resource
     - **Storage account**: The storage account that was created by the setup script (select the *Set as default* option, select the `sampleforms` blob container, and leave the folder path blank)
6. When your project is created, on the top right of the page, select **Train** to train your model. Use the following configurations:
   - **Model ID**: A valid name for your model (*you'll need the model ID name in the next step*)

- **Build Mode**: Template.

7. Select **Go to Models**.
8. Training can take some time. Wait until the status is **succeeded**.

Test your custom Document Intelligence model

1. Return to the browser tab containing the Azure Portal and cloud shell. In the command line, run the following command to change to the folder containing the application code files:

```
cd Python
```

2. Install the Document Intelligence package by running the following command:

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-ai-formrecognizer==3.3.3
```

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

```
code .env
```

4. In the pane containing the Azure portal, on the **Overview** page for your Document Intelligence resource, select **Click here to manage keys** to see the endpoint and keys for your resource. Then edit the configuration file with the following values:
   - Your Document Intelligence endpoint
   - Your Document Intelligence key
   - The Model ID you specified when training your model
5. After you've replaced the placeholders, within the code editor, 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.
6. Open the code file for your client application (`code Program.cs` for C#, `code test-model.py` for Python) and review the code it contains, particularly that the image in the URL refers to the file in this GitHub repo on the web. Close the file without making any changes.
7. In the command line, and enter the following command to run the program:

