

# Identifying and extracting text with Document Intelligence in Microsoft Fabric

## Introduction

The analysis of structured data has been well-established. However, unstructured data – including text, images, and videos – has historically posed significant analytical hurdles. The development of advanced AI models, particularly OpenAI's GPT-3 and GPT-4, is revolutionizing this landscape, enabling easier analysis and the extraction of valuable insights from unstructured data. A key illustration of this progress is the ability to query documents in natural language, a capability enabled by the synergy of information retrieval and language generation.

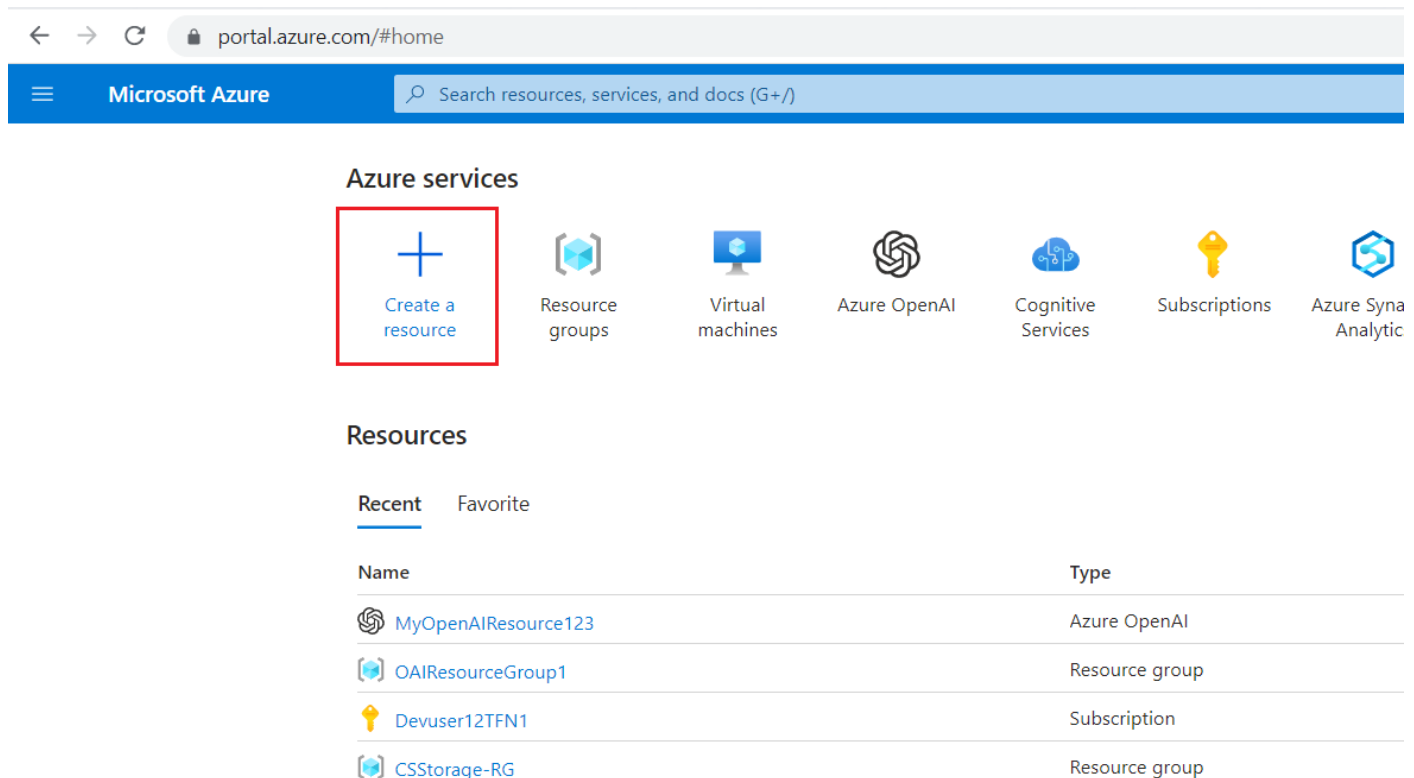
## Objective

- Pre-process PDF Documents using Azure AI Document Intelligence in Azure AI Services.
- Perform text chunking using SynapseML.
- Generate embeddings for the chunks using SynapseML and Azure OpenAI Services.
- Store the embeddings in Azure AI Search.
- Build a question-answering pipeline.

## Exercise 1: Setup

### Task 1: Create an Azure AI Search service in the portal

1. In Azure portal home page, click on **+ Create Resource**.



Azure services

Create a resource

Resource groups

Virtual machines

Azure OpenAI

Cognitive Services

Subscriptions

Azure Synapse Analytics

Resources

Recent Favorite

Name	Type
MyOpenAIResource123	Azure OpenAI
OAIResourceGroup1	Resource group
Devuser12TFN1	Subscription
CSStorage-RG	Resource group

2. In the **Create a resource** page search bar, type **Azure AI Search** and click on the appeared **azure ai search**.

Home >

## Create a resource

Get Started

Recently created

Categories

AI + Machine Learning

Analytics

Blockchain

Compute

Containers

Databases

Developer Tools

...

Azure AI Search

azure ai search

Virtual machine

Create | Docs | MS Learn

Web App

Create | Docs | MS Learn

SQL Database

Create | Docs | MS Learn

Function App

Create | Docs

Getting Started? Try our Quickstart cer

Popular Marke

Windows

Create

Windows

Create

Ubuntu

Create

Ubuntu

Create

3. Click on **azure ai search** section.

Home > Create a resource >

## Marketplace

Get Started

Service Providers

Management

Private Marketplace

Private Offer Management

My Marketplace

Favorites

Recently created

Private plans

Categories

AI + Machine Learning (23)

Analytics (6)

azure cognitive search

Pricing : All

Operating System : All

Publisher Type : All

Product Type : All

Publisher name : All

☐ Azure services only

Showing 1 to 20 of 33 results for 'azure cognitive search'. [Clear search](#)

BA Insight

Free trial

BA Insight for Azure Cognitive Search

BA-Insight - Global HQ (Boston)

SaaS

Create AI-Driven Connected & Personalized Search Experiences for Employees, Customers & Web Visitors

Starts at Free

Subscribe

Azure Cognitive Search solution for Sentinel

Microsoft Sentinel, Microsoft Co...

Azure Application

Azure Cognitive Search solution for Sentinel

Price varies

Create

Azure AI Search

Microsoft

Azure Service

AI-powered cloud search service for mobile and web app development (formerly Azure Search)

Create

4. In the **Azure AI Search** page, click on the **Create** button.

Home > Create a resource > Marketplace >

# Azure AI Search

Microsoft



## Azure AI Search [Add to Favorites](#)

Microsoft | Azure Service

★ 3.8 (185 ratings)

Plan

Azure AI Search

Create

Overview Plans Usage Information + Support Ratings + Reviews

5. On the **Create a search service** page, provide the following information and click on **Review+create** button.

Field	Description
Subscription	Select your Azure OpenAI subscription
Resource group	Select your Resource group(that you have created in <b>Lab 1</b> )
Region	EastUS 2
Name	mysearchserviceXX (XXcan be unique number)
Pricing Tier	Click on change Price Tire>select <b>Basic</b>

# Create a search service ...

Basics Scale Networking Tags Review + create

## Project details

Subscription \* Azure Pass - Sponsorship 1 ▼

Resource Group \* AI-Padme-001 2 ▼

[Create new](#)

## Instance Details

Service name \* ⓘ mysearchservice859 3 ✓

Location \* East US 2 4 ▼

Pricing tier \* ⓘ

**Standard2**

512 GB/Partition, max 12 replicas, max 12 partitions, max 36 search units

Change Pricing Tier 5

6.

