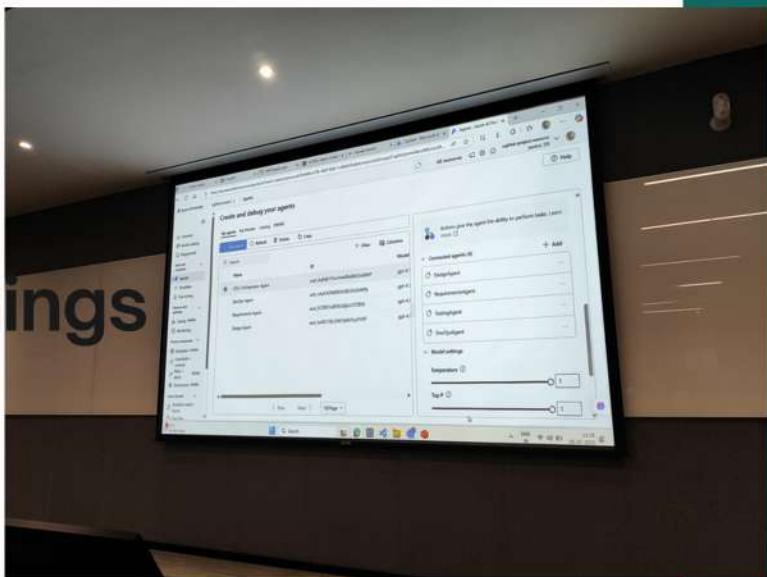


Learnings



```
Run the following cell to configure your working environment for this notebook.  
NOTE: The qa object is only used in Databricks Academy courses and is not available outside of these courses. It will dynamically reference the information needed to run the course in the lab environment.  
  
%%python  
from qd import ClassroomSetup  
qa = ClassroomSetup()  
qa.notebook_setup()  
  
Run the cell below to view your default catalog and schema. Notice that your default catalog is dbacademy and your default schema is your unique fall2022 schema.  
NOTE: The default catalog and schema are pre-configured for you to avoid the need to specify the three-level name when writing your tables (e.g., catalog.schema.table).  
  
SELECT current_catalog(), current_schema();
```

B. Overview of CTAS with `read_files()` for ingestion of JSON files

Data Engineering | Data Ingestion with Lakeflow Connect

```
00 - Ingesting JSON files with Databricks
File Edit View Run Help
--> The JSON file is clearly read into a tabular format with 8 columns.
--> The key and value columns are base64-encoded and returned as STRING data type.
--> There are no rows in the _measurement_data column.

SELECT *
FROM raw_data
WHERE _measurement = 'power'
LIMIT 100
```

key	offset	partition	timestamp	topic	value
1	2193200000	0	1533000000000	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
2	2193200001	0	1533000001200	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
3	2193200002	0	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
4	2193200003	0	1533000001710	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
5	2193200004	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
6	2193200005	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
7	2193200006	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
8	2193200007	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
9	2193200008	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
10	2193200009	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
11	2193200010	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
12	2193200011	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==
13	2193200012	1	1533000001700	rawdata	ewAD2fDpUQ0L0mHdJwzWmHw==

The screenshot shows a Databricks notebook interface. On the left is a sidebar with navigation links like 'Recent', 'Catalog', 'Jobs & Pipelines', 'Compute', 'Marketplace', 'SQL', 'SQL Editor', 'Queries', 'Dashboards', 'Gene', 'Alerts', 'Query History', 'SQL Warehouses', 'Data Engineering', 'Job Runs', 'Data Ingestion', 'Notebooks', 'Playground', 'Agents Beta', 'Experiments', and 'Features'. The main area has a title '2.1 - Preparing Data for RAG' and a sub-section '2.1.1 - Saving Embeddings'. It contains two code cells:

```
# splitting the contents into batches of 150 items each, since the embedding model takes at most 150 inputs per request.
max_batch_size = 150
batches = [content[i:i+max_batch_size] for i in range(0, len(contents), max_batch_size)]

# process each batch and collect the results.
all_embeddings = []
for batch in batches:
    all_embeddings.append(get_embeddings(batch))

return pd.Series(all_embeddings)
```

```
import pyspark.sql.functions as F
df_chunks_mod = df_chunks
    .withColumn("embedding", get_embedding("content"))
    .selectExpr("path_name", "content", "embedding")
display(df_chunks_mod)
```