```
python test-model.py
```

8. View the output and observe how the output for the model provides field names like `Merchant` and `CompanyPhoneNumber` .

Clean up

If you're done with your Azure resource, remember to delete the resource in the Azure portal to avoid further charges.

More information

For more information about the Document Intelligence service, see the Document Intelligence documentation.

Create Azure resources

Upload documents to Azure Storage

Create and run an indexer

Search the index

Create a search client application

View the knowledge store

Clean-up

More information

Create an knowledge mining solution

In this exercise, you use AI Search to index a set of documents maintained by Margie's Travel, a fictional travel agency. The indexing process involves using AI skills to extract key information to make them searchable, and generating a knowledge store containing data assets for further analysis.

While this exercise is based on Python, you can develop similar applications using multiple language-specific SDKs; including:

- Azure AI Search client library for Python
- Azure AI Search client library for Microsoft .NET
- Azure AI Search client library for JavaScript

This exercise takes approximately **40** minutes.

Create Azure resources

The solution you will create for Margie's Travel requires multiple resources in your Azure subscription. In this exercise, you'll create them directly in the Azure portal. You could also create them by using a script, or an ARM or BICEP template; or you could create an Azure AI Foundry project that includes an Azure AI Search resource.

> **Important**: Your Azure resources should be created in the same location!

Create an Azure AI Search resource

1. In a web browser, open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials.
2. Select the **+ Create a resource** button, search for `Azure AI Search`, and create an **Azure AI Search** resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Service name**: *A valid name for your search resource*
    - **Location**: *Any available location*
    - **Pricing tier**: Free
3. Wait for deployment to complete, and then go to the deployed resource.
4. Review the **Overview** page on the blade for your Azure AI Search resource in the Azure portal. Here, you can use a visual interface to create, test, manage, and monitor the various components of a search solution; including data sources, indexes, indexers, and skillsets.

Create a storage account

1. Return to the home page, and then create a **Storage account** resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *The same resource group as your Azure AI Search and Azure AI Services resources*
    - **Storage account name**: *A valid name for your storage resource*
    - **Region**: *The same region as your Azure AI Search resource*
    - **Primary service**: Azure Blob Storage or Azure Data Lake Storage Gen 2
    - **Performance**: Standard
    - **Redundancy**: Locally-redundant storage (LRS)
2. Wait for deployment to complete, and then go to the deployed resource.

> **Tip**: Keep the storage account portal page open - you will use it in the next procedure.

Upload documents to Azure Storage

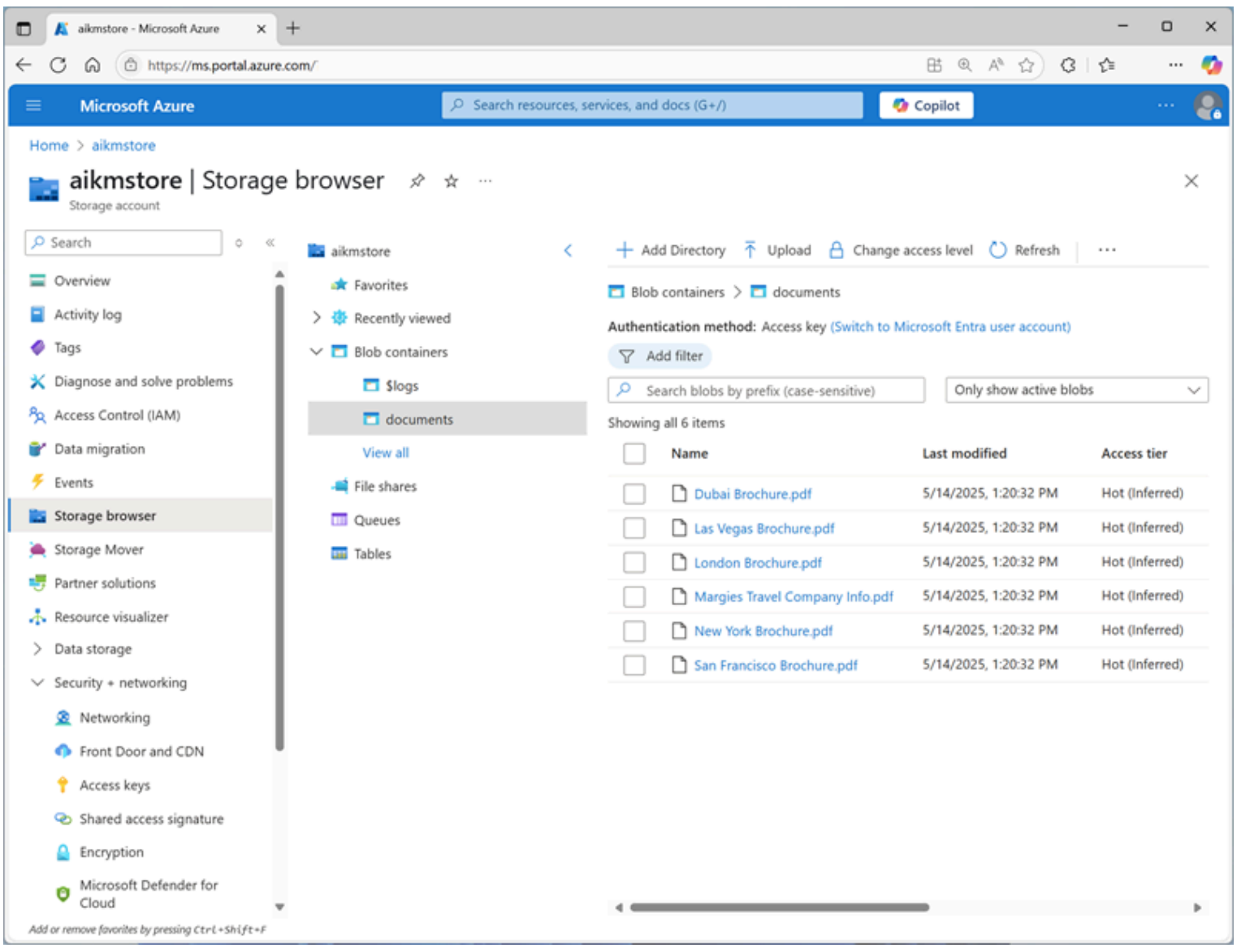
Your knowledge mining solution will extract information from travel brochure documents in an Azure Storage blob container.

1. In a new browser tab, download documents.zip from
   ```
   https://github.com/microsoftlearning/mslearn-ai-information-extraction/raw/main/Labfiles/knowledge/documents.zip
   ```
   and save it in a local folder.
2. Extract the downloaded *documents.zip* file and view the travel brochure files it contains. You'll extract and index information from these files.
3. In the browser tab containing the Azure portal page for your storage account, in the navigation pane on the left, select **Storage browser**.
4. In the storage browser, select **Blob containers**.

   Currently, your storage account should contain only the default **$logs** container.
5. In the toolbar, select **+ Container** and create a new container with the following settings:
   - **Name**: `documents`
   - **Anonymous access level**: Private (no anonymous access)*

> **Note**: *Unless you enabled the option to allow anonymous container access when creating your storage account, you won't be able to select any other setting!

6. Select the **documents** container to open it, and then use the **Upload** toolbar button to upload the .pdf files you extracted from **documents.zip** previously into the root of the container, as shown here:



Create and run an indexer

Now that you have the documents in place, you can create an indexer to extract information from them.

1. In the Azure portal, browse to your Azure AI Search resource. Then, on its **Overview** page, select **Import data**.
2. On the **Connect to your data** page, in the **Data Source** list, select **Azure Blob Storage**. Then complete the data store details with the following values:

- **Data Source**: Azure Blob Storage
- **Data source name**: `margies-documents`