## Select Pricing Tier

Browse available skus and their features

SKU	Offering	Indexes	Indexers	Storage	Search units	Replicas	Partitions
F	Free	3	3	50 MB	1	1	1
<span style="border: 1px solid red; padding: 2px;">B</span>	<span style="border: 1px solid red; padding: 2px;">Basic</span>	15	15	2 GB	3	3	1
S	Standard	50	50	25 GB/Partition*	36	12	12
S2	Standard	200	200	100 GB/Partition*	36	12	12
S3	Standard	200	200	200 GB/Partition*	36	12	12
S3HD	High-density	1000	0	200 GB/Partition*	36	12	3
L1	Storage Optimized	10	10	1 TB/Partition*	36	12	12
L2	Storage Optimized	10	10	2 TB/Partition*	36	12	12

Select

Prices presented are estimates in your local currency that include only Azure infrastructure costs and any discounts for the subscription. Final charges will appear in your local currency in cost analysis and billing views. [View Azure pricing calculator](#)

7.

Microsoft Azure Search resources, services, and docs (G+)

Home > Azure AI services | AI Search >

## Create a search service

Basics Scale Networking Tags Review + create

**Project details**

Subscription \* Azure Pass - Sponsorship

Resource Group \* AI-FabricRG891  
[Create new](#)

**Instance Details**

Service name \* ⓘ mysearchservice859

Location \* East US 2

Pricing tier \* ⓘ

**Basic**

15 GB/Partition, max 3 replicas, max 3 partitions, max 9 search units

[Change Pricing Tier](#)

[Review + create](#) [Previous](#) [Next: Scale](#)

8. Once the Validation is passed, click on the **Create** button.

Microsoft Azure Search resources, services, and docs

Home > Azure AI services | AI Search >

## Create a search service

Validation Success

Basics Scale Networking Tags Review + create

**Basics**

Subscription	Azure Pass - Sponsorship
Resource Group	AI-FabricRG891
Location	East US 2
Service name	(new) mysearchservice859
Pricing tier	basic (15 GB/Partition, max 3 replicas, max 3 partitions, max 9 search units)
Estimated cost per month	\$75.14

**Scale**

Replicas	1
Partitions	1

**Networking**

Endpoint connectivity (data)	Public
------------------------------	--------

[Create](#) [Previous](#) [Next](#) [Download a template for automation](#)

9. After the deployment is completed, click on the **Go to resource** button.

10. copy **AI search name** and paste them in a notepad as shown in the below image, then **Save** the notepad to use the information in the upcoming lab.

The screenshot shows the Microsoft Azure portal interface for a Search service named 'mysearchservice21'. The service name is highlighted with a red box. The left sidebar contains a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Search management, Indexes, Indexers, Data sources, Aliases, Skillsets, Debug sessions, Settings, Semantic ranker, and Knowledge Center. The main content area displays the 'Essentials' section with the following details:

Property	Value
Resource group ( <a href="#">move</a> )	: <a href="#">AI-Fabric879</a>
Location ( <a href="#">move</a> )	: East US 2
Subscription ( <a href="#">move</a> )	: <a href="#">Azure Pass - Sponsorship</a>
Subscription ID	: 85677dc1-5950-436b-9280-7c7f9306d95c
Status	: Running
Tags ( <a href="#">edit</a> )	: <a href="#">Add tags</a>

Below the Essentials section, there is a tabbed interface with the following tabs: **Get started**, Properties, Usage, and Monitoring. The 'Get started' tab is currently selected. At the bottom right of the page, there is a blue cloud icon with a double-headed arrow.

#### Task : Create a lakehouse

1. New Item, Select **Lakehouse** to create a lakehouse.
2. In the **New lakehouse** dialog box, enter **!!\*\*data\_lakehouse\*\*!!** in the **Name** field, click on the **Create** button and open the new lakehouse.