Below the code cells, there is a button labeled 'Save Embeddings to a Delta Table'.

```
sql
CREATE TABLE IF NOT EXISTS pdf_text_embeddings (
    id BIGINT GENERATED BY DEFAULT AS IDENTITY,
    text STRING,
    content STRING,
    embedding ARRAY <FLOAT>
)
-- NOTE: the table has to be CDC because VectorSearch is using BLT that is requiring CDC state
) TBLPROPERTIES (delta.enableChangeDataFeed = true);
```

```
python
embedding_table_name = f"DA.catalog_name.{DA.schema_name}.pdf_text_embeddings"
df_chunk_end.write.mode("append").createOrReplaceTempView(embedding_table_name)
```

Why are Vector Databases So Hot?

Query time and scalability

- Specialized, full-fledged databases for **unstructured data**
 - Inherit database properties, i.e. Create-Read-Update-Delete (CRUD)
- Speed up query search for the closest vectors
 - Uses vector search algorithms such as Approximate Nearest Neighbor (ANN)
 - Organize embeddings into indices



©2024 Databricks Inc. — All rights reserved.

LIVE



The screenshot shows the Databricks interface with the following details:

- Left Sidebar:** Includes sections for Home, Workspace, Recent, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL (SQL Editor, Queries, Databricks, Gene, Alerts, Query History, SQL Workbooks), Data Engineering (Job Runs, Data Ingestion), and Admin (Pipelines, Policy).
- Central Area:** Shows the path Clusters > Vector Search > **vs_endpoint_3**. Below this, there is a "Serverless Budget Policy" section.
- Table View:** Displays the following information for the index:

Index Name	Type	Creator
dbacademy/dabuser10011729_1703933119.pdf_text_self_managed_dg_index	Delta Sync	dabuser10011729_1703933119@vicareum.com

Reranking

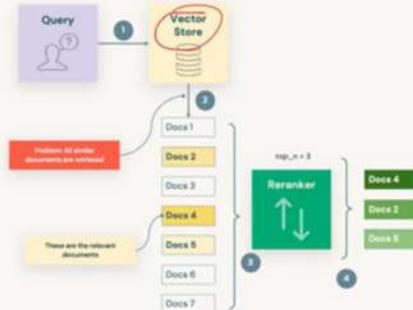
A method of prioritizing documents most relevant to user's query

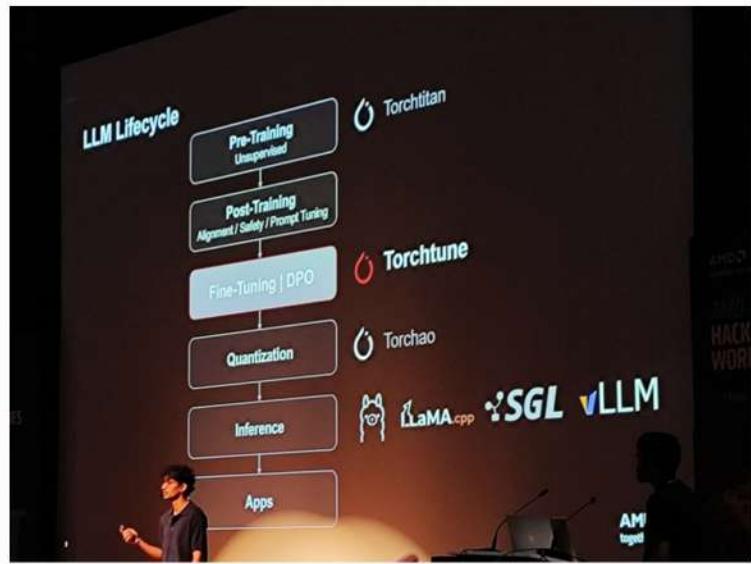
- **Initial retrieval**

- Not all documents are equally important.
- We should use only the relevant documents.

