Quick tip: Review the prerequisites before you run the lab

End Lab 00:10:03

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Open Google Cloud Console

GCP Username
student-00-c9611db2ad4d

GCP Password
YjTCixVPFod6

GCP Project ID
qwiklabs-gcp-03-c48548f1

Custom Embeddings with AI Applications

Lab 1 hour No cost Intermediate

★★★★☆

This lab may incorporate AI tools to support your learning.

Previous Next

Overview

This lab serves as a guide to utilizing text-embedding-004 in Vertex AI for obtaining text embeddings. It walks you through the process of converting these embeddings into the required format for Vertex AI Search. Additionally, the lab provides detailed instructions

Objective

This lab showcases steps to:

- Retrieve text embeddings using the Vertex AI API
- Transform embeddings into the format required by Vertex AI Search
- Create a search application with custom embeddings

Setup and requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

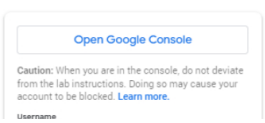
To complete this lab, you need:

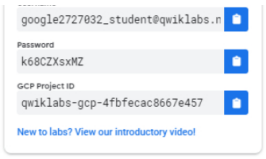
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

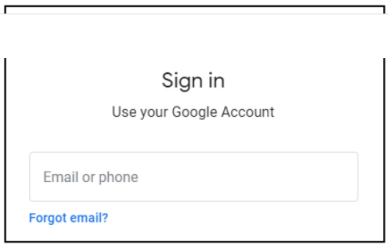
Note: If you are using a Pixelbook, open an Incognito window to run this lab.

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



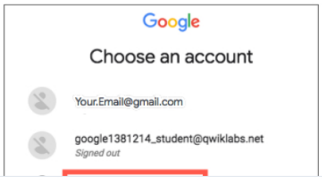


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

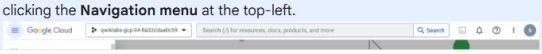
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.

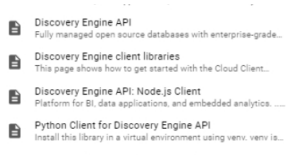


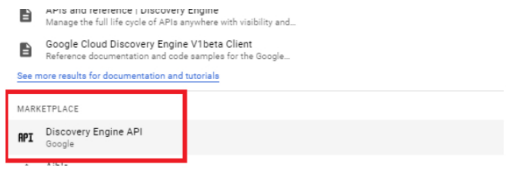
Task 1. Enable API

In this section, let's enable the discovery engine API.

To enable the Discovery Engine API, follow these steps:

1. Type **Discovery Engine API** into the top search bar of the Google Cloud Console, choose the result as shown in the following image.





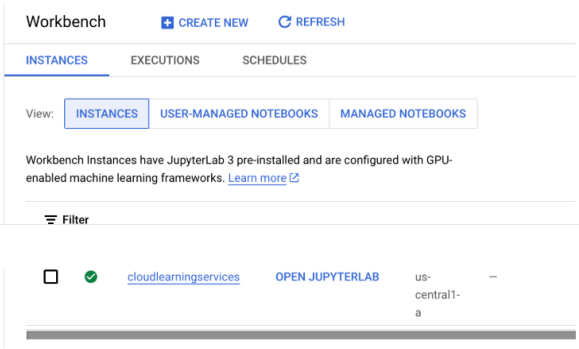
2. Click **Enable**.

Notebook and Install Packages

1. In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI > Workbench**.

2. Find the **cloudlearningservices** instance and click on the **Open JupyterLab** button.

The JupyterLab interface for your Workbench instance will open in a new browser tab.



3. On the **Launcher**, under **Notebook**, click on **Python 3** to open a new python notebook.

4. Install required packages by the running the following command in the first cell of the notebook. Either click the play ▶ button at the top or click **SHIFT+ENTER** keys on your keyboard to execute the cell.

```
%pip install -q --upgrade --user google-cloud-aiplatform google-cloud-discoveryengine google-cloud-storage google-cloud-bigquery[pandas] google-cloud-bigquery-storage pandas ipywidgets
```

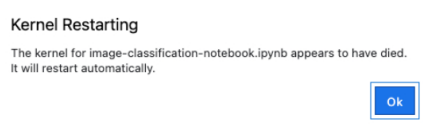
```
%load_ext google.cloud.bigquery
```

5. To use the newly installed packages in this Jupyter runtime, it is recommended to

```
# Automatically restart kernel after installs so that your environment can access the new packages
import IPython

app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

Output:



After the restart is complete, click **Ok** on the prompt to continute.

6. Run the following code snippet in the next cell. This code imports various python

Engine.