**Note:** Ensure to remove space before **data\_lakehouse**.

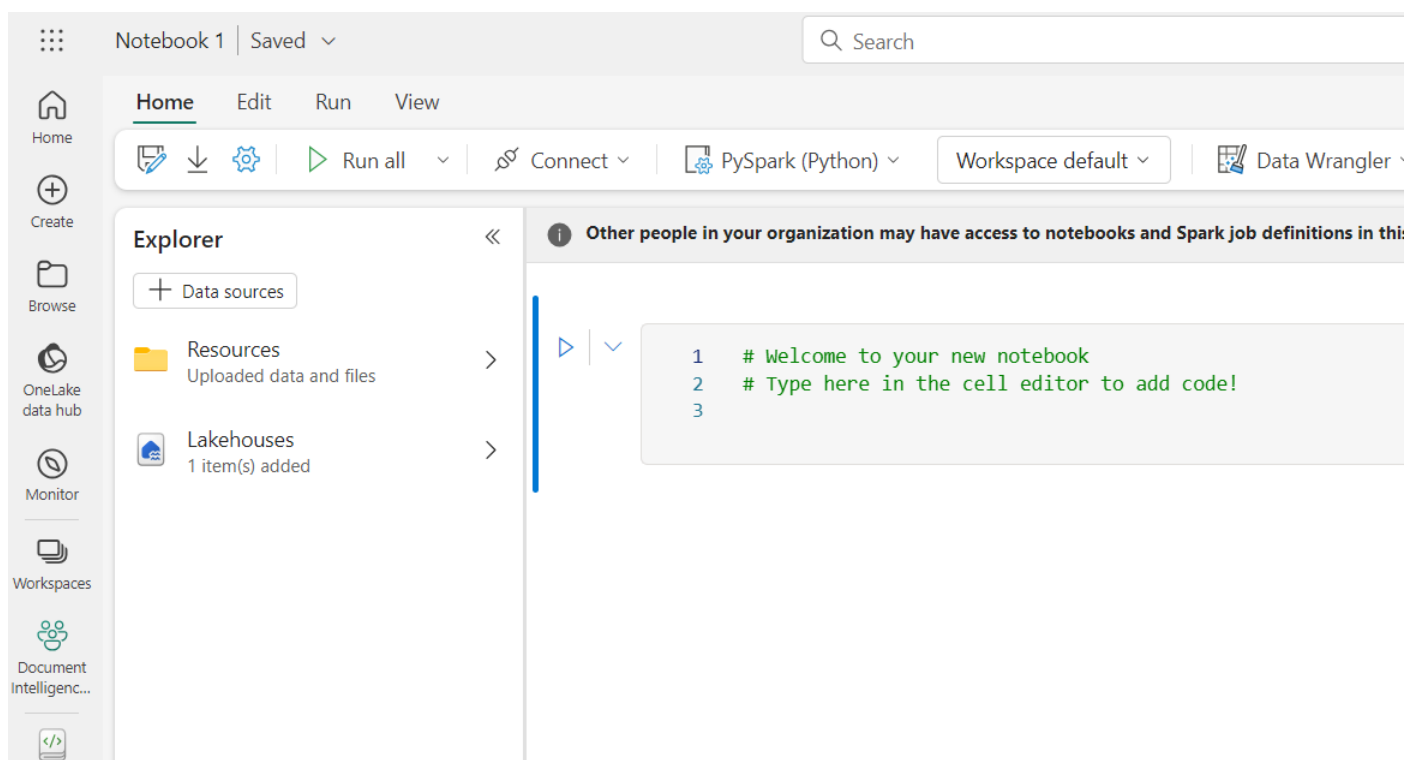
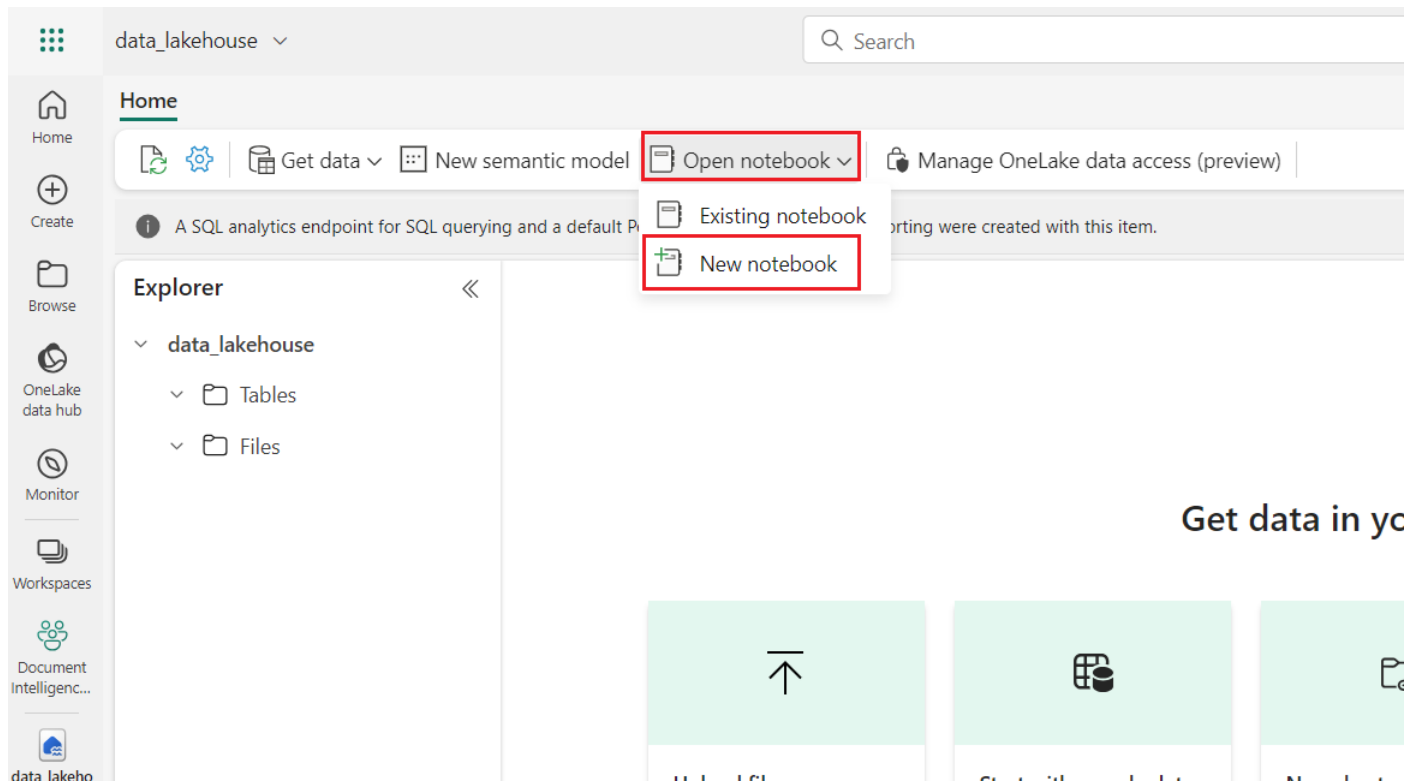
3. You will see a notification stating **Successfully created SQL endpoint**.

## Exercise 2: Loading and Pre-processing PDF Documents

### Task 1: Configure Azure API keys

To begin, navigate back to the rag\_workshop Lakehouse in your workspace and create a new notebook by selecting Open Notebook and selecting New Notebook from the options.

1. In the **Lakehouse** page, navigate and click on **Open notebook** drop in the command bar, then select **New notebook**.



2. In the query editor, paste the following code. Provide the keys for Azure AI Services, Azure Key Vault name and secrets to access the services

## Copy

# Azure AI Search

```
AI_SEARCH_NAME = ""
```

```
AI_SEARCH_INDEX_NAME = "rag-demo-index"
```

```
AI_SEARCH_API_KEY = ""
```

# Azure AI Services

```
AI_SERVICES_KEY = ""
```

```
AI_SERVICES_LOCATION = ""
```

Other people in your organization may have access to notebooks and Spark job definitions in this workspace. Carefully review and test before running in this environment.

**This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1).** Your notebook will use the updated runtime when you start a new Spark session and custom environments. [Learn more about Runtime 1.3](#)



```
1 # Azure AI Search
2 AI_SEARCH_NAME = "mysearchservice21" 1
3 AI_SEARCH_INDEX_NAME = "rag-demo-index"
4 AI_SEARCH_API_KEY = "8afQM7Y6A3sS7ZmXJZgg3ZkOL9yIW9e76jHmoeoBZqAzSeBoGMin" 2
5
6 # Azure AI Services
7 AI_SERVICES_KEY = "67a3cb7eac3d43f1b2e41b0676a27721" 3
8 AI_SERVICES_LOCATION = "eastus2" 4
9
```

[1] ✓ 9 sec - Apache Spark session ready in 7 sec 859 ms. Command executed in 1 sec 691 ms by MOD Administrator on 12:05:25 AM, 7/19/24

> Log

## Task 2: Loading & Analyzing the Document

1. we will be using a specific document named [support.pdf](#) which will be the source of our data.
2. To download the document, use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **Run cell** button and review the output

## Copy

```
import requests
```

```
import os
```

```
url = "https://github.com/Azure-Samples/azure-openai-rag-workshop/raw/main/data/support.pdf"
```

```
response = requests.get(url)
```

```
# Specify your path here
```

```
path = "/lakehouse/default/Files/"
```

```
# Ensure the directory exists
```

```
os.makedirs(path, exist_ok=True)
```



# Write the content to a file in the specified path

```
filename = url.rsplit("/")[-1]
```

```
with open(os.path.join(path, filename), "wb") as f:
```

```
    f.write(response.content)
```

Other people in your organization may have access to notebooks and Spark job definitions in this workspace. Carefully review this item before running it.

> Log

...