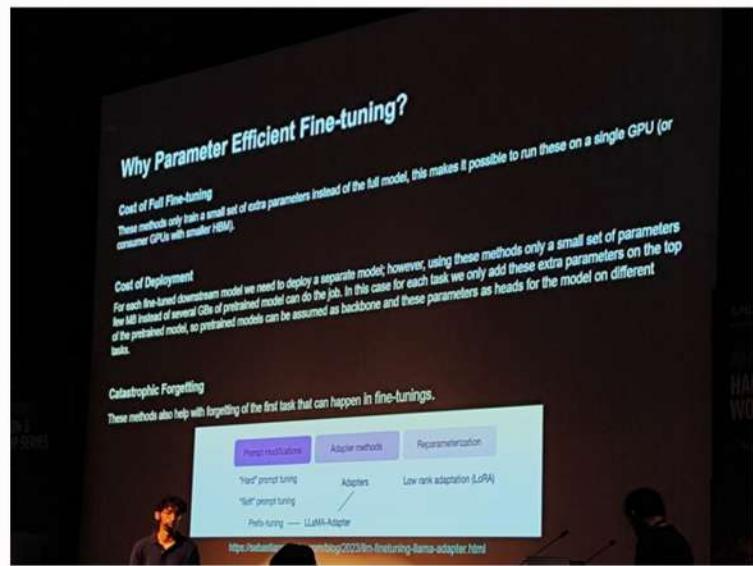
- **Reranker**

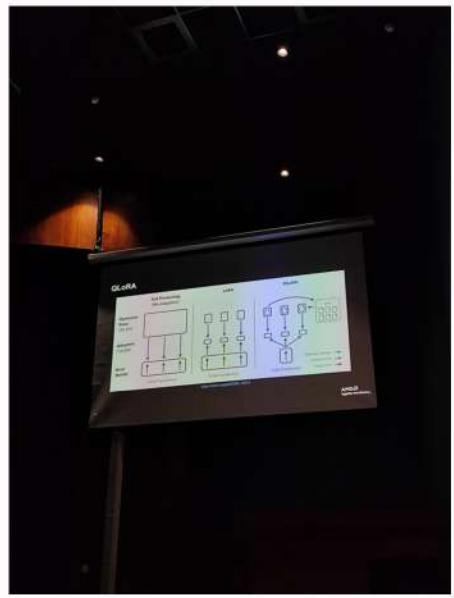
- Reorder documents based on the relevance scores.
- The goal is to **place most relevant documents at the top of the list**.





