**Lab 07: Use Azure OpenAI Service embeddings to perform document search**

**Introduction**

An embedding is a special format of data representation that can be easily utilized by machine learning models and algorithms. The embedding is an information-dense representation of the semantic meaning of a piece of text. Each embedding is a vector of floating-point numbers, such that the distance between two embeddings in the vector space is correlated with semantic similarity between two inputs in the original format. For example, if two texts are similar, then their vector representations should also be similar. Azure OpenAI embeddings rely on cosine similarity to compute the similarity between documents and a query.

**Objectives**

* To deploy text-embedding-ada-002 model and configure the bill\_sum\_data.csv file.
* To configure the environmental variables, import libraries and list models, clean the data for tokenization, and understand how tokenization works.
* To perform document search in Jupyter Notebook against the entire knowledge base.

**Prerequisites**

* Before starting the Lab 06 complete the Lab 01- Provisioning Azure OpenAI resource

**Exercise 1: Deploy the Embeddings model and perform the document search**

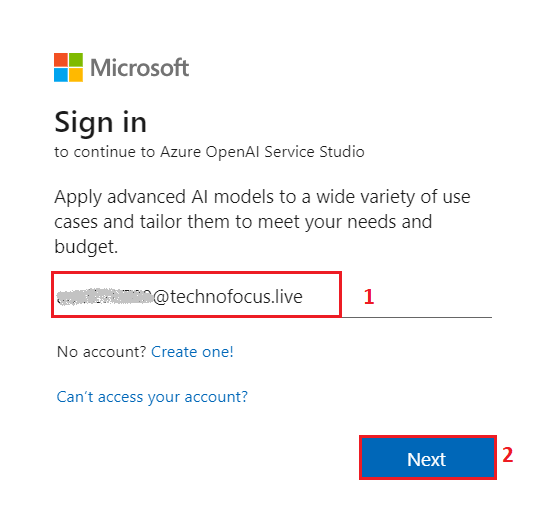
**Task 1: Create a resource and deploy the Embeddings model using Azure OpenAI**

1. Open your browser, navigate to the address bar, and type or paste the following URL: **https://oai.azure.com/** then press the **Enter** button.

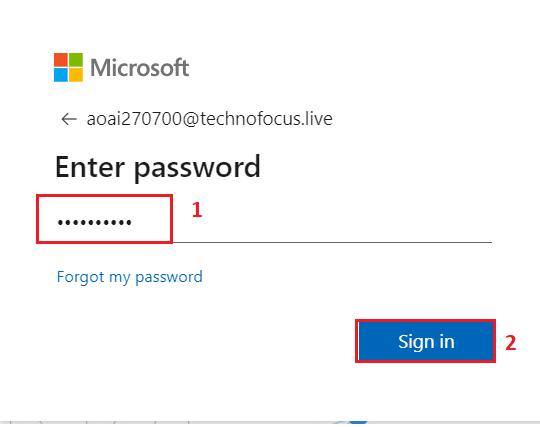
A screenshot of a browser

Description automatically generated

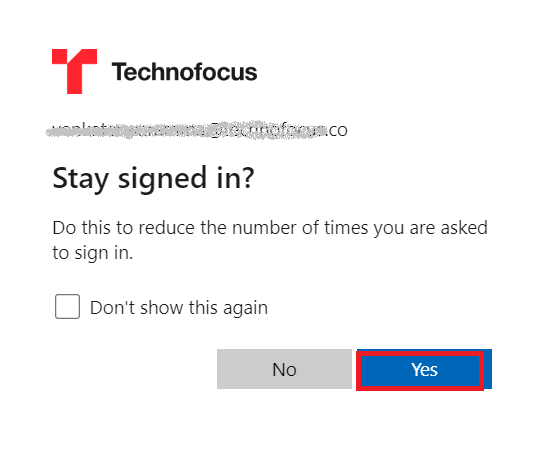
1. In the **Microsoft Azure** window, enter your **Sign-in** credentials, and click on the **Next** button.



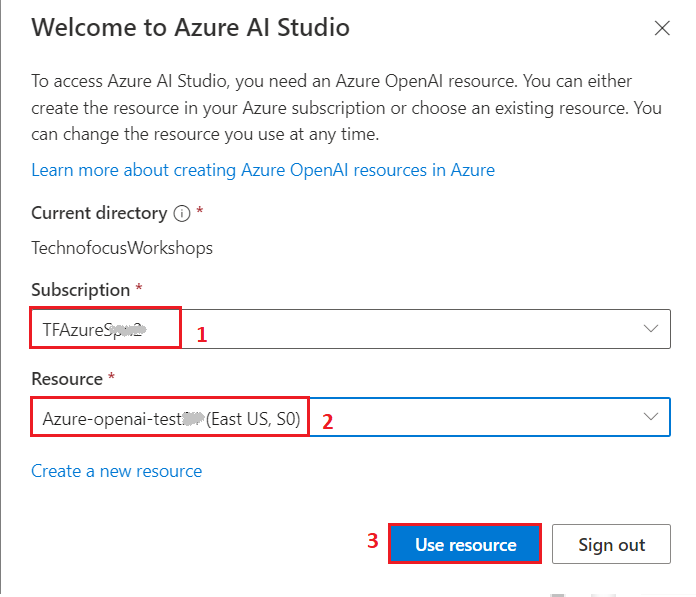
1. Then, enter the password and click on the **Sign in** button.



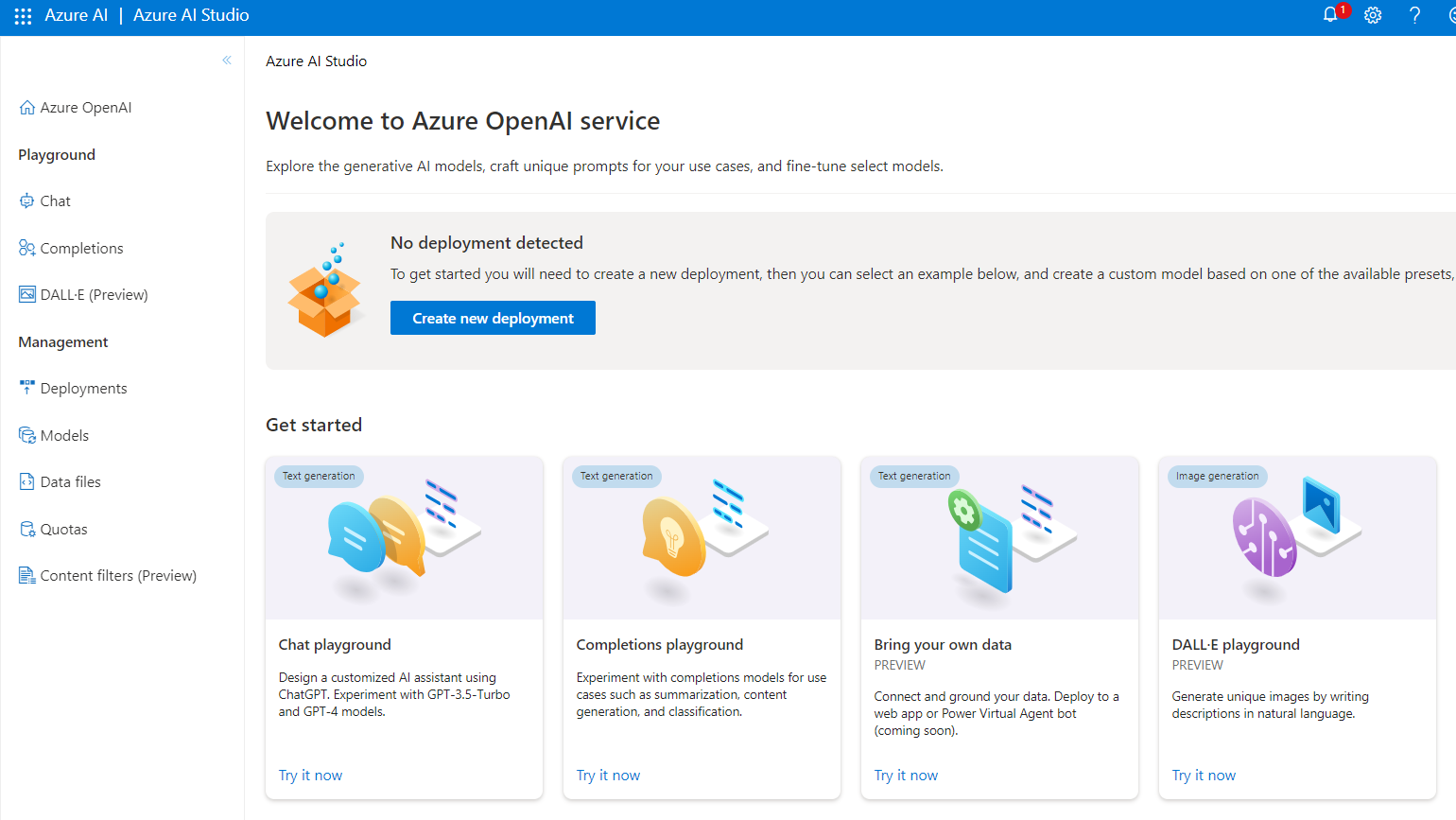
1. In **Stay signed in?** window, click on the **Yes** button.



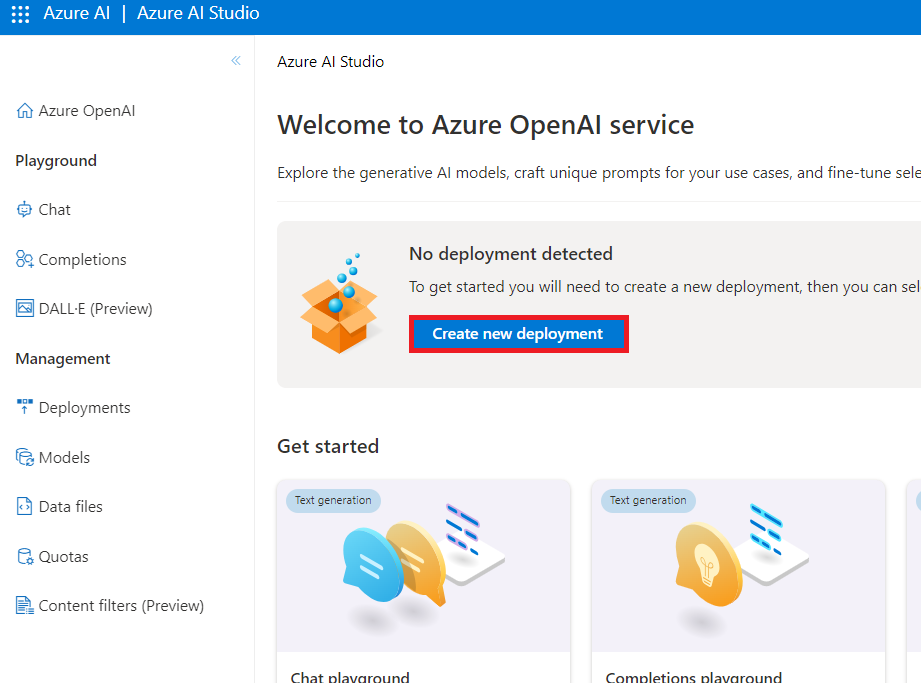
1. On the **Welcome to Azure OpenAI Studio** dialog box, under the **Subscription** field, enter the subscription assigned to you, and in the **Resource** field, enter the assigned Resource, and click on the **Use resource** button.