+ Code 1+ Markdown

▶ 3

```
1 import requests
2 import os
3
4 url = "https://github.com/Azure-Samples/azure-openai-rag-workshop/raw/main/data/support.pdf"
5 response = requests.get(url)
6
7 # Specify your path here
8 path = "/lakehouse/default/Files/"
9
10 # Ensure the directory exists
11 os.makedirs(path, exist_ok=True)
12
13 # Write the content to a file in the specified path
14 filename = url.rsplit("/")[-1]
15 with open(os.path.join(path, filename), "wb") as f:
16     f.write(response.content)
17
```

[2] ✓ 2 sec - Command executed in 1 sec 634 ms by MOD Administrator on 10:52:57 PM, 7/18/24

3. Now, load the PDF document into a Spark DataFrame using the `spark.read.format("binaryFile")` method provided by Apache Spark
4. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on ▶ **Run cell** button and review the output

### Copy

```
from pyspark.sql.functions import udf
```

```
from pyspark.sql.types import StringType
```

```
document_path = f"Files/{filename}"
```

```
df = spark.read.format("binaryFile").load(document_path).select("_metadata.file_name",  
"content").limit(10).cache()
```

```
display(df)
```

Other people in your organization may have access to notebooks and Spark job definitions in this workspace. Carefully review this item before running it.

[2] ✓ 2 sec - Command executed in 1 sec 634 ms by MOD Administrator on 10:52:57 PM, 7/18/24 PySpark (Python) ▾

...

+ Code + Markdown

3

```

1 from pyspark.sql.functions import udf
2 from pyspark.sql.types import StringType
3
4 document_path = f"Files/{filename}"
5
6 df = spark.read.format("binaryFile").load(document_path).select("_metadata.file_name", "content").limit(10).cache()
7
8 display(df)
9

```

2

[3] ✓ 9 sec - Command executed in 8 sec 287 ms by MOD Administrator on 10:54:59 PM, 7/18/24 PySpark (Python) ▾

> Spark jobs (1 of 1 succeeded) Resources Log

...

Table Chart |→ Download ▾ Showing rows 1 - 1 Inspect Search

ABC	file_name	ANY	content
1	support.pdf	JVBERi0xLjU...	

This code will read the PDF document and create a Spark DataFrame named df with the contents of the PDF. The DataFrame will have a schema that represents the structure of the PDF document, including its textual content.

- Next, we'll use the Azure AI Document Intelligence to read the PDF documents and extract the text from them.
- Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **▶ Run cell** button and review the output

### Copy

```
from synapse.ml.services import AnalyzeDocument
```

```
from pyspark.sql.functions import col
```

```

analyze_document = (
    AnalyzeDocument()
    .setPrebuiltModelId("prebuilt-layout")
    .setSubscriptionKey(AI_SERVICES_KEY)
    .setLocation(AI_SERVICES_LOCATION)
    .setImageBytesCol("content")
    .setOutputCol("result")
)

```

```

analyzed_df = (
    analyze_document.transform(df)
    .withColumn("output_content", col("result.analyzeResult.content"))
    .withColumn("paragraphs", col("result.analyzeResult.paragraphs"))
).cache()

```

This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1). Your notebook will use the updated runtime when you start a settings and custom environments. [Learn more about Runtime 1.3](#)

3

```
1 from synapse.ml.services import AnalyzeDocument
2 from pyspark.sql.functions import col
3
4 analyze_document = (
5     AnalyzeDocument()
6     .setPrebuiltModelId("prebuilt-layout")
7     .setSubscriptionKey('67a3cb7eac3d43f1b2e41b0676a27721') 1
8     .setLocation('eastus2')
9     .setImageBytesCol("content" 2)
10    .setOutputCol("result")
11 )
12
13 analyzed_df = (
14     analyze_document.transform(df)
15     .withColumn("output_content", col("result.analyzeResult.content"))
16     .withColumn("paragraphs", col("result.analyzeResult.paragraphs"))
17 ) .cache()
```

[4] ✓ 4 sec - Command executed in 3 sec 519 ms by MOD Administrator on 12:05:54 AM, 7/19/24

> Log

7. We can observe the analyzed Spark DataFrame named analyzed\_df using the following code. Note that we drop the content column as it is not needed anymore.
8. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **Run cell** button and review the output

## Copy

```
analyzed_df = analyzed_df.drop("content")
```

```
display(analyzed_df)
```

This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1). Your notebook will use the updated runtime when you start a new Spark session settings and custom environments. [Learn more about Runtime 1.3](#)

> Log

Code 1+ Markdown

3

```
1 analyzed_df = analyzed_df.drop("content") 2
2 display(analyzed_df)
```

[5] ✓ 9 sec - Command executed in 8 sec 202 ms by MOD Administrator on 12:06:13 AM, 7/19/24

> Spark jobs (2 of 2 succeeded) Resources Log

Table Chart | Download | Showing rows 1 - 1

	ABC file_name	ANY AnalyzeDocument_a723a8cca2...	ANY result	ABC output_content	ANY paragraphs
1	support.pdf	NULL	{"status":"s...	Contoso Real Estat...	[{"role":"title","c...

## Exercise 3: Generating and Storing Embeddings

### Task 1: Text Chunking

Before we can generate the embeddings, we need to split the text into chunks. To do this we leverage SynapseML's PageSplitter to divide the documents into smaller sections, which are subsequently stored in the chunks column. This allows for more granular representation and processing of the document content.

1. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **▶ Run cell** button and review the output

### Copy

```
from synapse.ml.featurize.text import PageSplitter
```

```
ps = (  
    PageSplitter()  
    .setInputCol("output_content")  
    .setMaximumPageLength(4000)  
    .setMinimumPageLength(3000)  
    .setOutputCol("chunks")  
)  
  
splitted_df = ps.transform(analyzed_df)  
display(splitted_df)
```

The screenshot shows a Databricks notebook interface. At the top, there are tabs for '+ Code' and '1+ Markdown'. Below the tabs is a code cell with a red border, containing the following Python code:

```
1 from synapse.ml.featurize.text import PageSplitter  
2  
3 ps = (  
4     PageSplitter()  
5     .setInputCol("output_content")  
6     .setMaximumPageLength(4000)  
7     .setMinimumPageLength(3000)  
8     .setOutputCol("chunks")  
9 )  
10  
11 splitted_df = ps.transform(analyzed_df)  
12 display(splitted_df)
```

Below the code cell, there is a status bar indicating '[6] ✓ 2 sec - Command executed in 1 sec 577 ms by MOD Administrator on 12:06:25 AM, 7/19/24'. Below the status bar, there are tabs for 'Spark jobs (1 of 1 succeeded)', 'Resources', and 'Log'. Below the tabs, there is a table with the following columns: 'file\_name', 'result', 'output\_content', 'paragraphs', and 'chunks'. The table shows one row of data for 'support.pdf'.

file_name	result	output_content	paragraphs	chunks
support.pdf	NULL	Contoso Real Estat...	[{"role": "title", "c...	[{"Contoso R...

Note that the chunks for each document are presented in a single row inside an array. In order to embed all the chunks in the following cells, we need to have each chunk in a separate row.

2. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **▶ Run cell** button and review the output

### Copy

```
from pyspark.sql.functions import posexplode, col, concat
```

```
# Each "chunks" column contains the chunks for a single document in an array
```

```
# The posexplode function will separate each chunk into its own row
```

```
exploded_df = splitted_df.select("file_name", posexplode(col("chunks")).alias("chunk_index", "chunk"))
```

```
# Add a unique identifier for each chunk
```

```
exploded_df = exploded_df.withColumn("unique_id", concat(exploded_df.file_name, exploded_df.chunk_index))
```

```
display(exploded_df)
```

The screenshot shows a Databricks notebook interface. At the top, there are tabs for '+ Code' and '+ Markdown'. Below the tabs is a code editor with a red border containing the following PySpark code:

```
1 from pyspark.sql.functions import posexplode, col, concat
2
3 # Each "chunks" column contains the chunks for a single document in an array
4 # The posexplode function will separate each chunk into its own row
5 exploded_df = splitted_df.select("file_name", posexplode(col("chunks")).alias("chunk_index", "chunk"))
6
7 # Add a unique identifier for each chunk
8 exploded_df = exploded_df.withColumn("unique_id", concat(exploded_df.file_name, exploded_df.chunk_index))
9
10 display(exploded_df)
```

Below the code editor, there is a status bar indicating the command was executed successfully in 793 ms. Below that, there is a table view showing the first 3 rows of the output:

file_name	chunk_index	chunk	unique_id
support.pdf	0	Contoso Re...	support.pdf0
support.pdf	1	inaccurate i...	support.pdf1
support.pdf	2	Document ...	support.pdf2

From this code snippet we first explode these arrays so there is only one chunk in each row, then filter the Spark DataFrame in order to only keep the path to the document and the chunk in a single row.

## Task 2: Generating Embeddings

Next we'll generate the embeddings for each chunk. To do this we utilize both SynapseML and Azure OpenAI Service. By integrating the built in Azure OpenAI service with SynapseML, we can leverage the power of the Apache Spark distributed computing framework to process numerous prompts using the OpenAI service.

1. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **Run cell** button and review the output

### Copy

```
from synapse.ml.services import OpenAIEmbedding
```

```
embedding = (
```

```
    OpenAIEmbedding()
```

```
    .setDeploymentName("text-embedding-ada-002")
```

```
    .setTextCol("chunk")
```

```

.setErrorCol("error")

.setOutputCol("embeddings")
)

```

```
df_embeddings = embedding.transform(exploded_df)
```

```
display(df_embeddings)
```

**This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1).** Your notebook will use the updated runtime when you start a new environment. [Learn more about Runtime 1.3](#)

**+ Code**   **+ Markdown**

**▶**

```

1  from synapse.ml.services import OpenAIEmbedding
2
3  embedding = (
4      OpenAIEmbedding()
5      .setDeploymentName("text-embedding-ada-002")
6      .setTextCol("chunk")
7      .setErrorCol("error")
8      .setOutputCol("embeddings")
9  )
10
11 df_embeddings = embedding.transform(exploded_df)
12
13 display(df_embeddings)
14

```

[8] ✓ 5 sec - Command executed in 4 sec 855 ms by MOD Administrator on 12:07:02 AM, 7/19/24

> **Spark jobs** (1 of 1 succeeded)   **Resources**   **Log**

...

**Table**   **Chart**   **Download**   **Showing rows 1 - 3**

	ABC file_name	123 chunk_index	ABC chunk	ABC unique_id	ANY error	ANY embeddings
1	support.pdf	0	Contoso Re...	support.pdf0	NULL	{"type":1,"values"...
2	support.pdf	1	inaccurate i...	support.pdf1	NULL	{"type":1,"values"...
3	support.pdf	2	Document ...	support.pdf2	NULL	{"type":1,"values"...

This integration enables the SynapseML embedding client to generate embeddings in a distributed manner, enabling efficient processing of large volumes of data

### Task 3: Storing Embeddings

**Azure AI Search** [https://learn.microsoft.com/azure/search/search-what-is-azure-search?WT.mc\\_id=data-114676-jndemenge](https://learn.microsoft.com/azure/search/search-what-is-azure-search?WT.mc_id=data-114676-jndemenge) is a powerful search engine that includes the ability to perform full text search, vector search, and hybrid search. For more examples of its vector search capabilities, see the **azure-search-vector-samples repository** <https://github.com/Azure/azure-search-vector-samples/>.

Storing data in Azure AI Search involves two main steps:

**Creating the index:** The first step is to define the schema of the search index, which includes the properties of each field as well as any vector search strategies that will be used.

**Adding chunked documents and embeddings:** The second step is to upload the chunked documents, along with their corresponding embeddings, to the index. This allows for efficient storage and retrieval of the data using hybrid and vector search.

1. The following code snippet demonstrates how to create an index in Azure AI Search using the Azure AI Search REST API. This code creates an index with fields for the unique identifier of each document, the text content of the document, and the vector embedding of the text content.
2. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on ► **Run cell** button and review the output

### Copy

```
import requests
```

```
import json
```

```
# Length of the embedding vector (OpenAI ada-002 generates embeddings of length 1536)
```

```
EMBEDDING_LENGTH = 1536
```

```
# Create index for AI Search with fields id, content, and contentVector
```

```
# Note the datatypes for each field below
```

```
url = f"https://{AI_SEARCH_NAME}.search.windows.net/indexes/{AI_SEARCH_INDEX_NAME}?api-version=2023-11-01"
```

```
payload = json.dumps(
```

```
{
```

```
    "name": AI_SEARCH_INDEX_NAME,
```

```
    "fields": [
```

```
        # Unique identifier for each document
```

```
{
```

```
    "name": "id",
```

```
    "type": "Edm.String",
```

```
    "key": True,
```

```
    "filterable": True,
```

```
},
```

```
        # Text content of the document
```

```
{
```

```
    "name": "content",
```

```
    "type": "Edm.String",
```

```
    "searchable": True,
```

```
    "retrievable": True,
```

```
},
```

```
        # Vector embedding of the text content
```

```
{
```

```

        "name": "contentVector",
        "type": "Collection(Edm.Single)",
        "searchable": True,
        "retrievable": True,
        "dimensions": EMBEDDING_LENGTH,
        "vectorSearchProfile": "vectorConfig",
    },
],
"vectorSearch": {
    "algorithms": [{"name": "hnswConfig", "kind": "hnsw", "hnswParameters": {"metric": "cosine"}}],
    "profiles": [{"name": "vectorConfig", "algorithm": "hnswConfig"}],
},
}
)

headers = {"Content-Type": "application/json", "api-key": AI_SEARCH_API_KEY}

response = requests.request("PUT", url, headers=headers, data=payload)

if response.status_code == 201:
    print("Index created!")
elif response.status_code == 204:
    print("Index updated!")
else:
    print(f"HTTP request failed with status code {response.status_code}")
    print(f"HTTP response body: {response.text}")

```



Other people in your organization may have access to notebooks and Spark job definitions in this workspace. Carefully review this item before running it.