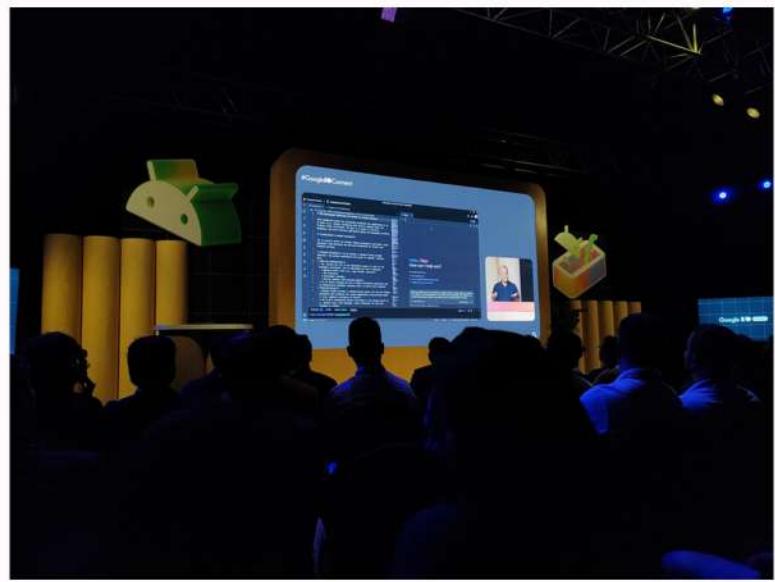


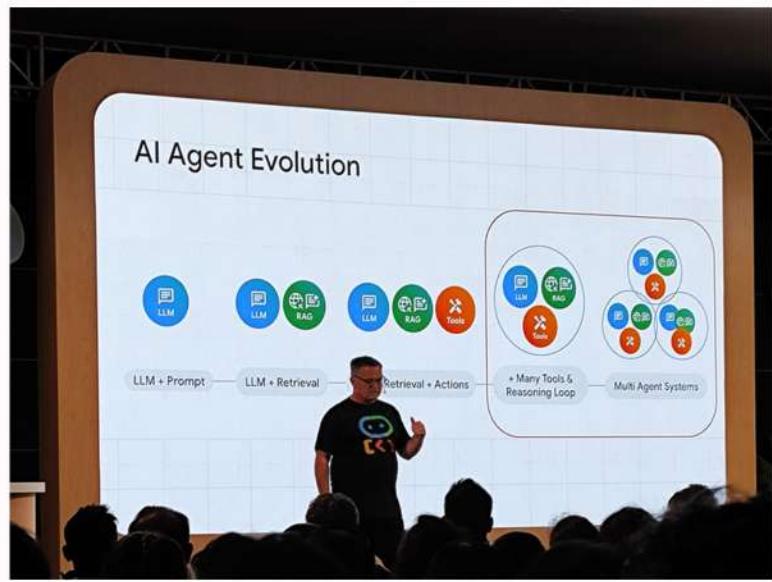


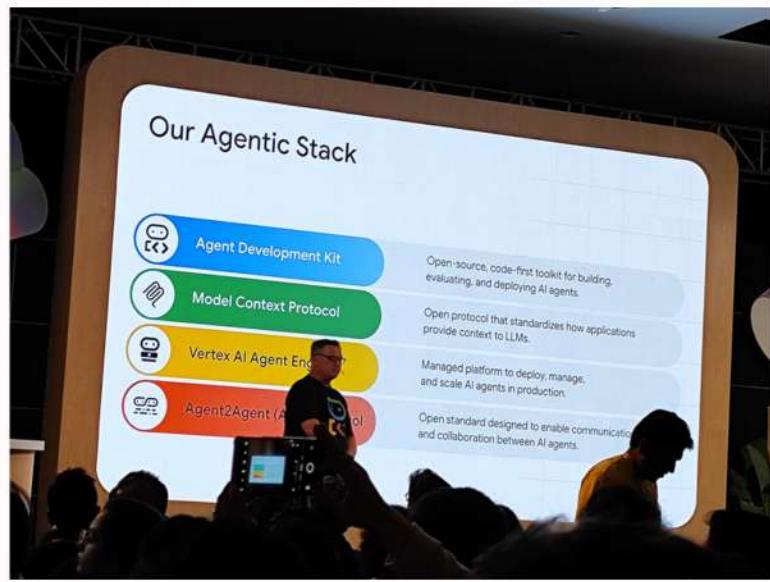


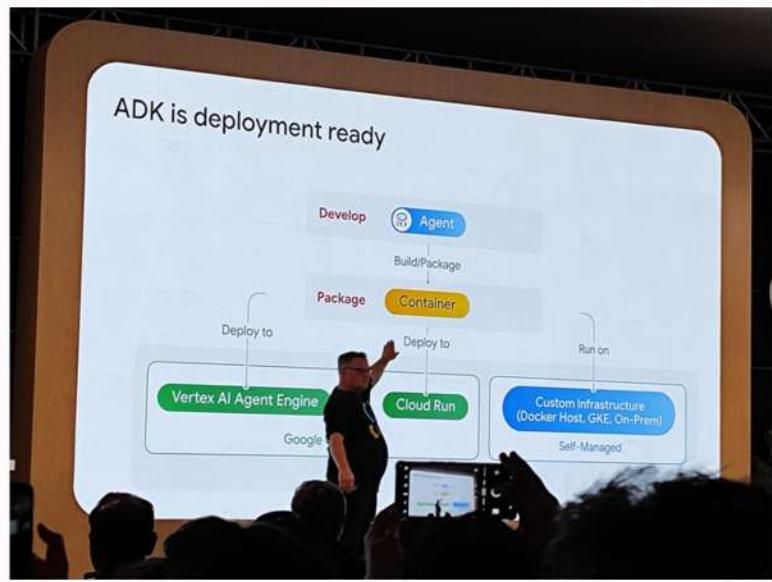


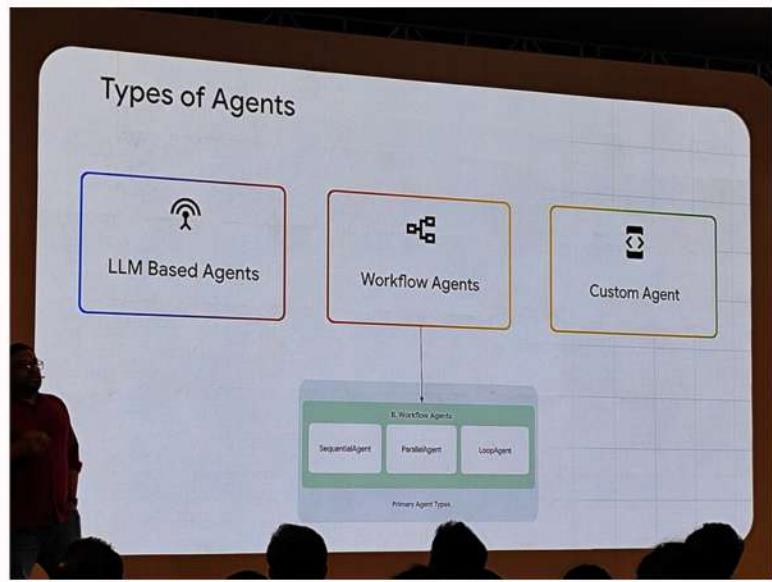


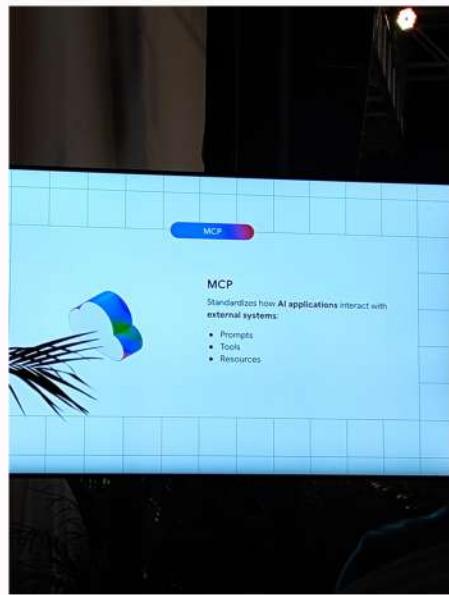