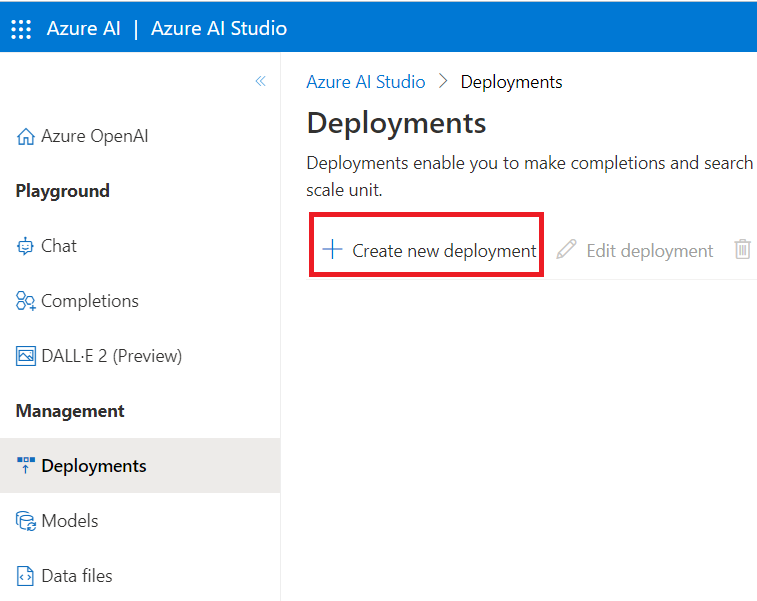
1. Wait for the Azure OpenAI studio to launch.



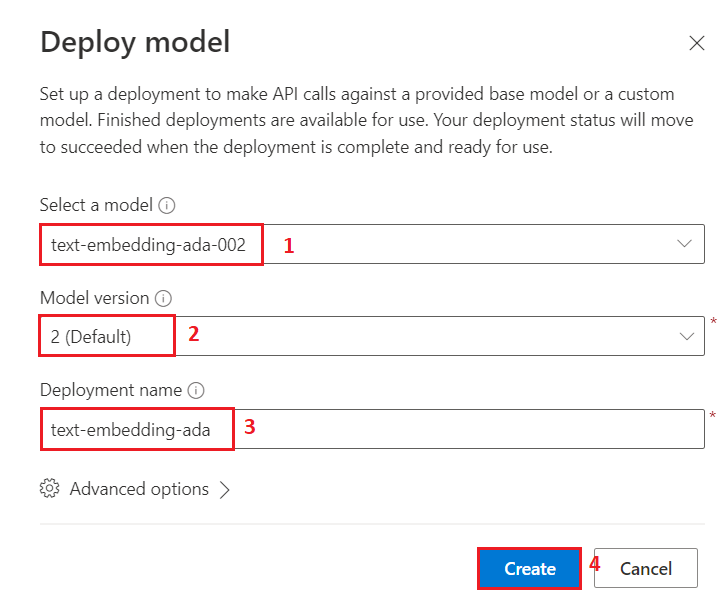
1. On the **Azure OpenAI Studio** homepage, click on **Create new deployment** button.



1. In the **Deployments** page, click on +**Create new deployment**.

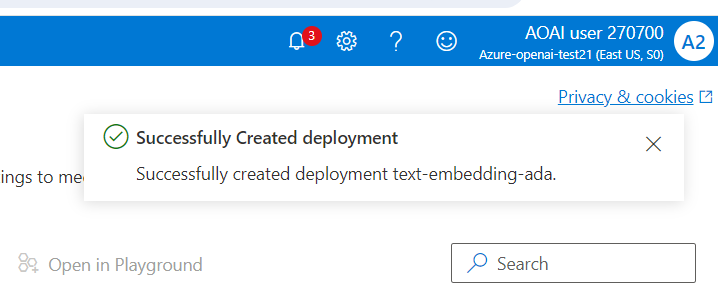


1. In the **Deploy model** dialog box, under **Select a model** click on the dropdown select **text-embedding-ada-02** field, under **Model version** select **2(Default)** and under **Deployment name** enter **text-embedding-ada** Click on the **Create** button.



**Note:** If you see an error in the Deployment name field, then remove the space before **text-embedding-ada**.

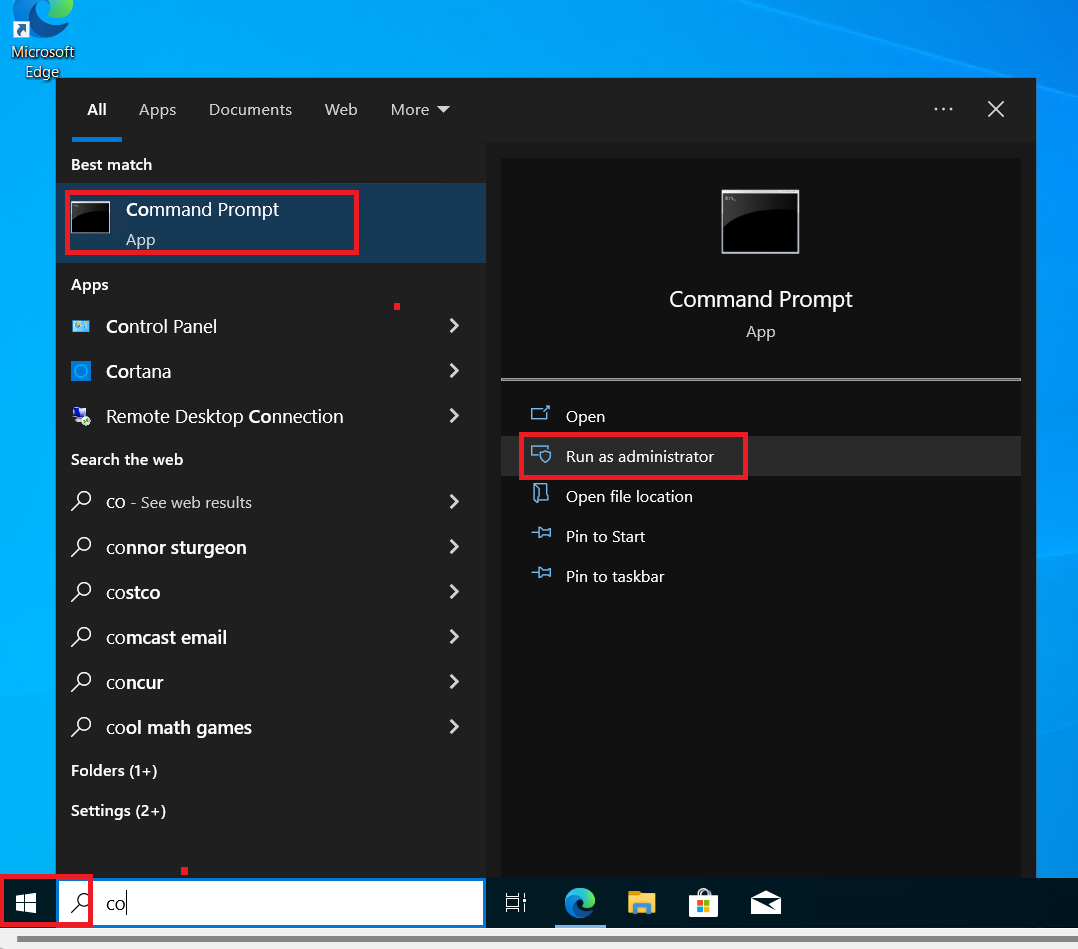
1. You will see a notification – **Successfully Created deployment** when the deployment is succeeded. (You can also view the notification by clicking on the bell icon beside **Azure AI | Azure AI Studio**).



**Important:** We strongly recommend using **text-embedding-ada-002 (Version 2)**. This model/version provides parity with OpenAI's text-embedding-ada-002. To learn more about the improvements offered by this model, please refer to OpenAI's blog post. Even if you are currently using Version 1, you should migrate to Version 2 to take advantage of the latest weights/updated token limit. Version 1 and Version 2 are not interchangeable, so document embedding and document search must be done using the same version of the model.

**Task 2: Environment variables**

1. Type **Command Prompt** in your local machine search box, and click on **Run as administrator**.



1. On **Do you allow this app to make changes on your device** dialog box, click on the **Yes** button.

A screenshot of a computer error

Description automatically generated

**Important Note**: You need to change the current directory to the **Labfiles** directory (The command used to move back to the previous directory is **cd .. [space after cd then two dots],** the command used to move to the next directory is **cd <\name of the directory>)**

1. In the **Command Prompt**, go to **Program files** directory. Set the environment variables by running the following commands.

**Note:** Update the Key value and Endpoint with the values that you have saved on your notepad in the in **Lab #4**

Copy

**setx AZURE\_OPENAI\_API\_KEY "REPLACE\_WITH\_YOUR\_KEY\_VALUE\_HERE"**

(here in this lab, we have used the Key1 that you have saved in **Lab #4 setx AZURE\_OPENAI\_API\_KEY "97baXXXXXXXXXXXXXXXXXXXXXX4f94"**)

Copy

**setx AZURE\_OPENAI\_ENDPOINT "REPLACE\_WITH\_YOUR\_ENDPOINT\_HERE"**

(here in this lab, after replacing with endpoint that you have saved in **Lab #4**, the command will be: **setx AZURE\_OPENAI\_ENDPOINT "https://azure-open-test21.openai.azure.com/"**)

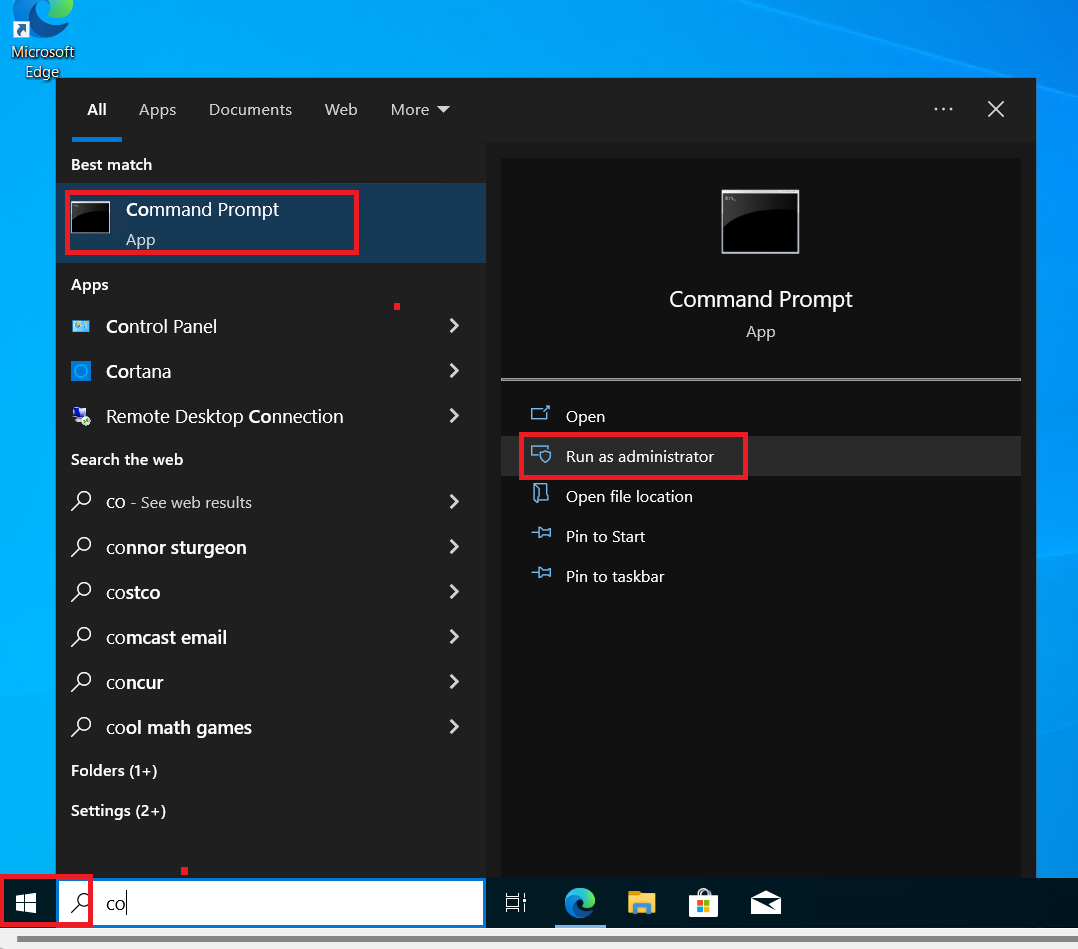
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **Close** the command prompt.
2. Then, type Command Prompt again in the window search box and click on **Run as administrator**. In the dialog box - **Do you want to allow this app to make changes to your device?**, click on the **Yes** button.