- **Data to extract**: Content and metadata
- **Parsing mode**: Default
- **Subscription**: *Your Azure subscription*
- **Connection string**:
    - Select **Choose an existing connection**
    - Select your storage account
    - Select the **documents** container
- **Managed identity authentication**: None
- **Container name**: documents
- **Blob folder**: *Leave this blank*
- **Description**: `Travel brochures`

3. Proceed to the next step (**Add cognitive skills**), which has three expandable sections to complete.

4. In the **Attach Azure AI Services** section, select **Free (limited enrichments)***.

> **Note**: *The free Azure AI Services resource for Azure AI Search can be used to index a maximum of 20 documents. In a real solution, you should create an Azure AI Services resource in your subscription to enable AI enrichment for a larger number of documents.

5. In the **Add enrichments** section:

    - Change the **Skillset name** to `margies-skillset`.
    - Select the option **Enable OCR and merge all text into merged_content field**.
    - Ensure that the **Source data field** is set to **merged_content**.
    - Leave the **Enrichment granularity level** as **Source field**, which is set the entire contents of the document being indexed; but note that you can change this to extract information at more granular levels, like pages or sentences.
    - Select the following enriched fields:

Cognitive Skill	Parameter	Field name
Text Cognitive Skills		
Extract people names		people
Extract location names		locations
Extract key phrases		keyphrases
Image Cognitive Skills		
Generate tags from images		imageTags
Generate captions from images		imageCaption

    Double-check your selections (it can be difficult to change them later).

6. In the **Save enrichments to a knowledge store** section:

    - Select only the following checkboxes (an error will be displayed, you'll resolve that shortly):
        - **Azure file projections**:
            - Image projections
        - **Azure table projections**:
            - Documents
                - Key phrases
        - **Azure blob projections**:

        - Document
    - Under **Storage account connection string** (beneath the error messages):
        - Select **Choose an existing connection**
        - Select your storage account
        - Select the **documents** container (*this is only required to select the storage account in the browse interface - you'll specify a different container name for the extracted knowledge assets!*)
    - Change the **Container name** to `knowledge-store`.
7. Proceed to the next step (**Customize target index**), where you'll specify the fields for your index.
8. Change the **Index name** to `margies-index`.
9. Ensure that the **Key** is set to **metadata_storage_path**, leave the **Suggester name** blank, and ensure **Search mode** is **analyzingInfixMatching**.
10. Make the following changes to the index fields, leaving all other fields with their default settings (**IMPORTANT**: you may need to scroll to the right to see the entire table):

Field name	Retrievable	Filterable	Sortable	Facetable	Searchable
metadata_storage_size	✓	✓	✓		
metadata_storage_last_modified	✓	✓	✓		
metadata_storage_name	✓	✓	✓		✓
locations	✓	✓			✓
people	✓	✓			✓
keyphrases	✓	✓			✓

    Double-check your selections, paying particular attention to ensure that the correct **Retrievable**, **Filterable**, **Sortable**, **Facetable**, and **Searchable** options are selected correctly for each field (it can be difficult to change them later).
11. Proceed to the next step (**Create an indexer**), where you'll create and schedule the indexer.
12. Change the **Indexer name** to `margies-indexer`.
13. Leave the **Schedule** set to **Once**.
14. Select **Submit** to create the data source, skillset, index, and indexer. The indexer is run automatically and runs the indexing pipeline, which:
    - Extracts the document metadata fields and content from the data source
    - Runs the skillset of cognitive skills to generate additional enriched fields
    - Maps the extracted fields to the index.
    - Saves the extracted data assets to the knowledge store.
15. In the navigation pane on the left, under **Search management** view the **Indexers** page, which should show the newly created **margies-indexer**. Wait a few minutes, and click **↻ Refresh** until the **Status** indicates **Success**.

Search the index

Now that you have an index, you can search it.

1. Return to the **Overview** page for your Azure AI Search resource, and on the toolbar, select **Search explorer**.
2. In Search explorer, in the **Query string** box, enter `*` (a single asterisk), and then select **Search**.

    This query retrieves all documents in the index in JSON format. Examine the results and note the fields for each document, which contain document content, metadata, and enriched data extracted by the cognitive skills you selected.
3. In the **View** menu, select **JSON view** and note that the JSON request for the search is shown, like this:

Code Copy