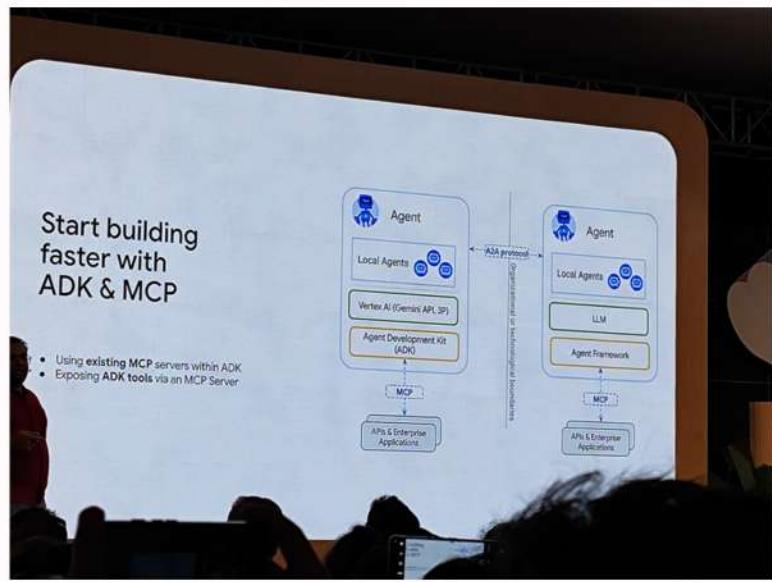


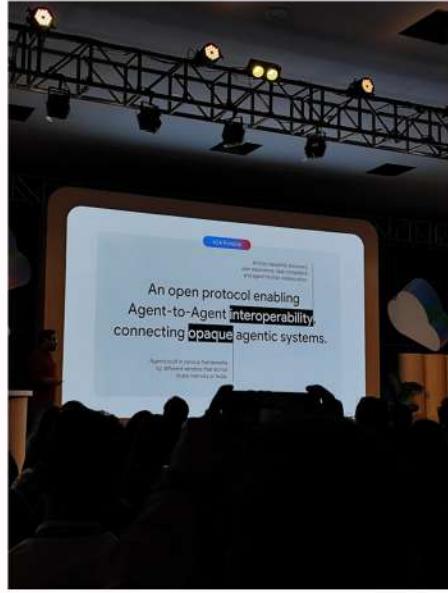






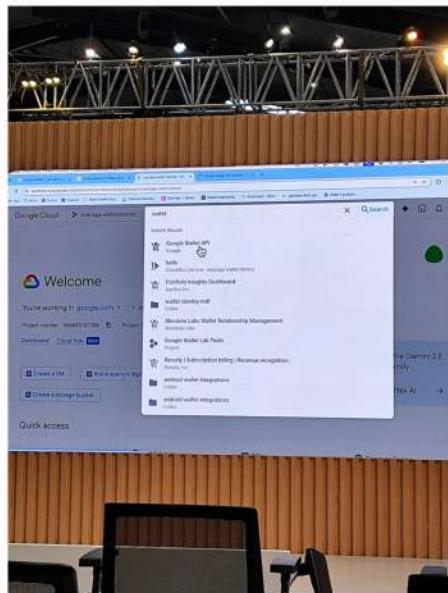












Why are Vector Databases So Hot?

Query time and scalability

- Specialized, full-fledged databases for **unstructured data**
 - Inherit database properties, i.e. Create-Read-Update-Delete (CRUD)
- Speed up query search for the closest vectors
 - Uses vector search algorithms such as Approximate Nearest Neighbor (ANN)
 - Organize embeddings into indices



©2024 Databricks Inc. — All rights reserved.