1. Open the **Jupyter Notebook** by running the following command in the Command Prompt **C:\Labfiles**.

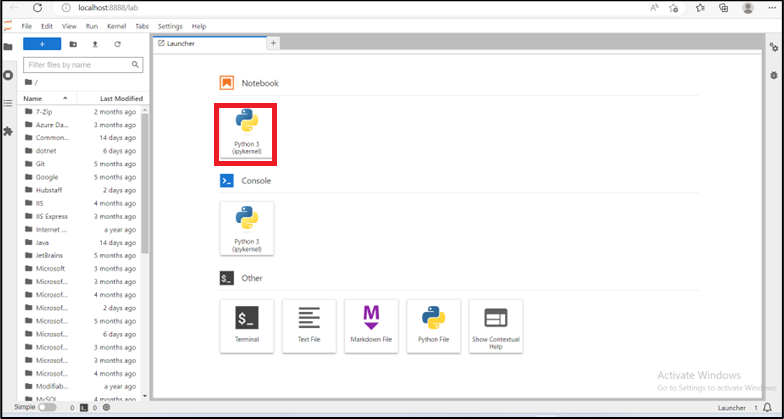
Copy

**jupyter-lab**

A screenshot of a computer

Description automatically generated

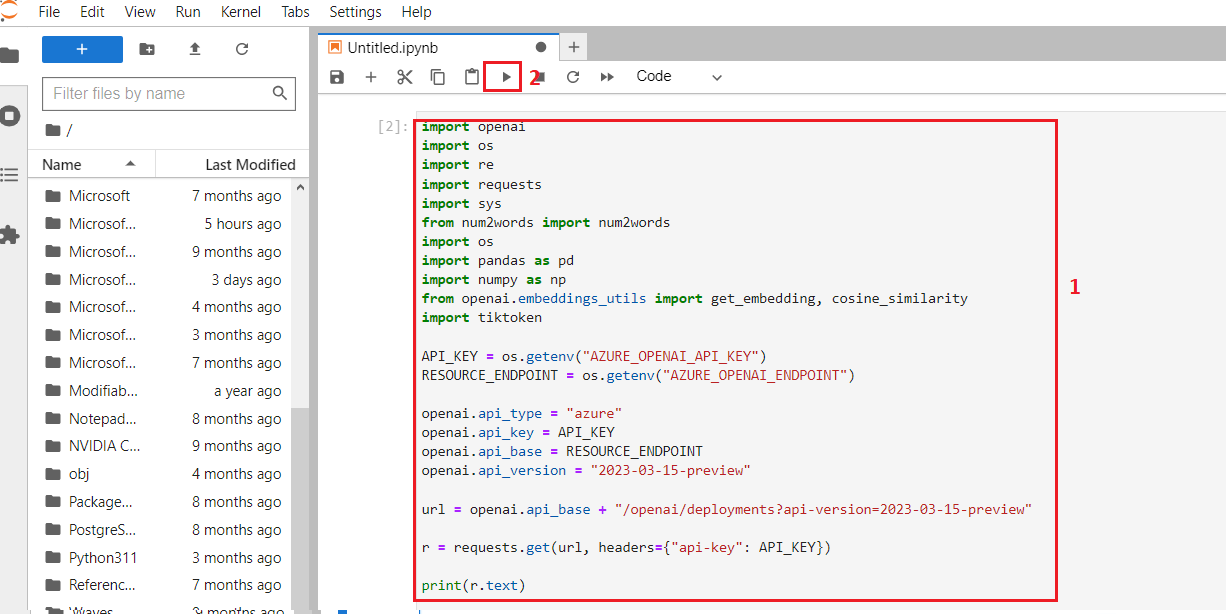
1. Under the **Jupyter Notebook**, click on **Python 3(ipykernel**).



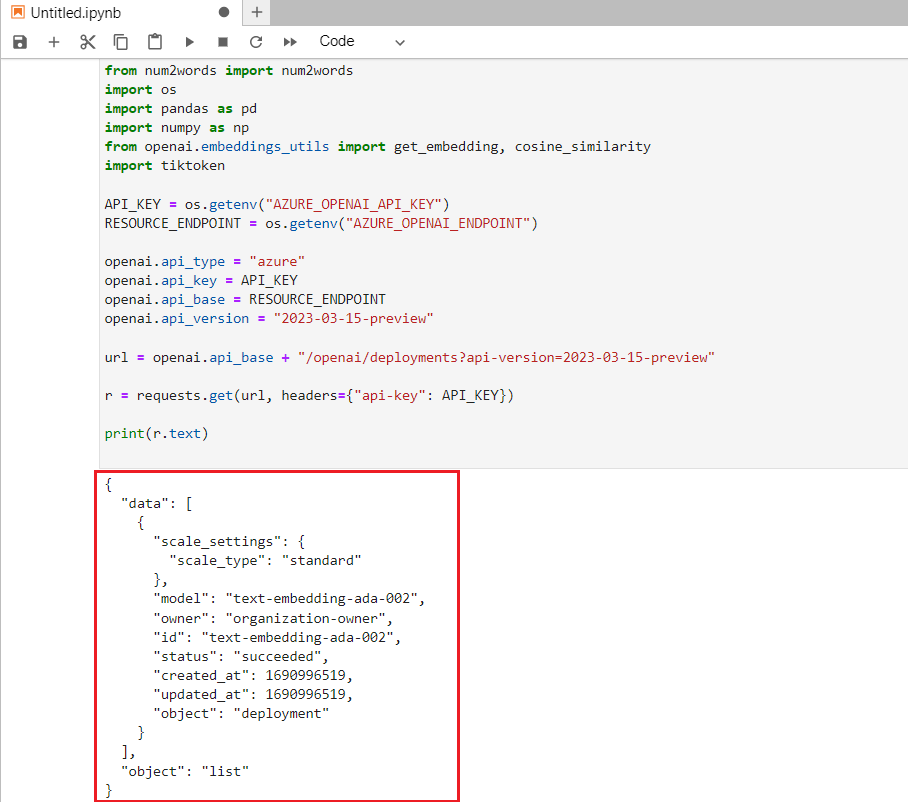
**Task 3: Import libraries and list models**

1. Copy and paste the below Python code into the **Jupyter Notebook** and click on the **Run** icon as shown in the image.
2. import openai
3. import os
4. import re
5. import requests
6. import sys
7. from num2words import num2words
8. import os
9. import pandas as pd
10. import numpy as np
11. from openai.embeddings\_utils import get\_embedding, cosine\_similarity
12. import tiktoken
13. API\_KEY = os.getenv("AZURE\_OPENAI\_API\_KEY")
14. RESOURCE\_ENDPOINT = os.getenv("AZURE\_OPENAI\_ENDPOINT")
15. openai.api\_type = "azure"
16. openai.api\_key = API\_KEY
17. openai.api\_base = RESOURCE\_ENDPOINT
18. openai.api\_version = "2023-03-15-preview"
19. url = openai.api\_base + "/openai/deployments?api-version=2023-03-15-preview"
20. r = requests.get(url, headers={"api-key": API\_KEY})

print(r.text)



**Output in Jupyter Notebook:**



{

"data": [

{

"scale\_settings": {

"scale\_type": "standard"

},

"model": "text-embedding-ada-002",

"owner": "organization-owner",

"id": "text-embedding-ada-002",

"status": "succeeded",

"created\_at": 1690996519,

"updated\_at": 1690996519,

"object": "deployment"

}

],

"object": "list"

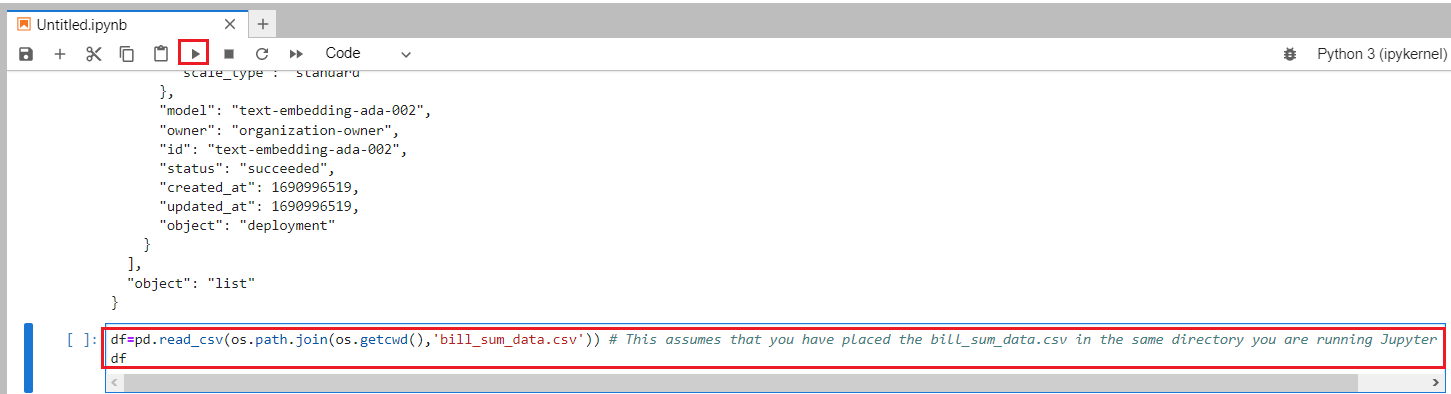
}

1. The output of this command will vary based on the number and type of models you've deployed. In this case, we need to confirm that we have an entry for **text-embedding-ada-002**.
2. Now, we need to read our **CSV** file and create Pandas DataFrame. After the initial DataFrame is created, we can view the contents of the table by running **df**.
3. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**Copy**

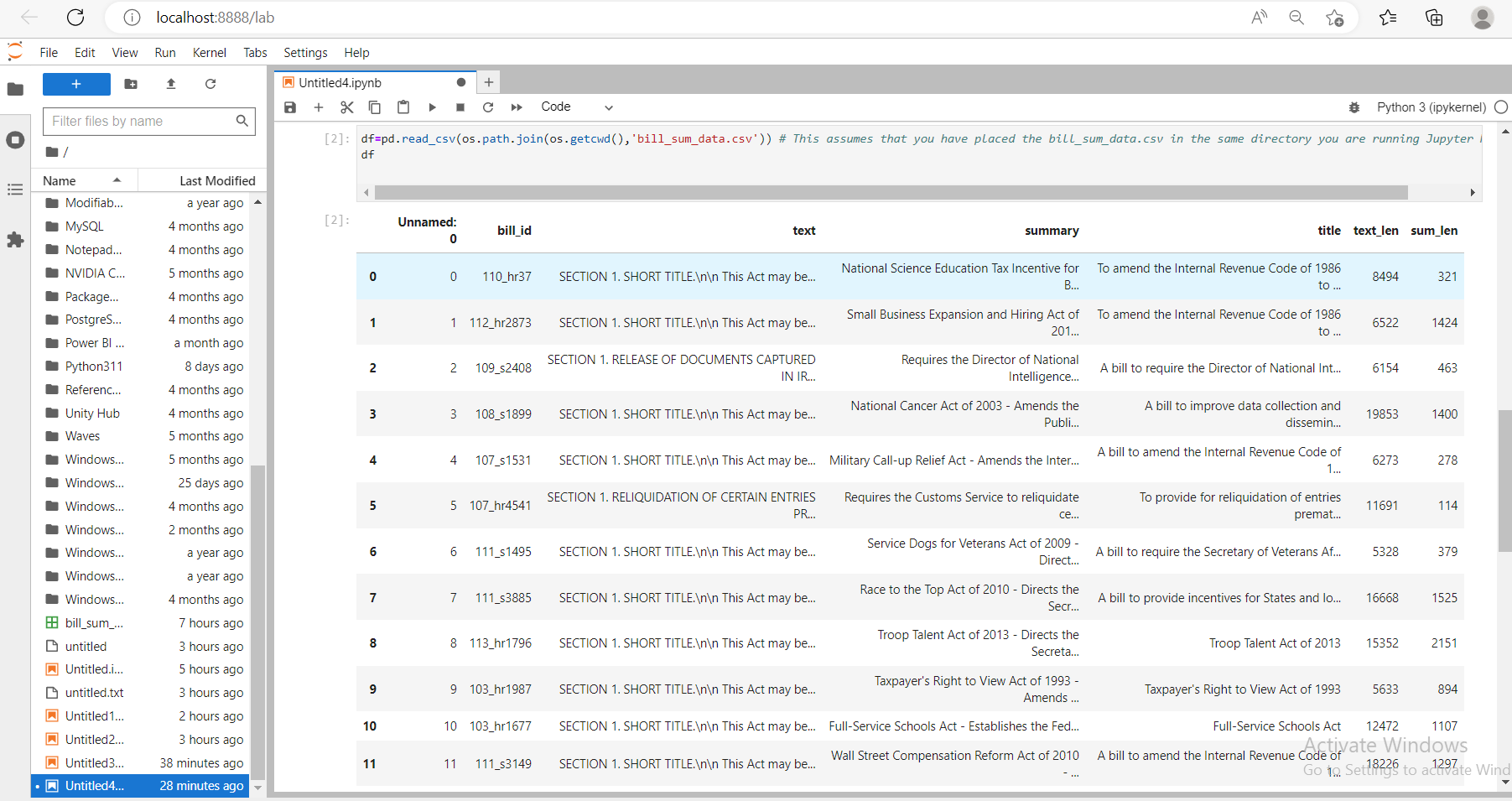
df=pd.read\_csv(os.path.join(os.getcwd(),'bill\_sum\_data.csv')) # This assumes that you have placed the bill\_sum\_data.csv in the same directory you are running Jupyter Notebooks