This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1). Your notebook will use the updated runtime when you start a new Spark session. Update in your environments. [Learn more about Runtime 1.3](#)

+ Code 1+ Markdown

▶ 3

```

1 import requests
2 import json
3
4 # Length of the embedding vector (OpenAI ada-002 generates embeddings of length 1536)
5 EMBEDDING_LENGTH = 1536
6
7 # Create index for AI Search with fields id, content, and contentVector
8 # Note the datatypes for each field below
9 url = f"https://mysearchservice21.search.windows.net/indexes/{AI_SEARCH_INDEX_NAME}?api-version=2023-11-01"
10 payload = json.dumps(
11     {
12         "name": AI_SEARCH_INDEX_NAME,
13         "fields": [
14             # Unique identifier for each document
15             {
16                 "name": "id",
17                 "type": "Edm.String",
18                 "key": True,
19                 "filterable": True,
20             },
21             # Text content of the document
22             {
23                 "name": "content",
24                 "type": "Edm.String",
25                 "searchable": True,
26                 "retrievable": True,

```

2

Other people in your organization may have access to notebooks and Spark job definitions in this workspace. Carefully review this item before running it.

This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1). Your notebook will use the updated runtime when you start a new Spark session. Update in your environments. [Learn more about Runtime 1.3](#)

```

33         retrievable : true,
34         "dimensions": EMBEDDING_LENGTH,
35         "vectorSearchProfile": "vectorConfig",
36     },
37 ],
38     "vectorSearch": {
39         "algorithms": [{"name": "hnswnnConfig", "kind": "hnswnn", "hnswnnParameters": {"metric": "cosine"}}],
40         "profiles": [{"name": "vectorConfig", "algorithm": "hnswnnConfig"}],
41     },
42 }
43 )
44 headers = {"Content-Type": "application/json", "api-key": AI_SEARCH_API_KEY}
45
46 response = requests.request("PUT", url, headers=headers, data=payload)
47 if response.status_code == 201:
48     print("Index created!")
49 elif response.status_code == 204:
50     print("Index updated!")
51 else:
52     print(f"HTTP request failed with status code {response.status_code}")
53     print(f"HTTP response body: {response.text}")
54

```

[9] ✓ 1 sec - Command executed in 1 sec 492 ms by MOD Administrator on 12:07:11 AM, 7/19/24

... Index created!

- The next step is to upload the chunks to the newly created Azure AI Search index. The Azure AI Search REST API supports up to 1000 "documents" per request. Note that in this case, each of our "documents" is in fact a chunk of the original file
- Use the + **Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on ▶ **Run cell** button and review the output

Copy

import re

```
from pyspark.sql.functions import monotonically_increasing_id
```

```
def insert_into_index(documents):
```

""Uploads a list of 'documents' to Azure AI Search index.""

```
url = f"https://{AI_SEARCH_NAME}.search.windows.net/indexes/{AI_SEARCH_INDEX_NAME}/docs/index?api-version=2023-11-01"
```

```
payload = json.dumps({"value": documents})
```

```
headers = {
```

```
"Content-Type": "application/json",
```

```
"api-key": AI_SEARCH_API_KEY,
```

}

```
response = requests.request("POST", url, headers=headers, data=payload)
```

```
if response.status_code == 200 or response.status_code == 201:
```

```
return "Success"
```

```
else:
```

```
return f"Failure: {response.text}"
```

```
def make_safe_id(row_id: str):
```

""Strips disallowed characters from row id for use as Azure AI search document ID.""

```
return re.sub("[^0-9a-zA-Z -]", " ", row_id)
```

```
def upload rows(rows):
```

""Uploads the rows in a Spark dataframe to Azure AI Search.

Limits uploads to 1000 rows at a time due to Azure AI Search API limits.

■■■■■

BATCH SIZE = 1000

```
rows = list(rows)
```

```
for i in range(0, len(rows), BATCH_SIZE):
```

```
row_batch = rows[i : i + BATCH_SIZE]
```

```
documents = []
```

```
for row in rows:
```

```
documents.append(
```

{

```
"id": make_safe_id(row["unique id"]),
```

```

        "content": row["chunk"],
        "contentVector": row["embeddings"].tolist(),
        "@search.action": "upload",
    },
)

status = insert_into_index(documents)

yield [row_batch[0]["row_index"], row_batch[-1]["row_index"], status]

```

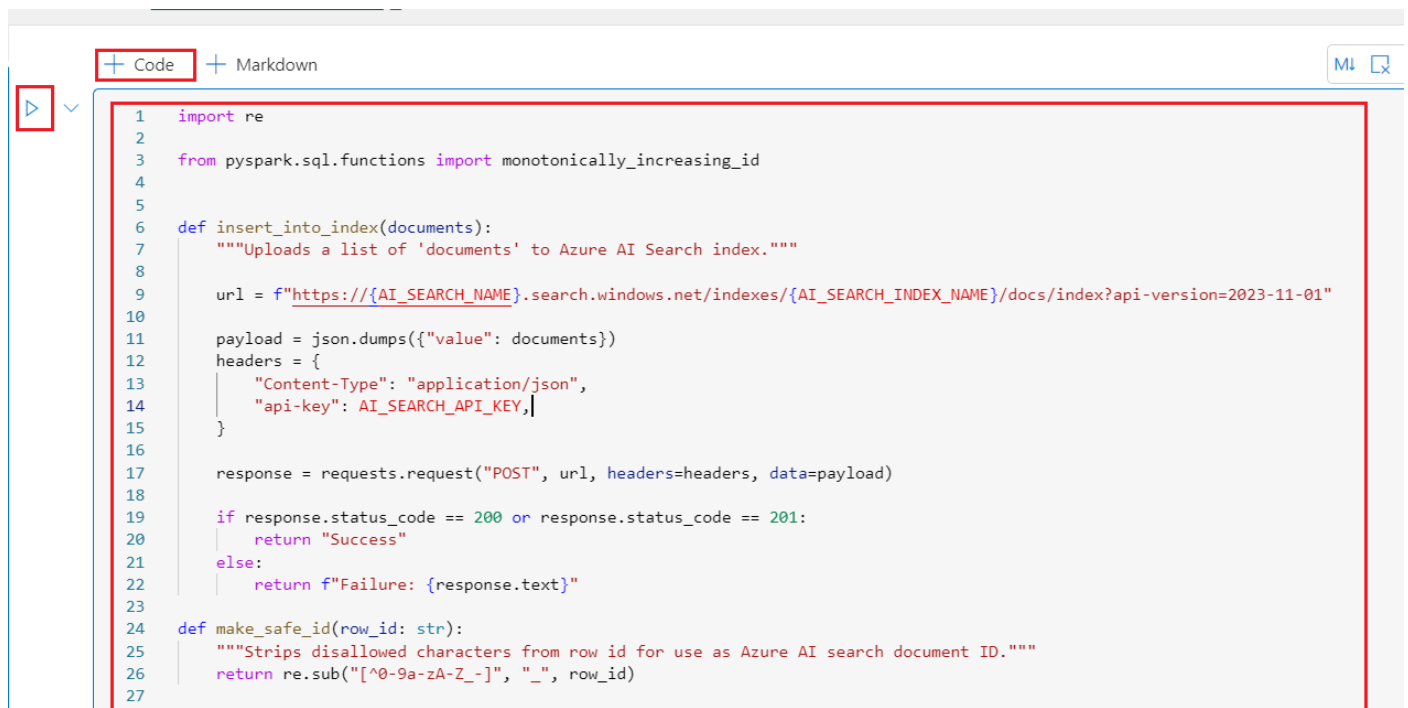
# Add ID to help track what rows were successfully uploaded

```
df_embeddings = df_embeddings.withColumn("row_index", monotonically_increasing_id())
```