LIVE

```
SQL
CREATE TABLE IF NOT EXISTS pdf_text_embeddings (
    id INT GENERATED BY DEFAULT AS IDENTITY,
    text STRING,
    embedding EMBEDDING,
    TBLPROPERTIES (enableChangeDataFeed = true)
```

```
Python
embedding_table_name = f'{DA.catalog_name}.{DA.schema_name}.pdf_text_embeddings'
df_chunk_end_write.mode("append").saveTable(embedding_table_name)
```

The screenshot shows a Databricks notebook interface with the following details:

- Title:** 2.1 - Preparing Data for RAG
- Languages:** Python
- Code Content:**

```
# splitting the contents into batches of 150 items each, since the embedding model takes at most 150 inputs per request.
max_batch_size = 150
batches = [contents[i:i+max_batch_size] for i in range(0, len(contents), max_batch_size)]

# process each batch and collect the results
all_embeddings = []
for batch in batches:
    all_embeddings.append(get_embeddings(batch))

return pd.Series(all_embeddings)
```

```
interrupt 10:09
import pyspark.sql.functions as F

df_chunks_emb = df_chunks
    .withColumn("embedding", get_embedding("content"))
    .selectExpr("pdf_name", "content", "embedding")
    .display(df_chunks_emb)
```
- Bottom Panel:** Save Embeddings to a Delta Table

Data Engineering | Data Ingestion with Lakeflow Connect

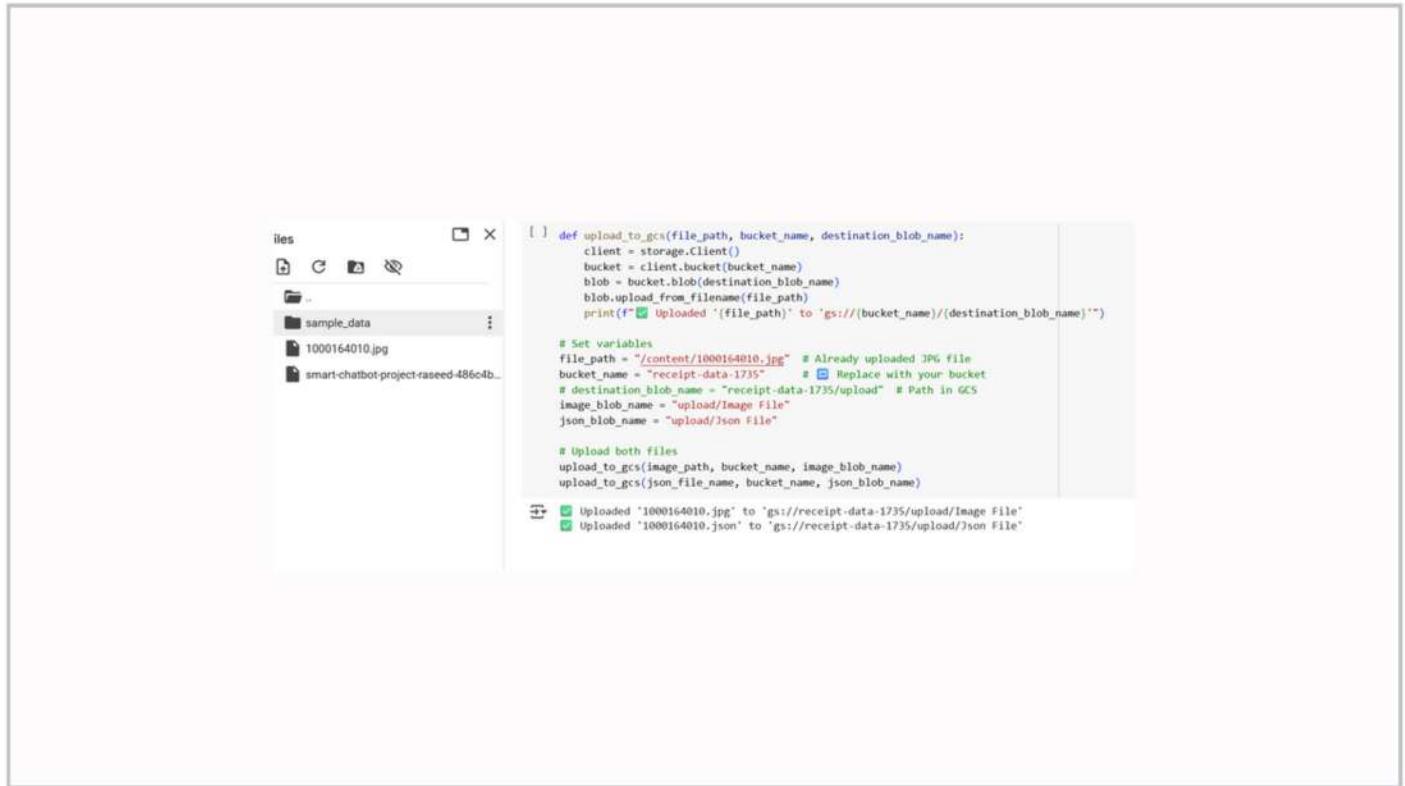
```
00 - Ingesting JSON files with Databricks
File Edit View Run Help
--> The JSON file is clearly read into a tabular format with 8 columns:
--> The key and value columns are base64-encoded and returned as STRING data type.
--> There are no rows in the _raw_json_data column.

SELECT *
FROM read_json(
    's3://cloud-databricks-connector/testdata/S3/2023/07/11/_raw_json_data',
    format := 'json'
)
LIMIT 10
```