```
{
  "search": "*",
  "count": true
}
```

4. The results include a **@odata.count** field at the top of the results that indicates the number of documents returned by the search.

5. Modify the JSON request to include the **select** parameter as shown here:

Code Copy

```
{
  "search": "*",
  "count": true,
  "select": "metadata_storage_name,locations"
}
```

This time the results include only the file name and any locations mentioned in the document content. The file name is in the **metadata_storage_name** field, which was extracted from the source document. The **locations** field was generated by an AI skill.

6. Now try the following query string:

Code Copy

```
{
  "search": "New York",
  "count": true,
  "select": "metadata_storage_name,keyphrases"
}
```

This search finds documents that mention “New York” in any of the searchable fields, and returns the file name and key phrases in the document.

7. Let’s try one more query:

Code Copy

```
{
    "search": "New York",
    "count": true,
    "select": "metadata_storage_name,keyphrases",
    "filter": "metadata_storage_size lt 380000"
}
```

This query returns the filename and key phrases for any documents mentioning “New York” that are smaller than 380,000 bytes in size.

Create a search client application

Now that you have a useful index, you can use it from a client application. You can do this by consuming the REST interface, submitting requests and receiving responses in JSON format over HTTP; or you can use the software development kit (SDK) for your preferred programming language. In this exercise, we’ll use the SDK.

> **Note**: You can choose to use the SDK for either **C#** or **Python**. In the steps below, perform the actions appropriate for your preferred language.

Get the endpoint and keys for your search resource

1. In the Azure portal, close the search explorer page and return to the **Overview** page for your Azure AI Search resource.

   Note the **Url** value, which should be similar to **https://*your_resource_name*.search.windows.net**. This is the endpoint for your search resource.

2. In the navigation pane on the left, expand **Settings** and view the **Keys** page.

   Note that there are two **admin** keys, and a single **query** key. An *admin* key is used to create and manage search resources; a *query* key is used by client applications that only need to perform search queries.

   *You will need the endpoint and **query** key for your client application.*

Prepare to use the Azure AI Search SDK

1. Use the **[>_]** button to the right of the search bar at the top of the Azure portal 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. Initially, you'll need to see both the cloud shell and the Azure portal (so you can find and copy the endpoint and key you'll need).

2. 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.

3. 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):

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

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

4. After the repo has been cloned, navigate to the folder containing the application code files:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/knowledge/python
   ls -a -l
   ```

5. Install the Azure AI Search SDK and Azure identity packages by running the following commands:

   Code Copy

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-identity azure-search-documents==11.5.1
```

6. Run the following command to edit the configuration file for your app:

Code Copy

```
code .env
```

The configuation file is opened in a code editor.

7. Edit the configuration file to replace the following placeholder values:
    - **your_search_endpoint** (*replace with the endpoint for your Azure AI Search resource*)
    - **your_query_key** *(replace with the query key for your Azure AI Search resource*)
    - **your_index_name** (*replace with the name of your index, which should be* `margies-index`)
8. When you've updated the placeholders, use the **CTRL+S** command to save the file and then use the **CTRL+Q** command to close it.

> **Tip**: Now that you've copied the endpoint and key from the Azure portal, you might want to maximize the cloud shell pane to make it easier to work in.

9. Run the following command to open the code file for your app:

Code Copy