df



**Output in Jupyter Notebook:**



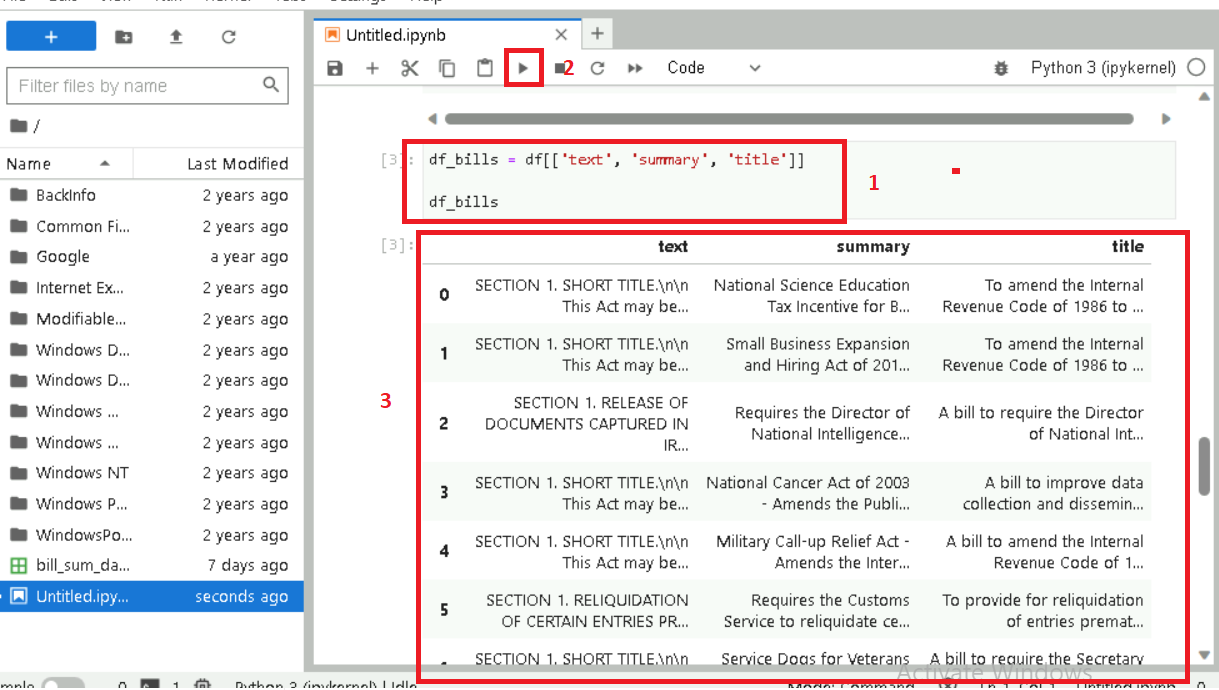


1. The initial table has more columns than we needed. We’ll create a new smaller DataFrame called df\_bills, which will contain only the columns for text, summary, and title.
2. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**PythonCopy**

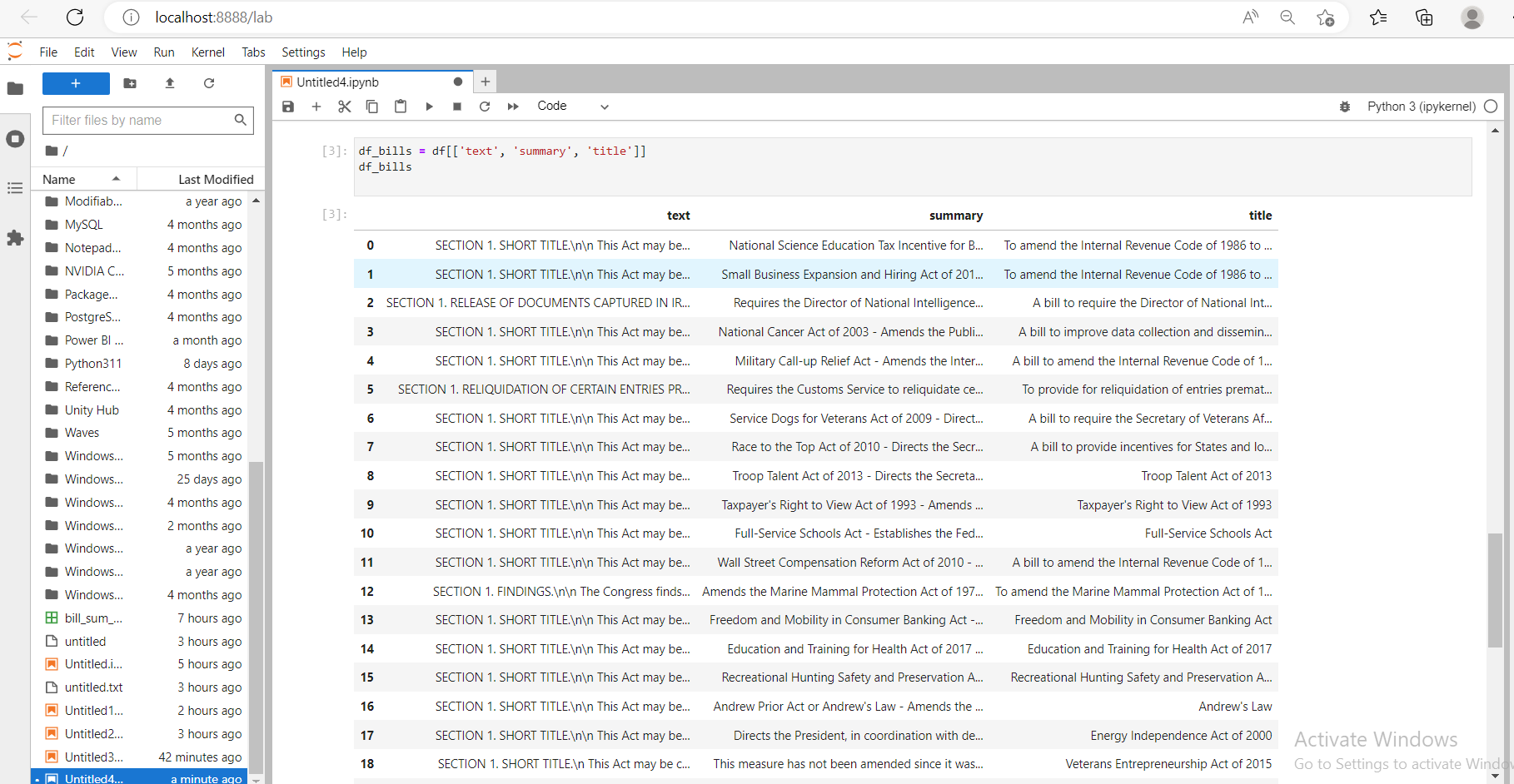
df\_bills = df[['text', 'summary', 'title']]

df\_bills



**Output in Jupyter Notebook:**





1. Perform some light data cleaning by removing redundant whitespace and cleaning up the punctuation to prepare the data for tokenization.
2. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**PythonCopy**

pd.options.mode.chained\_assignment = None #https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#evaluation-order-matters

# s is input text

def normalize\_text(s, sep\_token = " \n "):

s = re.sub(r'\s+', ' ', s).strip()

s = re.sub(r". ,","",s)

# remove all instances of multiple spaces

s = s.replace("..",".")

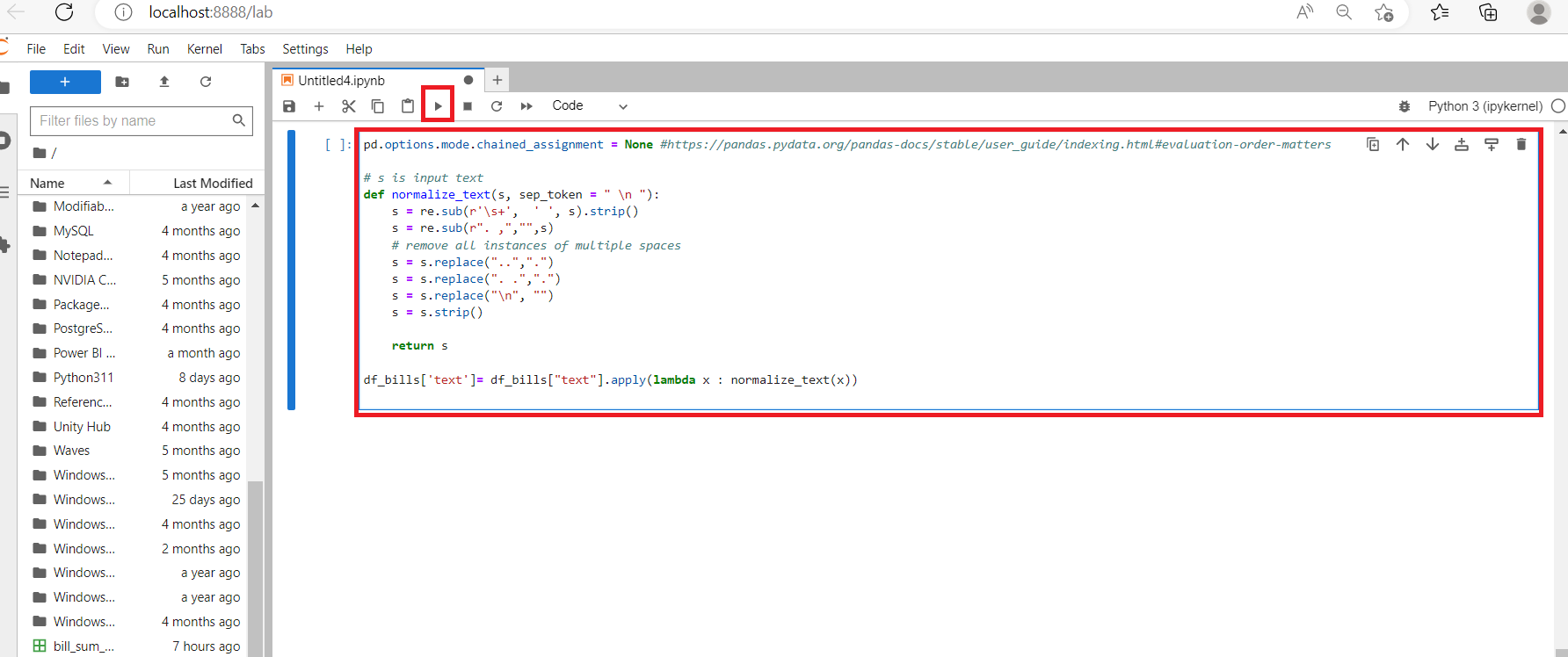
s = s.replace(". .",".")

s = s.replace("\n", "")

s = s.strip()

return s

df\_bills['text']= df\_bills["text"].apply(lambda x : normalize\_text(x))



1. Now, we need to remove any bills that are too long for the token limit (8192 tokens).
2. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