```
from typing import List
```

```
import requests
import subprocess
import time

from google.api_core.client_options import ClientOptions
from google.api_core.exceptions import GoogleAPICallError
from google.cloud import bigquery
from google.cloud import discoveryengine_v1alpha as
discoveryengine
from google.cloud import storage

from tqdm import tqdm  # to show a progress bar

import vertexai
from vertexai.language_models import TextEmbeddingModel,
TextEmbeddingInput

tqdm.pandas()
```

7. Run the below code snippet in the next cell. This code sets up project information and initializes the Vertex AI SDK for a Google Cloud Platform (GCP) project.

```
# Define project information for Vertex AI
PROJECT_ID = "qwiklabs-gcp-03-c48548fb59c9"  # @param
{type:"string"}
LOCATION = "us-central1"  # @param {type:"string"}

# Initialize Vertex AI SDK
vertexai.init(project=PROJECT_ID, location=LOCATION)
```

Task 3. Create embeddings with Vertex AI

We will be using the Stack Overflow public dataset hosted on BigQuery table `bigquery-public-data.stackoverflow.posts_questions`. This is a very big dataset with 23 million rows that doesn't fit into memory. We are going to limit it to 500 rows for this lab.

In this task, we will:

- Fetch the data from BigQuery
- Concat the Title and Body, and create embeddings from the text.

1. Run the following code snippet in the next cell, that connects to Google BigQuery, executes a SQL query to retrieve information from the Stack Overflow dataset, loads the results into a Pandas DataFrame, and then performs some data manipulation.

```
bq_client = bigquery.Client(project=PROJECT_ID)
query = f"""
SELECT
  DISTINCT
  q.id,
  q.answer_count,
  q.comment_count,
  q.creation_date,
  q.last_activity_date,
  q.score,
  q.tags,
  q.view_count
FROM
  `bigquery-public-data.stackoverflow.posts_questions` AS q
WHERE
  q.score > 0
ORDER BY
  q.view_count DESC
LIMIT
  500;
"""

# Load the BQ Table into a Pandas Dataframe
df = bq_client.query(query).result().to_dataframe()

# Convert ID to String
df["id"] = df["id"].apply(str)
```

```
df.head()
```

Output:

Call the API to generate embeddings

With the Stack Overflow dataset, we will use the `title` and body columns (the question title and description) and generate embedding for it with Embeddings for Text API. The API is available under the `vertexai` package of the SDK.

From the package, import `TextEmbeddingModel` and get a model.

For more information, refer to:

- Vertex AI: Get Text Embeddings

2. Run the following code snippet in the next cell, that loads a pre-trained text embeddings model.

```
# Load the text embeddings model
model = TextEmbeddingModel.from_pretrained("text-embedding-004")
```

3. Run the following code snippet in the next cell, to define a python function named `get_embeddings_wrapper` that takes a list of texts and an optional batch size as input parameters. The function uses the previously loaded model (a text embeddings model) to obtain embeddings for the provided texts.

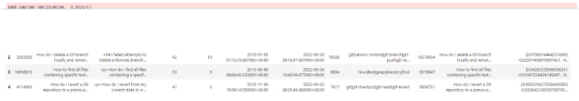
```
# Get embeddings for a list of texts
def get_embeddings_wrapper(texts, batch_size: int = 50) -> List:
    embs = []
    for i in tqdm(range(0, len(texts), batch_size)):
        # Create embeddings optimized for document retrieval
        # (supported in textembedding-gecko@002 and later)
        result = model.get_embeddings(
            [
                TextEmbeddingInput(text=text,
            ]
        )
        embs.extend([e.values for e in result])
    return embs
```

4. This code snippet modifies the previously loaded DataFrame (df) by combining the `title` and `body` columns into a new `title_body` column. Then, it uses the `get_embeddings_wrapper` function to obtain text embeddings for each combined title and body, and the resulting embeddings are added as a new `embedding` column to the DataFrame. Finally, the first few rows of the updated DataFrame are displayed.