```
code search-app.py
```

The code file is opened in a code editor.

10. Review the code, and note that it performs the following actions:
    - Retrieves the configuration settings for your Azure AI Search resource and index from the configuration file you edited.
    - Creates a **SearchClient** with the endpoint, key, and index name to connect to your search service.
    - Prompts the user for a search query (until they enter "quit")
    - Searches the index using the query, returning the following fields (ordered by metadata_storage_name):
        - metadata_storage_name
        - locations
        - people
        - keyphrases
    - Parses the search results that are returned to display the fields returned for each document in the result set.
11. Close the code editor pane (*CTRL+Q*), keeping the cloud shell command line console pane open
12. Enter the following command to run the app:

Code Copy

```
python search-app.py
```

13. When prompted, enter a query such as `London` and view the results.
14. Try another query, such as `flights` .
15. When you're finished testing the app, enter `quit` to close it.

16. Close the Cloud shell, returning to the Azure portal.

View the knowledge store

After you have run an indexer that uses a skillset to create a knowledge store, the enriched data extracted by the indexing process is persisted in the knowledge store projections.

View object projections

The *object* projections defined in the Margie's Travel skillset consist of a JSON file for each indexed document. These files are stored in a blob container in the Azure Storage account specified in the skillset definition.

1. In the Azure portal, view the Azure Storage account you created previously.
2. Select the **Storage browser** tab (in the pane on the left) to view the storage account in the storage explorer interface in the Azure portal.
3. Expand **Blob containers** to view the containers in the storage account. In addition to the **documents** container where the source data is stored, there should be two new containers: **knowledge-store** and **margies-skillset-image-projection**. These were created by the indexing process.
4. Select the **knowledge-store** container. It should contain a folder for each indexed document.
5. Open any of the folders, and then select the **objectprojection.json** file it contains and use the **Download** button on the toolbar to download and open it. Each JSON file contains a representation of an indexed document, including the enriched data extracted by the skillset as shown here (formatted to make it easier to read).

Code Copy

```
{
    "metadata_storage_content_type": "application/pdf",
    "metadata_storage_size": 388622,
    "<more_metadata_fields>": "...",
    "key_phrases":[
        "Margie’s Travel",
        "Margie's Travel",
        "best travel experts",
        "world-leading travel agency",
        "international reach"
        ],
    "locations":[
        "Dubai",
        "Las Vegas",
        "London",
        "New York",
        "San Francisco"
        ],
    "image_tags":[
        "outdoor",
        "tree",
        "plant",
        "palm"
        ],
    "more fields": "..."
}
```

The ability to create *object* projections like this enables you to generate enriched data objects that can be incorporated into an enterprise data analysis solution.

View file projections

The *file* projections defined in the skillset create JPEG files for each image that was extracted from the documents during the indexing process.

1. In the *Storage browser* interface in the Azure portal, select the **margies-skillset-image-projection** blob container. This container contains a folder for each document that contained images.
2. Open any of the folders and view its contents - each folder contains at least one *.jpg file.
3. Open any of the image files, and download and view it to see the image.

The ability to generate *file* projections like this makes indexing an efficient way to extract embedded images from a large volume of documents.

View table projections

The *table* projections defined in the skillset form a relational schema of enriched data.

1. In the *Storage browser* interface in the Azure portal, expand **Tables**.
2. Select the **margiesSkillsetDocument** table to view data. This table contains a row for each document that was indexed:
3. View the **margiesSkillsetKeyPhrases** table, which contains a row for each key phrase extracted from the documents.

The ability to create *table* projections enables you to build analytical and reporting solutions that query a relational schema. The automatically generated key columns can be used to join the tables in queries - for example to return all of the key phrases extracted from a specific document.

Clean-up

Now that you've completed the exercise, delete all the resources you no longer need. Delete the Azure resources:

1. In the Azure portal, select **Resource groups**.
2. Select the resource group you don't need, then select **Delete resource group**.

More information

To learn more about Azure AI Search, see the Azure AI Search documentation.