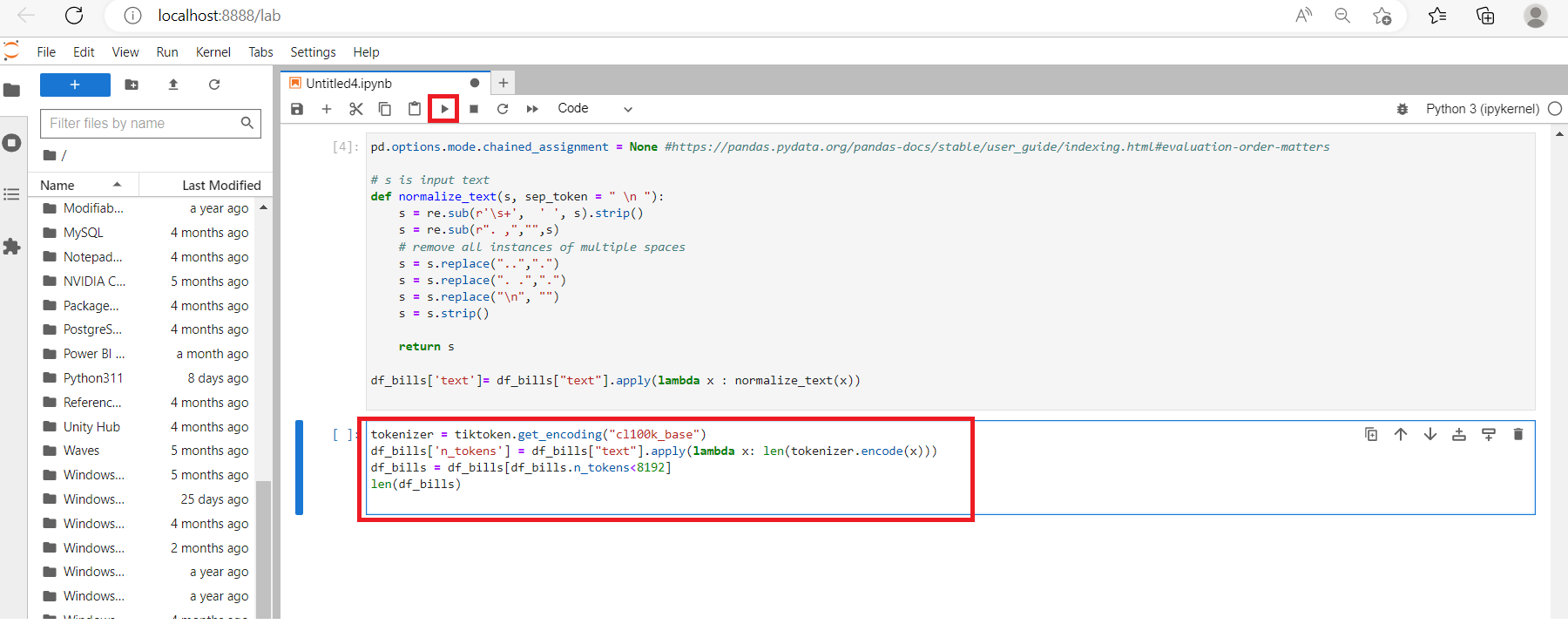
**PythonCopy**

tokenizer = tiktoken.get\_encoding("cl100k\_base")

df\_bills['n\_tokens'] = df\_bills["text"].apply(lambda x: len(tokenizer.encode(x)))

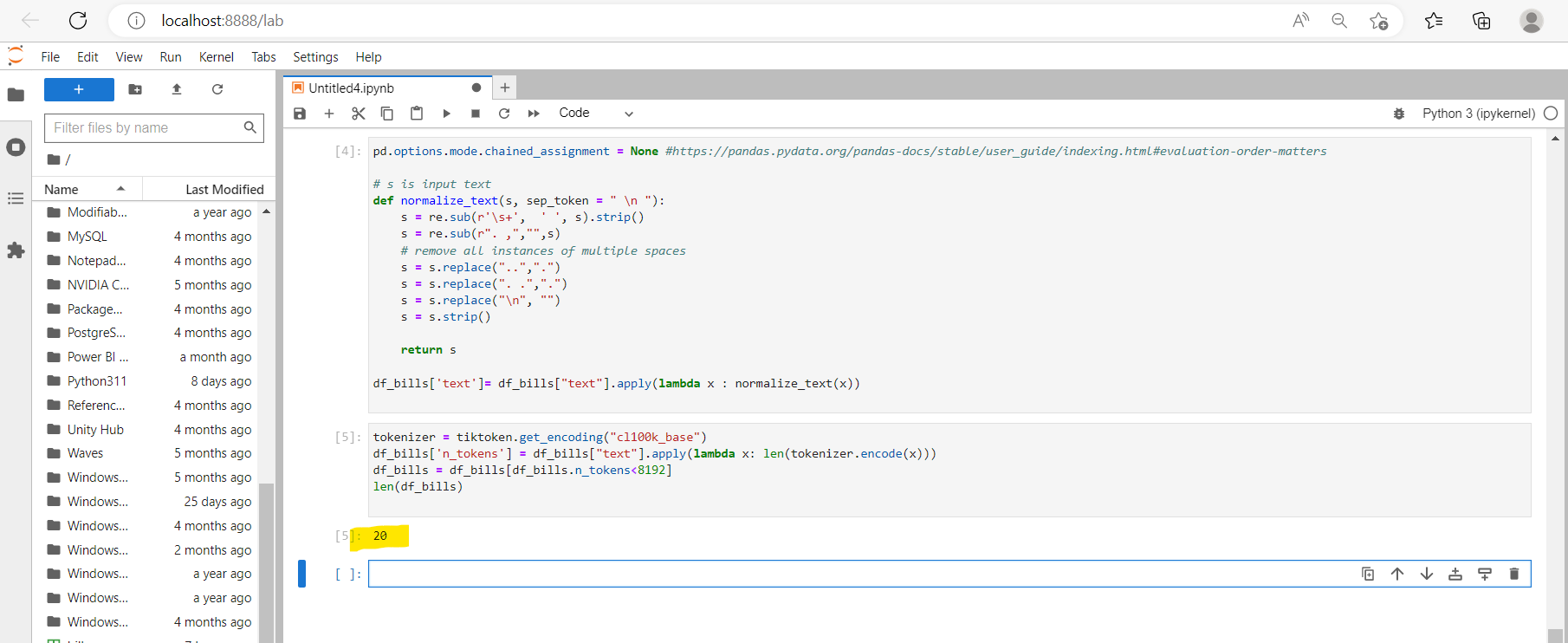
df\_bills = df\_bills[df\_bills.n\_tokens<8192]

len(df\_bills)



**Output in Jupyter Notebook:**

20

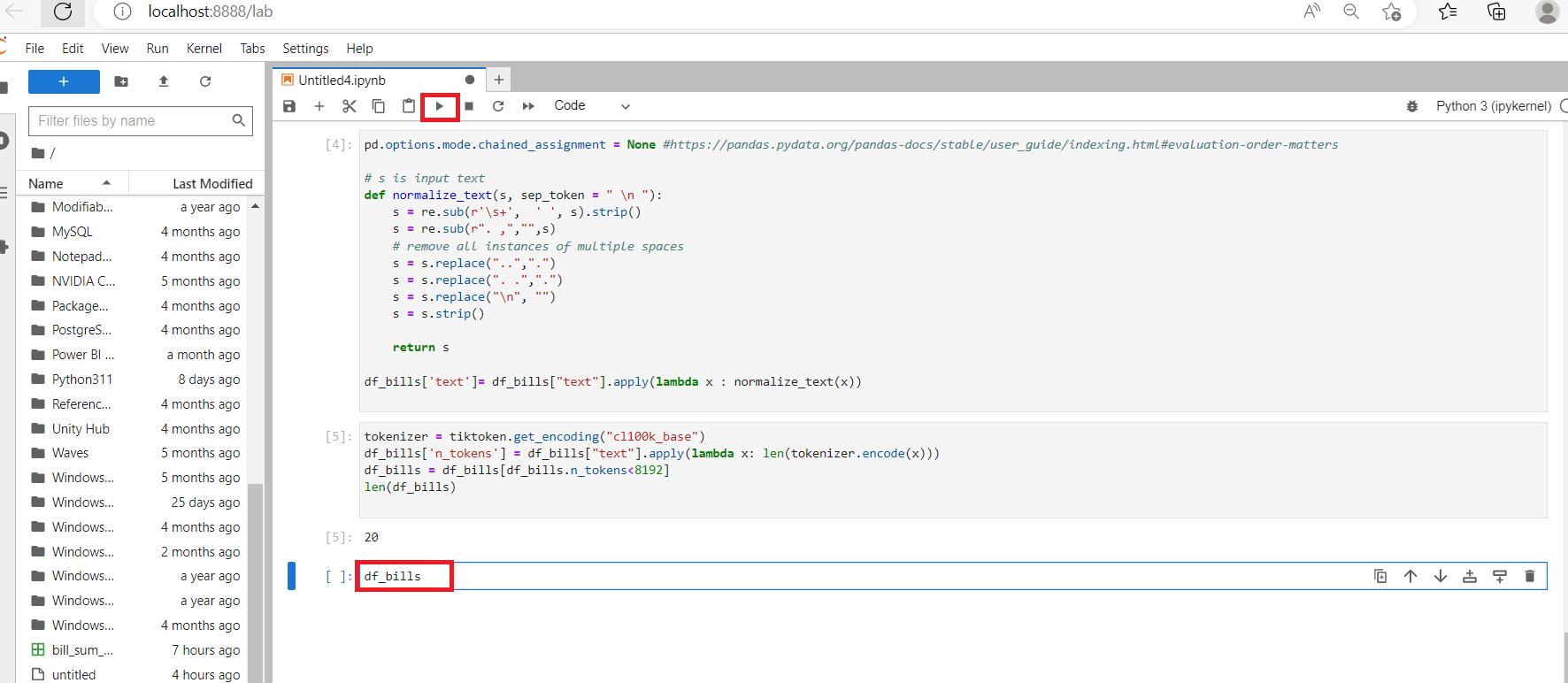


**Note:** In this case, all bills are under the embedding model input token limit, but you can use the technique above to remove entries that would otherwise cause embedding to fail. When faced with content that exceeds the embedding limit, you can also chunk the content into smaller pieces and then embed those one at a time.

1. Once again, examine **df\_bills**.
2. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**PythonCopy**

df\_bills



**Output in Jupyter Notebook:**



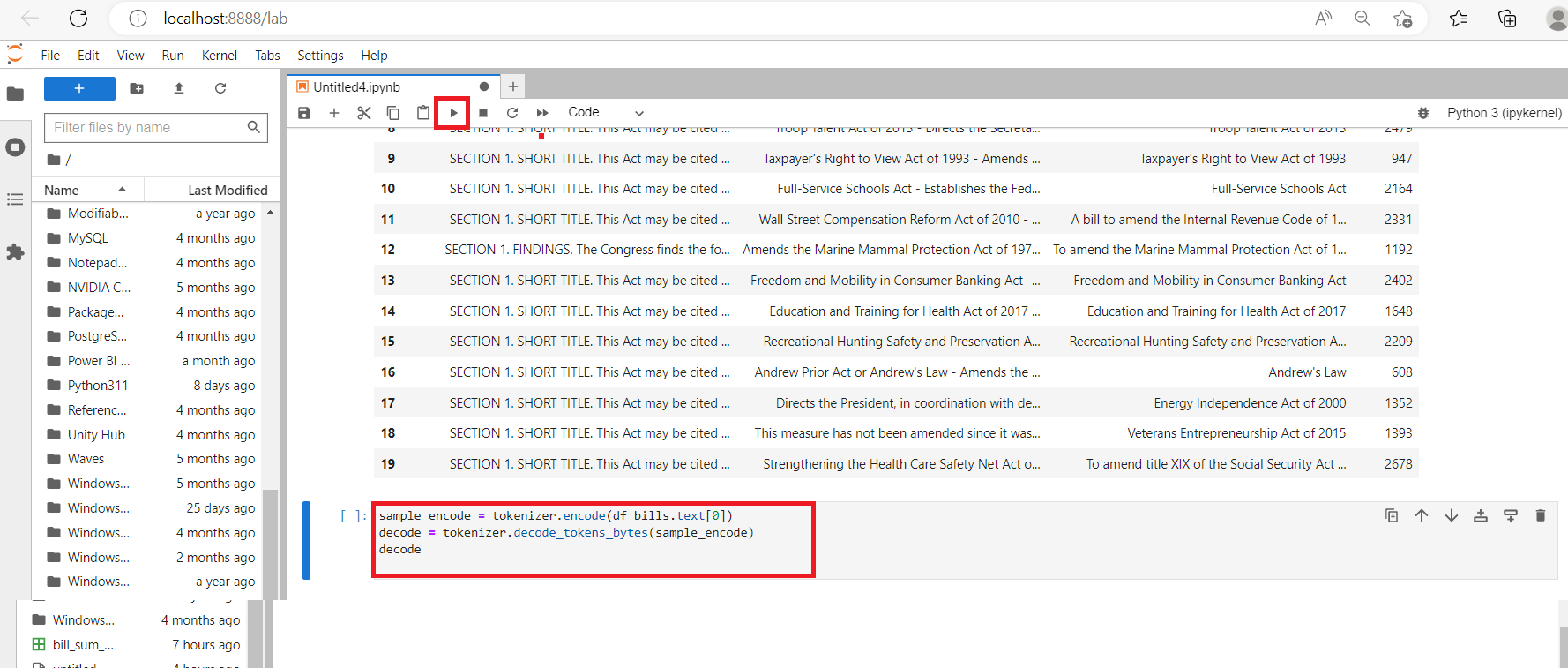
1. To understand the n\_tokens column and to know how text is tokenized, copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**PythonCopy**