```
df["title_body"] = df["title"] + "\n" + df["body"]

df = df.assign(embedding=get_embeddings_wrapper(df.title_body))
df.head()
```

Output:



Task 4. Scrape HTML from Question Pages

1. Run the following code snippet in the next cell to set up necessary project information.

```
BUCKET_NAME = "qwiklabs-gcp-03-c48548fb59c9"
BUCKET_URI = "gs://qwiklabs-gcp-03-c48548fb59c9"
REGION = "us-central1"
PROJECT_ID = "qwiklabs-gcp-03-c48548fb59c9"
```

2. Run the following code snippet in the next cell, to create the Google Cloud

```
! gsutil mb -l $REGION -p $PROJECT_ID $BUCKET_URI
```

Output:

```
e:"2024-02-10T17:51:38.051134419+00:00"}]}
Creating gs://qwiklabs-gcp-02-154f8d656070/...
```

3. Create directories within the bucket.

```
%%bash

# Set your Google Cloud Storage bucket name
BUCKET_NAME="gs://qwiklabs-gcp-03-c48548fb59c9"

# Array of top-level directory names you want to create
TOP_LEVEL_DIRECTORIES=("embeddings-stackoverflow")

# Loop through the top-level array and create directories
for TOP_LEVEL_DIRECTORY in "${TOP_LEVEL_DIRECTORIES[@]}"; do
  gsutil -m cp -r /dev/null "$BUCKET_NAME/$TOP_LEVEL_DIRECTORY/"

  # Array of subdirectory names you want to create inside the
top-level directory
  SUBDIRECTORIES=("html")

  # Loop through the subdirectories array and create
subdirectories inside the top-level directory
  for SUBDIRECTORY in "${SUBDIRECTORIES[@]}"; do
    gsutil -m cp -r /dev/null
"$BUCKET_NAME/$TOP_LEVEL_DIRECTORY/$SUBDIRECTORY/"
  done
done

echo "Directories created successfully."
```

Output:

```
Copying file:///dev/null [Content-Type=application/octet-stream]...
/ [1/1 files][    0.0 B/    0.0 B]
Operation completed over 1 objects.
Copying file:///dev/null [Content-Type=application/octet-stream]...
/ [1/1 files][    0.0 B/    0.0 B]
Operation completed over 1 objects.
Directories created successfully.
```

- It sends an HTTP GET request to a specified URL (question_url) to scrape the content of a question page.
- If the request is successful (HTTP status code 200) and the response contains content, it uploads the HTML content of the question page to Google Cloud Storage (GCS).
- The GCS URI (Uniform Resource Identifier) of the uploaded HTML file is returned.

```
JSONL_MIME_TYPE = "application/jsonl"
HTML_MIME_TYPE = "text/html"

BUCKET_NAME = "qwiklabs-gcp-03-c48548fb59c9"
DIRECTORY = "embeddings-stackoverflow"
BLOB_PREFIX = f"{DIRECTORY}/html/"

GCS_URI_PREFIX = f"gs://{BUCKET_NAME}/{BLOB_PREFIX}"

storage_client = storage.Client()
bucket = storage_client.bucket(BUCKET_NAME)
```

```
    if response.status_code != 200 or not response.content:
        print(f"URL: {question_url} Code:
{response.status_code}")
        return None

    print(f"Scraping {question_url}")

    link_title = response.url.split("/")[-1] + ".html"
    gcs_uri = f"{GCS_URI_PREFIX}{link_title}"

    # Upload HTML to Google Cloud Storage
    blob = bucket.blob(f"{BLOB_PREFIX}{link_title}")
    blob.upload_from_string(response.content,
content_type=HTML_MIME_TYPE)
    time.sleep(1)
    return gcs_uri
```

5. The following code snippet has two main parts: it constructs URLs for Stack

5. The following code snippet has two main parts: it constructs URLs for Stack Overflow questions based on their IDs and then scrapes the HTML content from each of these URLs before uploading it to Google Cloud Storage (GCS). Run the

```
# Get the published URL from the ID
QUESTION_BASE_URL = "https://stackoverflow.com/questions/"
df["question_url"] = df["id"].apply(lambda x: f"
{QUESTION_BASE_URL}{x}")

# Scrape HTML from stackoverflow.com and upload to GCS
df["gcs_uri"] = df["question_url"].apply(scrape_question)
```

Output:

```
Scraping https://stackoverflow.com/questions/927358
Scraping https://stackoverflow.com/questions/5767325
Scraping https://stackoverflow.com/questions/2003505
Scraping https://stackoverflow.com/questions/16956810
Scraping https://stackoverflow.com/questions/4114095
Scraping https://stackoverflow.com/questions/2906582
Scraping https://stackoverflow.com/questions/503093
Scraping https://stackoverflow.com/questions/1789945
Scraping https://stackoverflow.com/questions/1783405
Scraping https://stackoverflow.com/questions/3207219
Scraping https://stackoverflow.com/questions/1125968
Scraping https://stackoverflow.com/questions/5585779
Scraping https://stackoverflow.com/questions/4366730
Scraping https://stackoverflow.com/questions/3437059
Scraping https://stackoverflow.com/questions/3552461
```

```
Scraping https://stackoverflow.com/questions/1602934
Scraping https://stackoverflow.com/questions/1200621
Scraping https://stackoverflow.com/questions/3294889
Scraping https://stackoverflow.com/questions/1085801
Scraping https://stackoverflow.com/questions/17071871
Scraping https://stackoverflow.com/questions/332289
Scraping https://stackoverflow.com/questions/11346283
Scraping https://stackoverflow.com/questions/1628088
Scraping https://stackoverflow.com/questions/9329446
Scraping https://stackoverflow.com/questions/2765421
Scraping https://stackoverflow.com/questions/1024847
Scraping https://stackoverflow.com/questions/2334712
Scraping https://stackoverflow.com/questions/82831
Scraping https://stackoverflow.com/questions/3010840
Scraping https://stackoverflow.com/questions/363681
Scraping https://stackoverflow.com/questions/901115
Scraping https://stackoverflow.com/questions/48198
Scraping https://stackoverflow.com/questions/3740152
Scraping https://stackoverflow.com/questions/455612
Scraping https://stackoverflow.com/questions/114543
Scraping https://stackoverflow.com/questions/351409
Scraping https://stackoverflow.com/questions/154059
Scraping https://stackoverflow.com/questions/613183
Scraping https://stackoverflow.com/questions/901712
Scraping https://stackoverflow.com/questions/4181703
Scraping https://stackoverflow.com/questions/348170
Scraping https://stackoverflow.com/questions/379906
Scraping https://stackoverflow.com/questions/3481828
```

Note: Please wait for 10 minutes to finish the scrapping..

6. In the next cell, restructure the embeddings data to JSONL to follow the Vertex AI Search format (Unstructured with Metadata). This format is required to use custom embeddings.

```
EMBEDDINGS_FIELD_NAME = "embedding_vector"


def format_row(row):
    return {
        "id": row["id"],
        "content": {"mimeType": HTML_MIME_TYPE, "uri":
row["gcs_uri"]},
        "structData": {
            EMBEDDINGS_FIELD_NAME: row["embedding"],
            "title": row["title"],
            "body": row["body"],
            "question_url": row["question_url"],
            "answer_count": row["answer_count"],
            "creation_date": row["creation_date"],
            "score": row["score"],
        },
```

```
vais_embeddings = (
    df.apply(format_row, axis=1)
    .to_json(orient="records", lines=True, force_ascii=False)
    .replace("\/", "/")  # To prevent escaping the / characters
)
```

7. In the next cell, upload the JSONL file to Google Cloud Storage.

```
jsonl_filename = f"{DIRECTORY}/vais_embeddings.jsonl"
embeddings_file = f"gs://{BUCKET_NAME}/{jsonl_filename}"

blob = bucket.blob(jsonl_filename)
blob.upload_from_string(vais_embeddings,
content_type=JSONL_MIME_TYPE)
```

Task 5. Set up AI Applications

1. In the next cell, set up client options for interacting with the Google Cloud Vertex AI Discovery Engine service. It specifies the API endpoint based on the provided `DATA_STORE_LOCATION`.

```
DATA_STORE_LOCATION = "global"

client_options = (
    ClientOptions(api_endpoint=f"{DATA_STORE_LOCATION}-
discoveryengine.googleapis.com")
    if DATA_STORE_LOCATION != "global"
    else None
)
```

2. In the next cell, define several functions that interact with the Google Cloud Vertex AI Discovery Engine service. These functions are responsible for creating a data store, updating its schema, importing documents, and creating a search engine.