# Run upload\_batch on partitions of the dataframe

```
res = df_embeddings.rdd.mapPartitions(upload_rows)
```

```
display(res.toDF(["start_index", "end_index", "insertion_status"]))
```



```

1  import re
2
3  from pyspark.sql.functions import monotonically_increasing_id
4
5
6  def insert_into_index(documents):
7      """Uploads a list of 'documents' to Azure AI Search index."""
8
9      url = f"https://{AI_SEARCH_NAME}.search.windows.net/indexes/{AI_SEARCH_INDEX_NAME}/docs/index?api-version=2023-11-01"
10
11     payload = json.dumps({"value": documents})
12     headers = {
13         "Content-Type": "application/json",
14         "api-key": AI_SEARCH_API_KEY,
15     }
16
17     response = requests.request("POST", url, headers=headers, data=payload)
18
19     if response.status_code == 200 or response.status_code == 201:
20         return "Success"
21     else:
22         return f"Failure: {response.text}"
23
24     def make_safe_id(row_id: str):
25         """Strips disallowed characters from row id for use as Azure AI search document ID."""
26         return re.sub("[^0-9a-zA-Z_-]", "_", row_id)
27

```

**This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1).** Your notebook will use the updated runtime when you start a new Spark session. Update environments. [Learn more about Runtime 1.3](#)

```
41         "id": make_safe_id(row["unique_id"]),
42         "content": row["chunk"],
43         "contentVector": row["embeddings"].tolist(),
44         "@search.action": "upload",
45     },
46 )
47 status = insert_into_index(documents)
48 yield [row_batch[0]["row_index"], row_batch[-1]["row_index"], status]
49
50 # Add ID to help track what rows were successfully uploaded
51 df_embeddings = df_embeddings.withColumn("row_index", monotonically_increasing_id())
52
53 # Run upload_batch on partitions of the dataframe
54 res = df_embeddings.rdd.mapPartitions(upload_rows)
55 display(res.toDF(["start_index", "end_index", "insertion_status"]))
56
```

[10] ✓ 5 sec - Command executed in 4 sec 972 ms by MOD Administrator on 12:07:52 AM, 7/19/24

> Spark jobs (2 of 2 succeeded) Resources Log

Table  Chart    Download  Showing rows 1 - 1				
	12L start_index	12L end_index	ABC insertion_status	
1	0	2	Success	

#### Exercise 4: Retrieving Relevant Documents and Answering Questions

After processing the document, we can proceed to pose a question. We will use SynapseML to convert the user's question into an embedding and then utilize cosine similarity to retrieve the top K document chunks that closely match the user's question.

##### Task 1: Configure Environment & Azure API Keys

Create a new notebook in the Lakehouse and save it as rag\_application. We'll use this notebook to build the RAG application.

1. Provide the credentials for access to Azure AI Search. You can copy the values from the from Azure Portal.(Exercise 1>Task 4)
2. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **▶ Run cell** button and review the output

Copy

# Azure AI Search

AI\_SEARCH\_NAME = "

AI\_SEARCH\_INDEX\_NAME = 'rag-demo-index'

AI\_SEARCH\_API\_KEY = "

This environment has been updated to Fabric Runtime 1.3 (Spark 3.3 and Delta 3.1). Your notebook will use the updated runtime when you save the notebook. [Learn more about Runtime 1.3](#)

1	0	2	Success
---	---	---	---------

**+ Code** + Markdown



```
1 # Azure AI Search
2 AI_SEARCH_NAME = 'mysearchservice21'
3 AI_SEARCH_INDEX_NAME = 'rag-demo-index'
4 AI_SEARCH_API_KEY = '8afQM7Y6A3sS7ZmXJZgg3Zk0L9yIW9e76jHmoeoBZqAzSeBoGMin'
5
```

[11] ✓ <1 sec - Command executed in 275 ms by MOD Administrator on 12:13:33 AM, 7/19/24

3. The following function takes a user's question as input and converts it into an embedding using the text-embedding-ada-002 model. This code assumes you're using the Pre-built AI Services in Microsoft Fabric
4. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **Run cell** button and review the output

### Copy

```
def gen_question_embedding(user_question):
    """Generates embedding for user_question using SynapseML."""
    from synapse.ml.services import OpenAIEmbedding
    df_ques = spark.createDataFrame([(user_question, 1)], ["questions", "dummy"])
    embedding = (
        OpenAIEmbedding()
        .setDeploymentName('text-embedding-ada-002')
        .setTextCol("questions")
        .setErrorCol("errorQ")
        .setOutputCol("embeddings")
    )
    df_ques_embeddings = embedding.transform(df_ques)
    row = df_ques_embeddings.collect()[0]
    question_embedding = row.embeddings.tolist()
    return question_embedding
```

+ Code + Markdown

```

1 def gen_question_embedding(user_question):
2     """Generates embedding for user_question using SynapseML."""
3     from synapse.ml.services import OpenAIEmbedding
4
5     df_ques = spark.createDataFrame([(user_question, 1)], ["questions", "dummy"])
6     embedding = (
7         OpenAIEmbedding()
8         .setDeploymentName('text-embedding-ada-002')
9         .setTextCol("questions")
10        .setErrorCol("errorQ")
11        .setOutputCol("embeddings")
12    )
13    df_ques_embeddings = embedding.transform(df_ques)
14    row = df_ques_embeddings.collect()[0]
15    question_embedding = row.embeddings.tolist()
16    return question_embedding
17

```

[12] ✓ <1 sec - Command executed in 229 ms by MOD Administrator on 12:13:55 AM, 7/19/24

## Task 2: Retrieve Relevant Documents

1. The next step is to use the user question and its embedding to retrieve the top K most relevant document chunks from the search index. The following function retrieves the top K entries using hybrid search
2. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **Run cell** button and review the output

### Copy

```
import json
```

```
import requests
```

```
def retrieve_top_chunks(k, question, question_embedding):
```

```
    """Retrieve the top K entries from Azure AI Search using hybrid search."""
```

```
    url = f"https://{AI_SEARCH_NAME}.search.windows.net/indexes/{AI_SEARCH_INDEX_NAME}/docs/search?api-version=2023-11-01"
```

```
    payload = json.dumps({
```

```
        "search": question,
```

```
        "top": k,
```

```
        "vectorQueries": [
```

```
            {
```

```
                "vector": question_embedding,
```

```
                "k": k,
```

```
                "fields": "contentVector",
```

```
                "kind": "vector"
```

```

    }
]
})

headers = {

    "Content-Type": "application/json",

    "api-key": AI_SEARCH_API_KEY,

}

response = requests.request("POST", url, headers=headers, data=payload)

output = json.loads(response.text)

return output

```

This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1). Your notebook will use the updated runtime when you start a new Spark session. Update in your workspace setting environments. [Learn more about Runtime 1.3](#)

+ Code + Markdown

```

1  import json
2  import requests
3
4  def retrieve_top_chunks(k, question, question_embedding):
5      """Retrieve the top K entries from Azure AI Search using hybrid search."""
6      url = f"https://{AI_SEARCH_NAME}.search.windows.net/indexes/{AI_SEARCH_INDEX_NAME}/docs/search?api-version=2023-11-01"
7
8      payload = json.dumps({
9          "search": question,
10         "top": k,
11         "vectorQueries": [
12             {
13                 "vector": question_embedding,
14                 "k": k,
15                 "fields": "contentVector",
16                 "kind": "vector"
17             }
18         ]
19     })
20
21     headers = {
22         "Content-Type": "application/json",
23         "api-key": AI_SEARCH_API_KEY,
24     }
25
26     response = requests.request("POST", url, headers=headers, data=payload)
27     output = json.loads(response.text)
28

```

With those functions defined, we can define a function that takes a user's question, generates an embedding for the question, retrieves the top K document chunks, and concatenates the content of the retrieved documents to form the context for the user's question.

3. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **▶ Run cell** button and review the output

### Copy

```

def get_context(user_question, retrieved_k = 5):

    # Generate embeddings for the question

    question_embedding = gen_question_embedding(user_question)

    # Retrieve the top K entries

    output = retrieve_top_chunks(retrieved_k, user_question, question_embedding)

    # concatenate the content of the retrieved documents

    context = [chunk["content"] for chunk in output["value"]]

```

return context

**This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1).** Your notebook will use the updated runtime when you start a new Spark environment. [Learn more about Runtime 1.3](#)

[13] ✓ <1 sec - Command executed in 242 ms by MOD Administrator on 12:14:23 AM, 7/19/24

+ Code + Markdown

```
1 def get_context(user_question, retrieved_k = 5):
2     # Generate embeddings for the question
3     question_embedding = gen_question_embedding(user_question)
4
5     # Retrieve the top K entries
6     output = retrieve_top_chunks(retrieved_k, user_question, question_embedding)
7
8     # concatenate the content of the retrieved documents
9     context = [chunk["content"] for chunk in output["value"]]
10
11     return context
12
```

[14] ✓ 1 sec - Command executed in 253 ms by MOD Administrator on 12:14:58 AM, 7/19/24

### Task 3: Answering the User's Question

Finally, we can define a function that takes a user's question, retrieves the context for the question, and sends both the context and the question to a large language model to generate a response. For this demo, we'll use the gpt-35-turbo-16k, a model that is optimized for conversation.

1. Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **▶ Run cell** button and review the output

#### Copy

```
from pyspark.sql import Row
```

```
from synapse.ml.services.openai import OpenAIChatCompletion
```

```
def make_message(role, content):
```

```
    return Row(role=role, content=content, name=role)
```

```
def get_response(user_question):
```

```
    context = get_context(user_question)
```

```
    # Write a prompt with context and user_question as variables
```

```
    prompt = f"""
```

```
    context: {context}
```

```
    Answer the question based on the context above.
```

```
    If the information to answer the question is not present in the given context then reply "I don't know".
```



```
"""
```

```
chat_df = spark.createDataFrame(
    [
        (
            [
                make_message(
                    "system", prompt
                ),
                make_message("user", user_question),
            ],
        ),
    ]
).toDF("messages")

chat_completion = (
    OpenAIChatCompletion()
    .setDeploymentName("gpt-35-turbo-16k") # deploymentName could be one of {gpt-35-turbo, gpt-35-turbo-16k}
    .setMessagesCol("messages")
    .setErrorCol("error")
    .setOutputCol("chat_completions")
)

result_df = chat_completion.transform(chat_df).select("chat_completions.choices.message.content")

result = []

for row in result_df.collect():
    content_string = ' '.join(row['content'])
    result.append(content_string)

# Join the list into a single string
result = ' '.join(result)

return result
```

Code Markdown



```
1 from pyspark.sql import Row
2 from synapse.ml.services.openai import OpenAIChatCompletion
3
4
5 def make_message(role, content):
6     return Row(role=role, content=content, name=role)
7
8 def get_response(user_question):
9     context = get_context(user_question)
10
11     # Write a prompt with context and user_question as variables
12     prompt = f"""
13     context: {context}
14     Answer the question based on the context above.
15     If the information to answer the question is not present in the given context then reply "I don't know".
16     """
17
18     chat_df = spark.createDataFrame(
19         [
20             (
21                 [
22                     make_message(
23                         "system", prompt
24                     ),
25                     make_message("user", user_question),
26                 ]
27             )
28         ]
29     ).toDF("messages")
30
31 chat_completion = (
32     OpenAIChatCompletion()
33     .setDeploymentName("gpt-35-turbo-16k") # deploymentName could be one of {gpt-35-turbo, gpt-35-turbo-16k}
34     .setMessagesCol("messages")
35     .setErrorCol("error")
36     .setOutputCol("chat_completions")
37 )
38
39 result_df = chat_completion.transform(chat_df).select("chat_completions.choices.message.content")
40
41 result = []
42 for row in result_df.collect():
43     content_string = ' '.join(row['content'])
44     result.append(content_string)
45
46 # Join the list into a single string
47 result = ' '.join(result)
48
49 return result
50
```

```
27         ),
28     ]
29 ).toDF("messages")
30
31 chat_completion = (
32     OpenAIChatCompletion()
33     .setDeploymentName("gpt-35-turbo-16k") # deploymentName could be one of {gpt-35-turbo, gpt-35-turbo-16k}
34     .setMessagesCol("messages")
35     .setErrorCol("error")
36     .setOutputCol("chat_completions")
37 )
38
39 result_df = chat_completion.transform(chat_df).select("chat_completions.choices.message.content")
40
41 result = []
42 for row in result_df.collect():
43     content_string = ' '.join(row['content'])
44     result.append(content_string)
45
46 # Join the list into a single string
47 result = ' '.join(result)
48
49 return result
50
```

[15] ✓ <1 sec - Command executed in 239 ms by MOD Administrator on 12:15:20 AM, 7/19/24

- Now, we can call that function with an example question to see the response:
- Use the **+ Code** icon below the cell output to add a new code cell to the notebook, and enter the following code in it. Click on **Run cell** button and review the output

### Copy

```
user_question = "how do i make a booking?"
```

```
response = get_response(user_question)
```

```
print(response)
```

**This environment has been updated to Fabric Runtime 1.3 (Spark 3.5 and Delta 3.1).** Your notebook will use the updated runtime when you start a new Spark session. Update in your workspace settings and custom environments. [Learn more about Runtime 1.3](#)

+ Code + Markdown

M:

```
1 user_question = "how do i make a booking?"
2 response = get_response(user_question)
3 print(response)
4
```

[16] ✓ 8 sec - Command executed in 8 sec 18 ms by MOD Administrator on 12:15:56 AM, 7/19/24

PySpark (Py

> Spark jobs (2 of 2 succeeded) Resources Log

... To make a booking on Contoso Real Estate, follow these steps:

1. Search for Rentals:
  - Enter your destination, check-in and check-out dates, and the number of guests.
  - Apply filters such as price range, property type, and amenities to narrow down your options.
  - Browse through the listings to find the perfect place for your stay.
2. View Listing Details:
  - Click on a listing to view detailed information, including photos, property description, reviews, and host information.
3. Make a Booking:
  - Click the "Book Now" button on the listing page.
  - Review the booking details, including the total cost and house rules.
  - Confirm your booking by providing payment information.
  - Once the host accepts your booking, you'll receive a confirmation.

Please note that Contoso Real Estate handles the payment process securely, and you'll only be charged once your booking is confirmed. You can also communicate with the host through our messaging system for any questions or special requests.