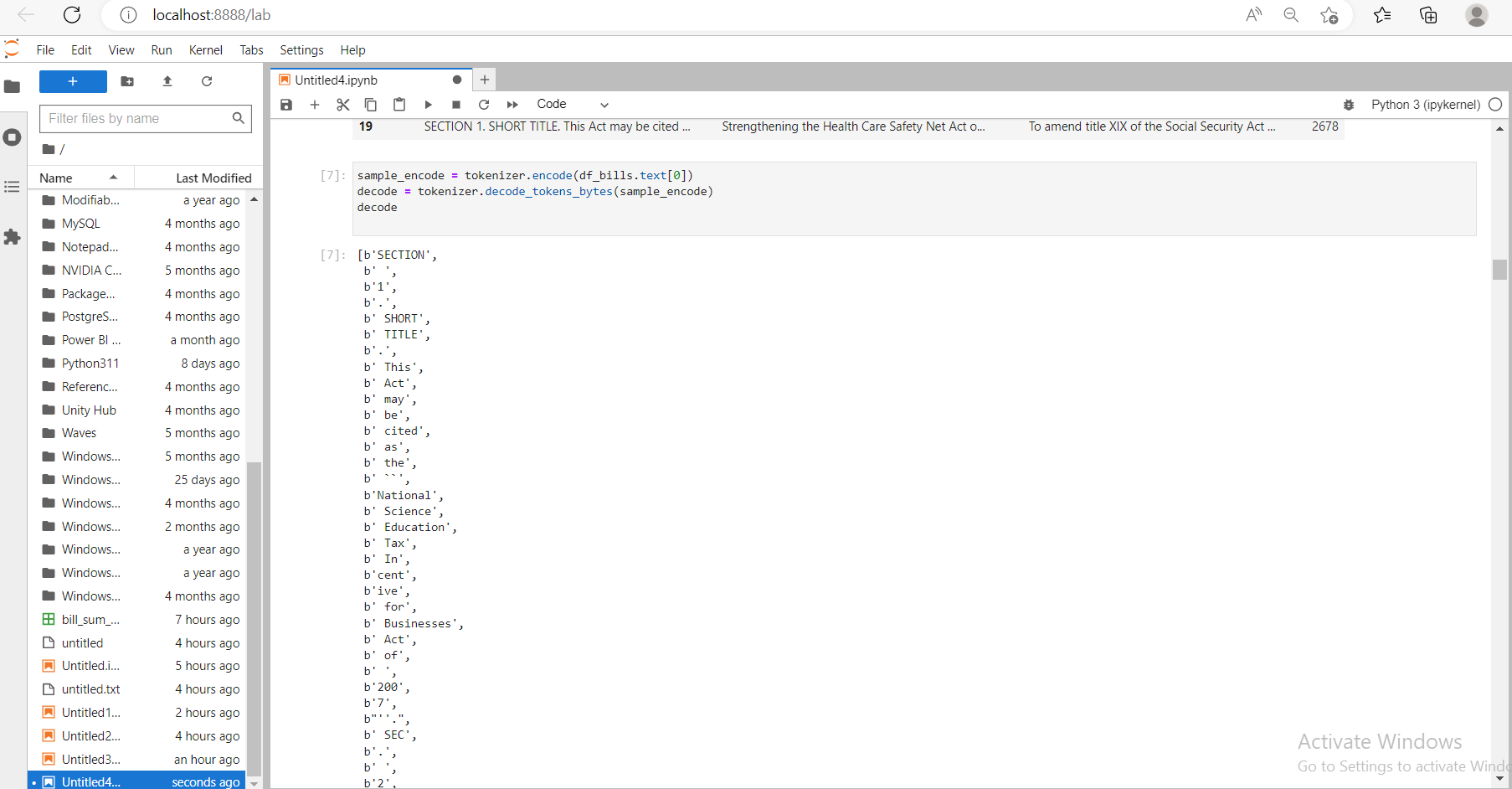
sample\_encode = tokenizer.encode(df\_bills.text[0])

decode = tokenizer.decode\_tokens\_bytes(sample\_encode)

decode



1. For our docs, we're intentionally truncating the output, but running this command in your environment will return the full text from index zero tokenized into chunks. You can see that in some cases an entire word is represented with a single token whereas in others parts of words are split across multiple tokens.



**Output in Jupyter Notebook:**

[b'SECTION',

b' ',

b'1',

b'.',

b' SHORT',

b' TITLE',

b'.',

b' This',

b' Act',

b' may',

b' be',

b' cited',

b' as',

b' the',

b' ``',

b'National',

b' Science',

b' Education',

b' Tax',

b' In',

b'cent',

b'ive',

b' for',

b' Businesses',

b' Act',

b' of',

b' ',

b'200',

b'7',

b"''.",

b' SEC',

b'.',

b' ',

b'2',

b'.',

b' C',

b'RED',

b'ITS',

b' FOR',

b' CERT',

b'AIN',

b' CONTRIBUT',

b'IONS',

b' BEN',

b'EF',

b'IT',

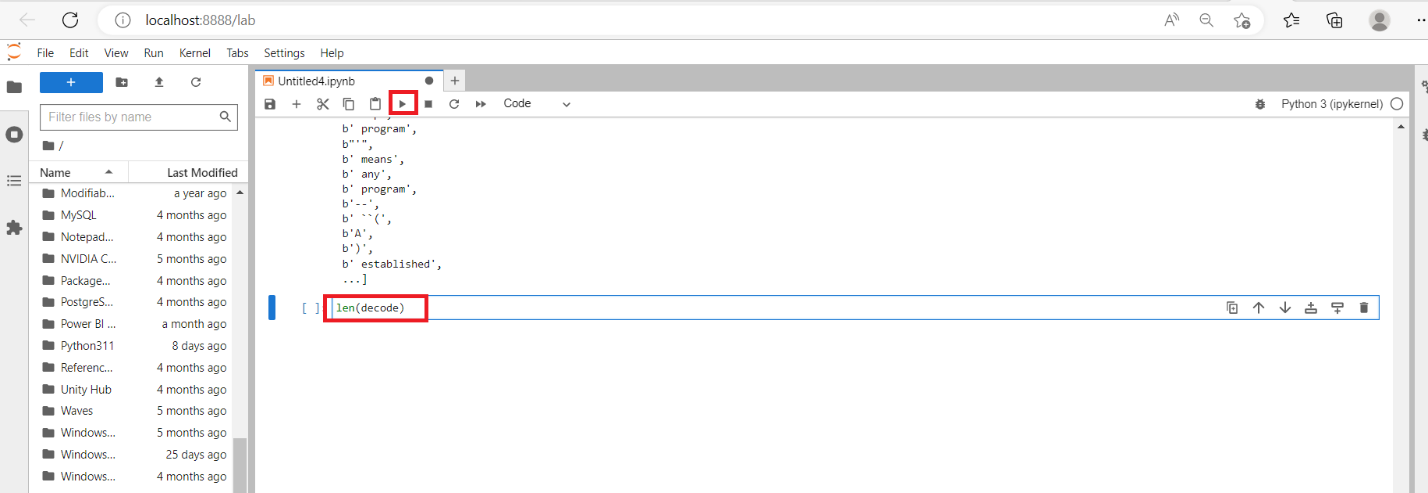
b'ING',

b' SC',

1. If you check the length of the decoded variable, you observe that it matches the first number in the n\_tokens column.
2. Copy and paste the below Python code into the JupyterNotebook and click on **Run**.

**PythonCopy**

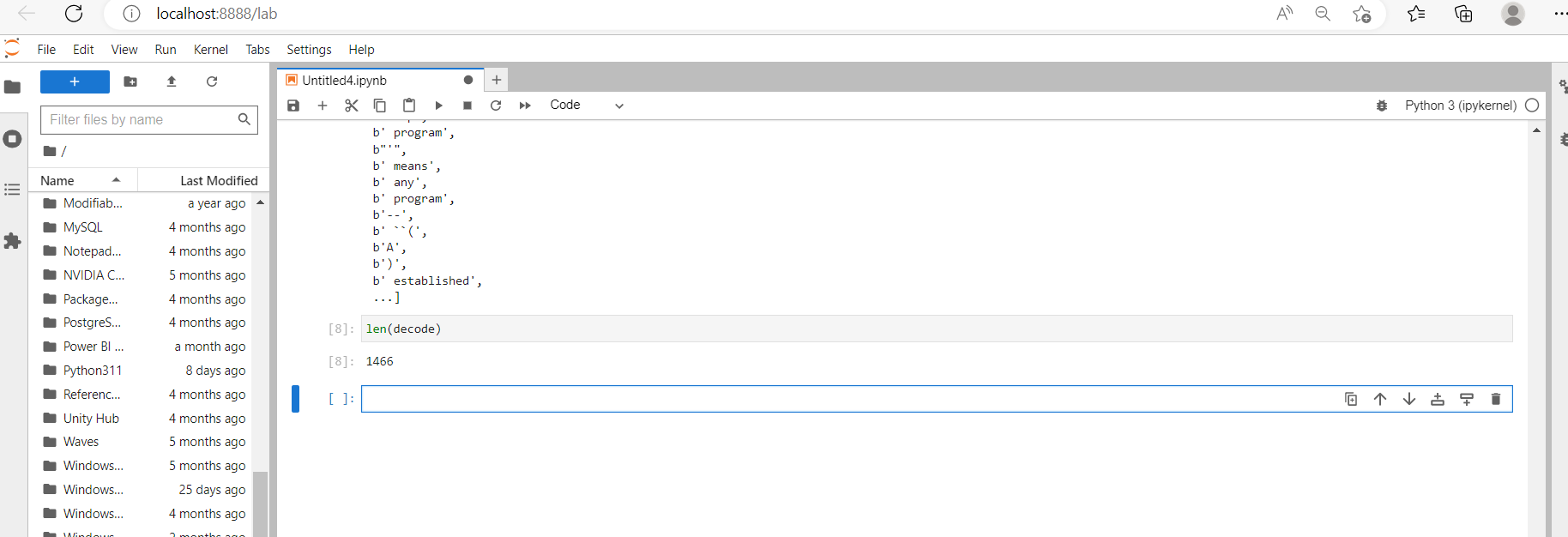
len(decode)