```
    project_id: str, location: str, data_store_name: str,
data_store_id: str
):
    # Create a client
    client =
discoveryengine.DataStoreServiceClient(client_options=client_optior

    # Initialize request argument(s)
    data_store = discoveryengine.DataStore(
        display_name=data_store_name,
        industry_vertical="GENERIC",
        content_config="CONTENT_REQUIRED",
        solution_types=["SOLUTION_TYPE_SEARCH"],
    )

    request = discoveryengine.CreateDataStoreRequest(

parent=discoveryengine.DataStoreServiceClient.collection_path(
            project_id, location, "default_collection"
        ),
        data_store=data_store,
        data_store_id=data_store_id,
```

```
    try:
        operation.result()
    except GoogleAPICallError:
        pass


def update_schema(
    project_id: str,
    location: str,
    data_store_id: str,
):
    client =
discoveryengine.SchemaServiceClient(client_options=client_options)

    schema = discoveryengine.Schema(
        name=client.schema_path(project_id, location,
data_store_id, "default_schema"),
        struct_schema={
            "$schema": "https://json-schema.org/draft/2020-
12/schema",
            "type": "object",
```

```
                    "type": "array",
                    "keyPropertyMapping": "embedding_vector",
                    "dimension": 768,
                    "items": {"type": "number"},
                }
            },
        },
    )

    operation = client.update_schema(

request=discoveryengine.UpdateSchemaRequest(schema=schema)
    )
```

```
    print("Waiting for operation to complete...")

    response = operation.result()

    # Handle the response
    print(response)
```

```
    location: str,
    data_store_id: str,
    gcs_uri: str,
):
    client =
discoveryengine.DocumentServiceClient(client_options=client_options

    # The full resource name of the search engine branch.
    # e.g.
projects/{project}/locations/{location}/dataStores/{data_store_id}/
    parent = client.branch_path(
        project=project_id,
        location=location,
        data_store=data_store_id,
        branch="default_branch",
    )

    request = discoveryengine.ImportDocumentsRequest(
        parent=parent,
        gcs_source=discoveryengine.GcsSource(input_uris=
[gcs_uri]),
        # Options: `FULL`, `INCREMENTAL`
```

```
    )

    # Make the request
    operation = client.import_documents(request=request)


def create_engine(
    project_id: str, location: str, data_store_name: str,
data_store_id: str
):
    client =
discoveryengine.EngineServiceClient(client_options=client_options)

    # Initialize request argument(s)
    config = discoveryengine.Engine.SearchEngineConfig(
        search_tier="SEARCH_TIER_ENTERPRISE", search_add_ons=
["SEARCH_ADD_ON_LLM"]
    )

    engine = discoveryengine.Engine(
        display_name=data_store_name,
        solution_type="SOLUTION_TYPE_SEARCH",
```

```
        search_engine_config=config,
    )

    request = discoveryengine.CreateEngineRequest(

parent=discoveryengine.DataStoreServiceClient.collection_path(
            project_id, location, "default_collection"
        ),
        engine=engine,
        engine_id=engine.display_name,
    )

    # Make the request
    operation = client.create_engine(request=request)
    response = operation.result(timeout=90)
```

3. In the next cell, set the project related variables.

```
DATA_STORE_NAME = "stackoverflow-embeddings"
DATA_STORE_ID = f"{DATA_STORE_NAME}-id"
```

Vertex AI Discovery Engine, including creating a data store, updating its schema for embeddings, importing documents, and creating a search engine attached to the data store.

```
# Create a Data Store
create_data_store(PROJECT_ID, DATA_STORE_LOCATION,
DATA_STORE_NAME, DATA_STORE_ID)

# Update the Data Store Schema for embeddings
```

```
update_schema(PROJECT_ID, DATA_STORE_LOCATION, DATA_STORE_ID)

# Import the embeddings JSONL file
import_documents(PROJECT_ID, DATA_STORE_LOCATION, DATA_STORE_ID,
embeddings_file)

# Create a Search App and attach the Data Store
create_engine(PROJECT_ID, DATA_STORE_LOCATION, DATA_STORE_NAME,
DATA_STORE_ID)
```

Output:

Next, we need to set the embedding specification for the data store. We will set the same specifications for all search requests: `0.5 * relevance_score`.

This is not supported in client libraries, so we will use the `requests` module to make a REST request Documentation: Bring Embeddings

5. Run the following code snippet in the next cell, that retrieves an access token using `gcloud auth print-access-token`, and then it sends a PATCH request to update the serving configuration of the search application in Google Cloud Vertex AI Discovery Engine. The request includes the embedding configuration and a ranking expression, and the server's response is printed.

```
access_token = (
    subprocess.check_output(["gcloud", "auth", "print-access-
token"])
    .decode("utf-8")
    .strip()
)

response = requests.patch(
    headers={
        "Authorization": f"Bearer {access_token}",
        "Content-Type": "application/json; charset=utf-8",
        "X-Goog-User-Project": PROJECT_ID,
    },
    json={
        "name":
f"projects/{PROJECT_ID}/locations/{DATA_STORE_LOCATION}/collections
        "embeddingConfig": {"fieldPath": EMBEDDINGS_FIELD_NAME},
        "ranking_expression": "0.5 * relevance_score",
    },
)

print(response.text)
```

Output:

Task 6. Test Search Application

1. Run the following code snippet in the next cell, to define a function named `search_data_store` that performs a search operation on a Google Cloud Vertex AI Discovery Engine data store.

```
def search_data_store(
    project_id: str,
    location: str,
    data_store_id: str,
    search_query: str,
) -> List[discoveryengine.SearchResponse]:
    # Create a client
    client =
discoveryengine.SearchServiceClient(client_options=client_options)

    # e.g.
projects/{project_id}/locations/{location}/dataStores/{data_store_i
    serving_config = client.serving_config_path(
        project=project_id,
        location=location,
        data_store=data_store_id,
        serving_config="default_config",
    )

    # Optional: Configuration options for search
    # Refer to the `ContentSearchSpec` reference for all
supported fields:
    #
https://cloud.google.com/python/docs/reference/discoveryengine/late
    content_search_spec =
discoveryengine.SearchRequest.ContentSearchSpec(
        # For information about snippets, refer to:
        # https://cloud.google.com/generative-ai-app-
builder/docs/snippets

snippet_spec=discoveryengine.SearchRequest.ContentSearchSpec.Snippe
            return_snippet=True

        # https://cloud.google.com/generative-ai-app-
builder/docs/get-search-summaries

summary_spec=discoveryengine.SearchRequest.ContentSearchSpec.Summar
            summary_result_count=5,
            include_citations=True,
            ignore_adversarial_query=True,
            ignore_non_summary_seeking_query=True,
        ),
    )

    # Refer to the `SearchRequest` reference for all supported
fields:
    #
https://cloud.google.com/python/docs/reference/discoveryengine/late
    request = discoveryengine.SearchRequest(
        serving_config=serving_config,
        query=search_query,
        page_size=10,
        content_search_spec=content_search_spec,

query_expansion_spec=discoveryengine.SearchRequest.QueryExpansionSp

        ),

spell_correction_spec=discoveryengine.SearchRequest.SpellCorrectior

mode=discoveryengine.SearchRequest.SpellCorrectionSpec.Mode.AUTO
        ),
    )

    response = client.search(request)
    return response
```

2. Run the following code snippet in the next cell, to perform a search operation on a Google Cloud Vertex AI Discovery Engine data store using a specified search query and prints the summary text of the search response.

```
search_query = "How do I create an array in Java?"

response = search_data_store(
    PROJECT_ID, DATA_STORE_LOCATION, DATA_STORE_ID, search_query
)
```

Output:

reate an array of integers with the values 1, 2, 3, 4, and 5, you would use the following code: `int myArray = {1, 2, 3, 4, 5};`. Java can determine the size of the array implicitl
y based on the number of values you provide. [1]

Note: If you receive an error, then re-run the same cell after a 3-4 minutes.

If you see the message **No results could be found. Try rephrasing the search query.** in the output wait for few more minutes and re-run the command.

Congratulations!

At the end of this lab, you have gained an understanding of utilizing `text-embedding-004` in Vertex AI to obtain text embeddings and converting them into the required format for Vertex AI Search. Additionally, you have learned how to create a search application that leverages custom embeddings, facilitating a seamless integration of advanced text-based search capabilities.

Manual Last Updated March 06, 2025

Lab Last Tested March 06, 2025