**Output in Jupyter Notebook:**

**Windows Command PromptCopy**

1466



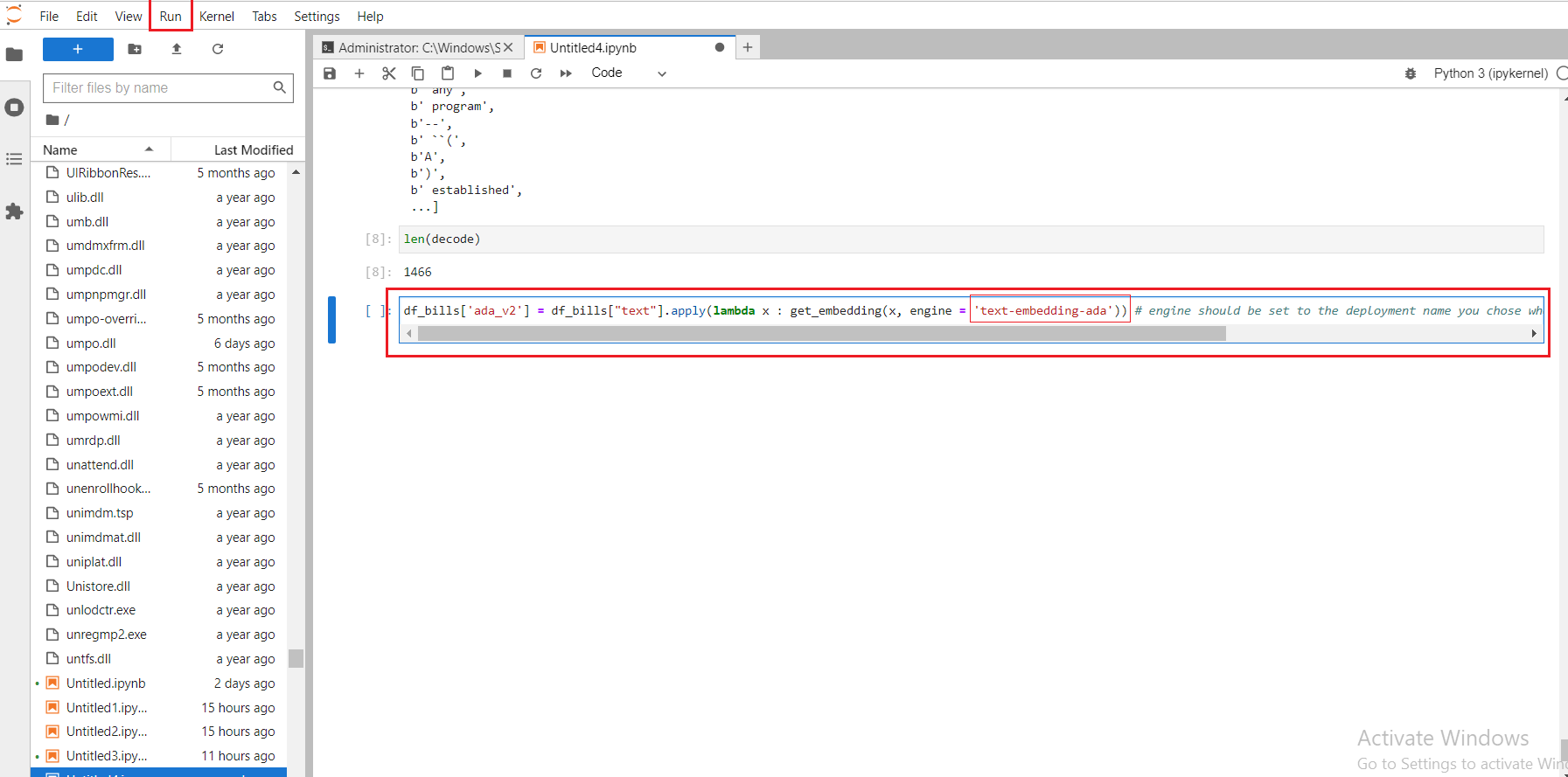
Now that we understand more about how tokenization works, we can move on to embedding. It is important to note that we haven't actually tokenized the documents yet. The n\_tokens column is simply a way of making sure none of the data we pass to the model for tokenization and embedding exceeds the input token limit of 8,192. When we pass the documents to the embeddings model, it will break the documents into tokens similar (though not necessarily identical) to the examples above and then convert the tokens to a series of floating-point numbers that will be accessible via vector search. These embeddings can be stored locally or in an Azure Database. As a result, each bill will have its own corresponding embedding vector in the new ada\_v2 column on the right side of the DataFrame.

1. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**

**PythonCopy**

df\_bills['ada\_v2'] = df\_bills["text"].apply(lambda x : get\_embedding(x, engine = 'text-embedding-ada')) # engine should be set to the deployment name you chose when you deployed the text-embedding-ada-002 (Version 2) model

(here, in this lab, we entered the **Model deployment name** as **text-embedding-ada;** Exercise #2, Task 1)



1. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

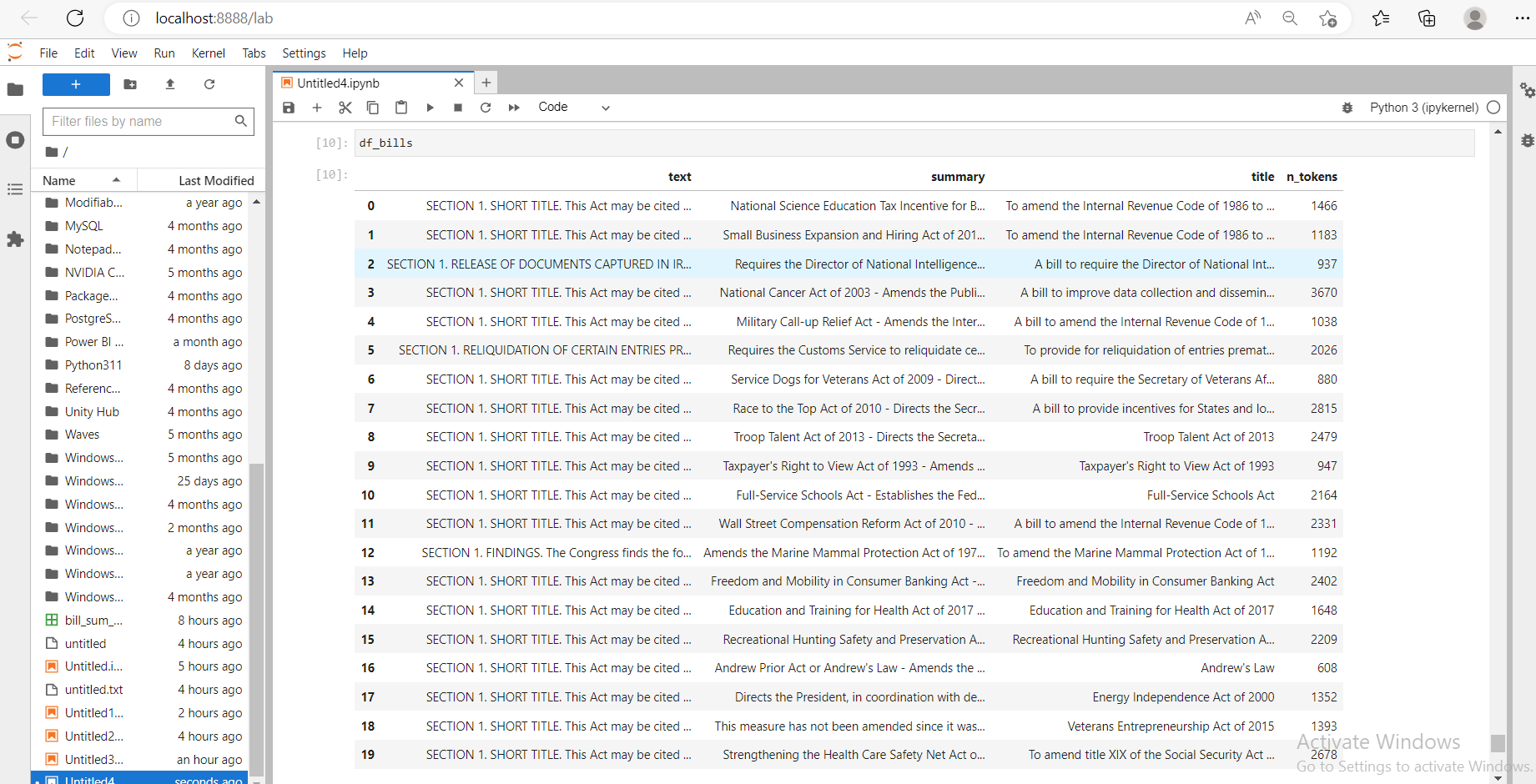
**PythonCopy**

df\_bills

A screenshot of a computer

Description automatically generated

**Output in Jupyter Notebook:**



1. As we run the search code block below, we'll embed the search query *"Can I get information on cable company tax revenue?"* with the same **text-embedding-ada-002 (Version 2)** model.
2. Next, find the closest bill embedding to the newly embedded text from our query ranked by cosine similarity.
3. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**PythonCopy**

# search through the reviews for a specific product

def search\_docs(df, user\_query, top\_n=3, to\_print=True):

embedding = get\_embedding(

user\_query,

engine="text-embedding-ada" # engine should be set to the deployment name you chose when you deployed the text-embedding-ada-002 (Version 2) model

)

df["similarities"] = df.ada\_v2.apply(lambda x: cosine\_similarity(x, embedding))

res = (

df.sort\_values("similarities", ascending=False)

.head(top\_n)

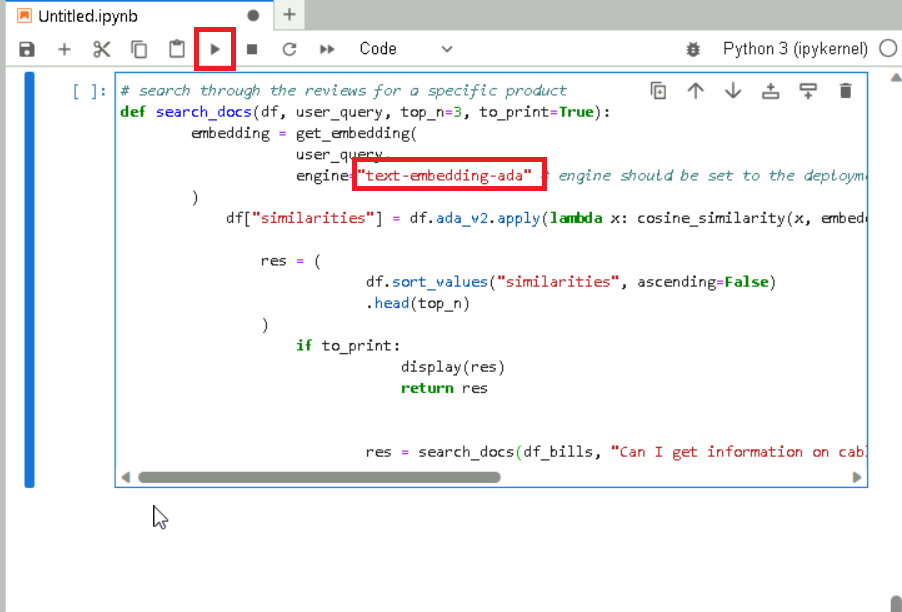
)

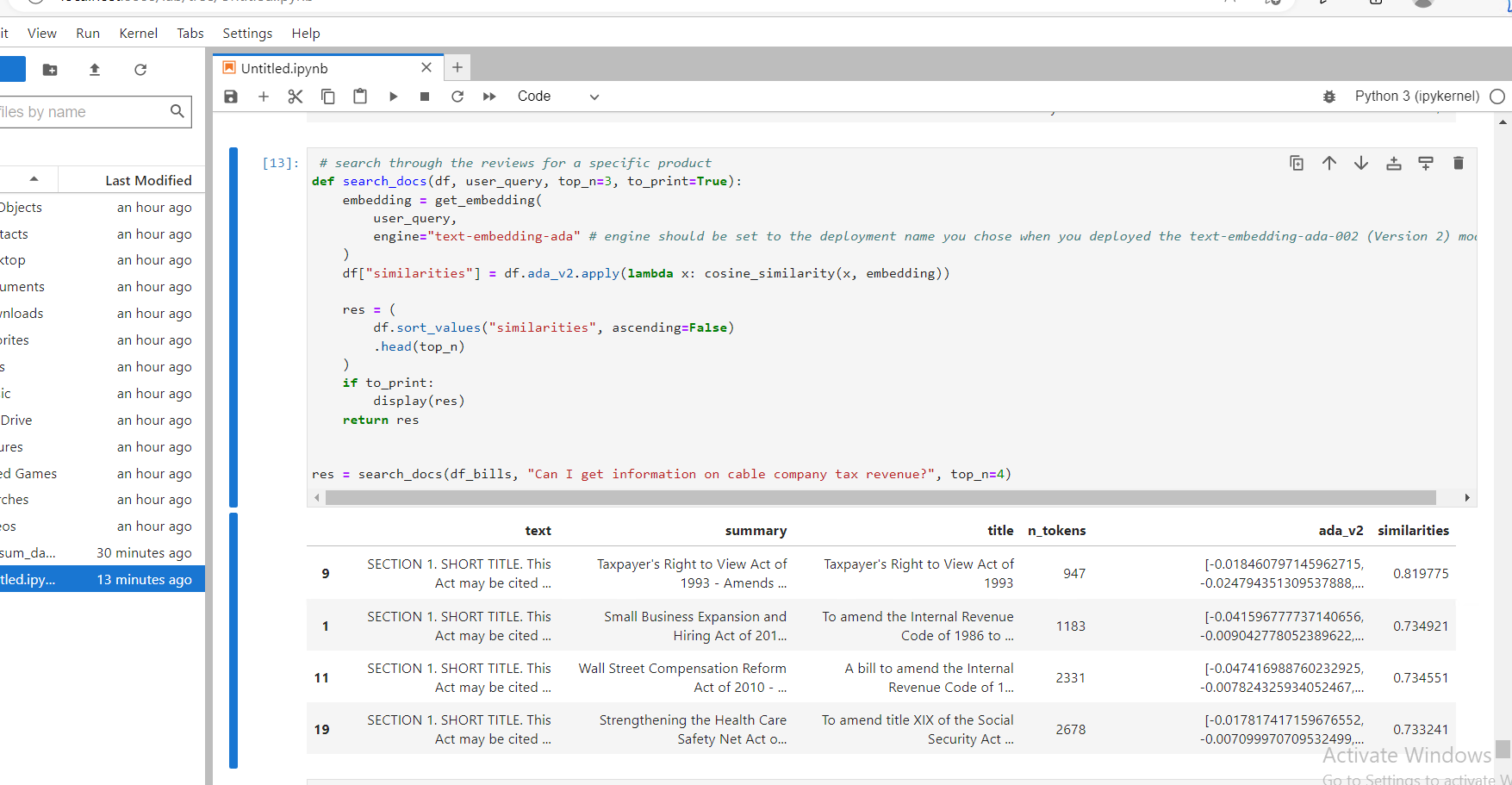
if to\_print:

display(res)

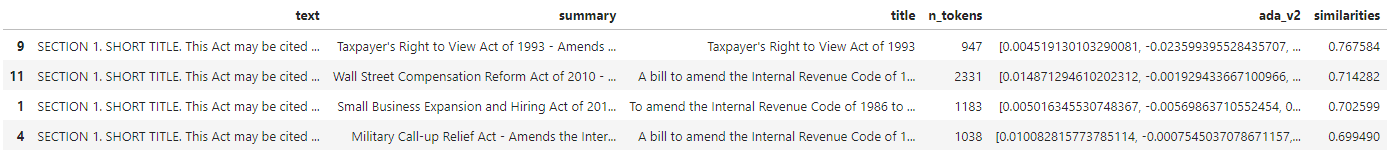
return res

res = search\_docs(df\_bills, "Can I get information on cable company tax revenue?", top\_n=4)





**Output in Jupyter Notebook**:

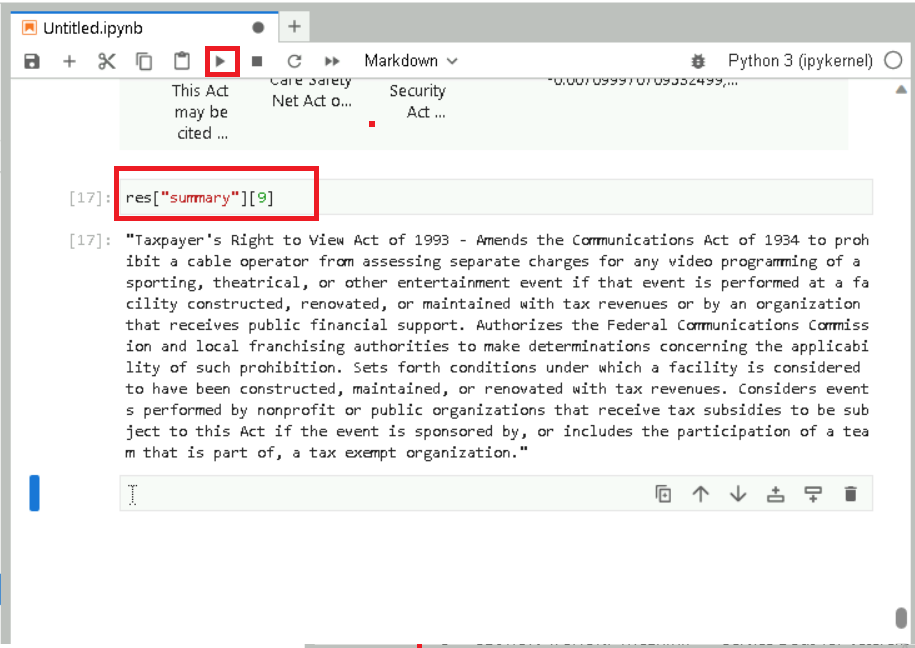


Finally, you'll see the top result from document searches based on user queries against the entire knowledge base. This returns the top result of the "Taxpayer's Right to View Act of 1993". This document has a cosine similarity score of 0.36 between the query and the document

1. Copy and paste the below Python code into the Jupyter Notebook and click on **Run**.

**PythonCopy**

res["summary"][9]



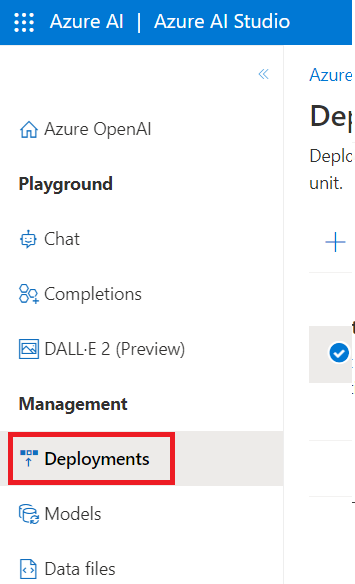
**Output:**

"Taxpayer's Right to View Act of 1993 - Amends the Communications Act of 1934 to prohibit a cable operator from assessing separate charges for any video programming of a sporting, theatrical, or other entertainment event if that event is performed at a facility constructed, renovated, or maintained with tax revenues or by an organization that receives public financial support. Authorizes the Federal Communications Commission and local franchising authorities to make determinations concerning the applicability of such prohibition. Sets forth conditions under which a facility is considered to have been constructed, maintained, or renovated with tax revenues. Considers events performed by nonprofit or public organizations that receive tax subsidies to be subject to this Act if the event is sponsored by, or includes the participation of a team that is part of, a tax exempt organization."

1. Using this approach, you can use embeddings as a search mechanism across documents in a knowledge base. The user can then take the top search result and use it for their downstream task, which prompted their initial query.

**Task 4: Delete the deployed models**

1. In Azure OpenAI Studio, on the left pane, under the **Management** section, click on **Deployments**.

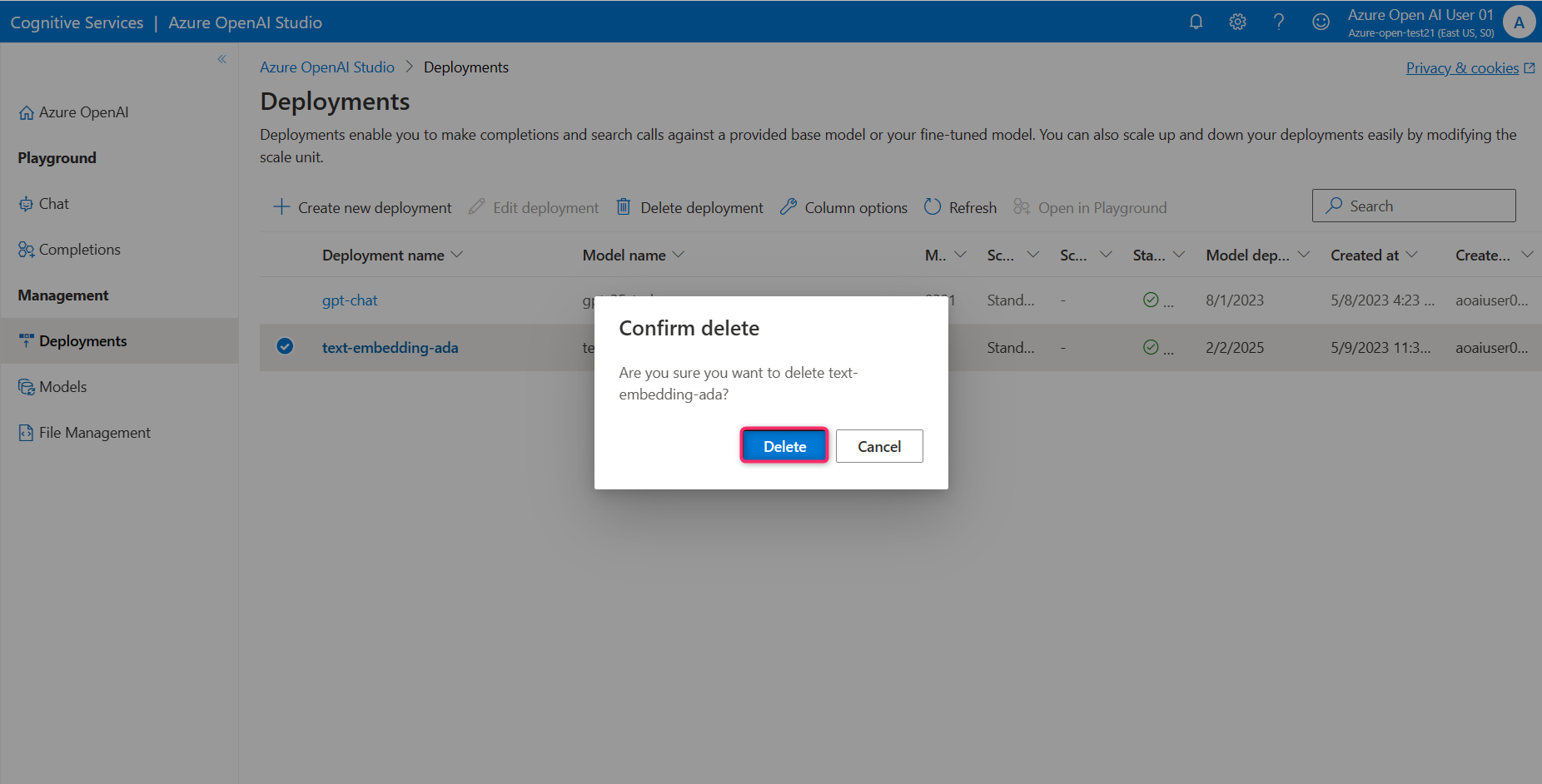


1. Select **text-embedding-ada** deployment name and click on **Delete deployment**.

A screenshot of a computer

Description automatically generated

1. In the **Confirm delete** dialog box, click on the **Delete** button. You will see the notification – **Successfully Deleted deployment** (In case, you did not see the notification, then click on the bell icon beside **Cognitive Services | Azure OpenAI Studio**).



**Summary** In this lab, you’ve deployed text-embedding-ada-002 model, configured bill\_sum\_data.csv and environmental variable, imported the libraries and list models, cleaned the data and understood how tokenization works, performed document search against the entire knowledge based. Using this approach, you can use embeddings as a search mechanism across documents in a knowledge base. In the last task, you’ve deleted the deployed model to effectively and efficiently manage the OpenAI resource.

**Important Note: Please do not delete the Resource group and Azure OpenAI Service (Azure-openai-testXX). The same Resource group and AOAI service will be used throughout all the labs.**