key	offset	partition	timestamp	topic	value
1	219350000	0	193000000000	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
2	219350001	0	193000000100	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
3	219350002	0	193000000110	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
4	219350003	0	193000000120	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
5	219350004	1	193000000130	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
6	219350005	1	193000000140	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
7	219350006	1	193000000150	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
8	219350007	1	193000000160	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
9	219350008	1	193000000170	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
10	219350009	1	193000000180	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
11	219350010	1	193000000190	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
12	219350011	1	193000000200	customer	ewAD2fDpUQ0L0mHdLwzWmHw==
13	219350012	1	193000000210	customer	ewAD2fDpUQ0L0mHdLwzWmHw==

```
Run the following cell to configure your working environment for this notebook.  
NOTE: The 'pa' object is only used in Databricks Academy courses and is not available outside of these courses. It will dynamically reference the information needed to run the course in the lab environment.  
  
%%spark  
from io import StringIO  
import requests  
from pyspark.sql import DataFrame, Row  
from pyspark.sql.functions import col, lit  
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType  
  
# Read the file from the URL  
url = "https://databricks-datasets/structured-data/json/retail_sample.json"  
response = requests.get(url)  
data = response.content  
  
# Create a DataFrame from the JSON data  
df = DataFrame.from_json(StringIO(data), schema)  
  
# Show the first few rows of the DataFrame  
df.show(5)  
  
# Print the schema of the DataFrame  
df.printSchema()  
  
# Select specific columns from the DataFrame  
df.select("product_id", "product_name", "category_id", "category_name").show(5)  
  
# Print the schema of the selected DataFrame  
df.select("product_id", "product_name", "category_id", "category_name").printSchema()
```

B. Overview of CTAS with `read_files()` for ingestion of JSON files



The screenshot shows a terminal window with a file browser interface on the left and a code editor on the right. The file browser shows a directory structure with a folder named 'sample_data' containing two files: '1000164010.jpg' and 'smart-chatbot-project-raseed-486c4b..'. The code editor contains Python code for uploading files to Google Cloud Storage (GCS). The code defines a function 'upload_to_gcs' that takes file path, bucket name, and destination blob name as arguments. It uses the 'storage.Client()' class to interact with GCS. The code then sets variables for the file path ('/content/1000164010.jpg'), bucket name ('receipt-data-1735'), and destination blob names ('receipt-data-1735/upload/Image File' and 'receipt-data-1735/upload/Json File'). Finally, it calls the function twice: once for the image file and once for the JSON file.

```
[ ] def upload_to_gcs(file_path, bucket_name, destination_blob_name):
    client = storage.Client()
    bucket = client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)
    blob.upload_from_filename(file_path)
    print(f"Uploaded '{file_path}' to 'gs://{bucket_name}/{destination_blob_name}'")

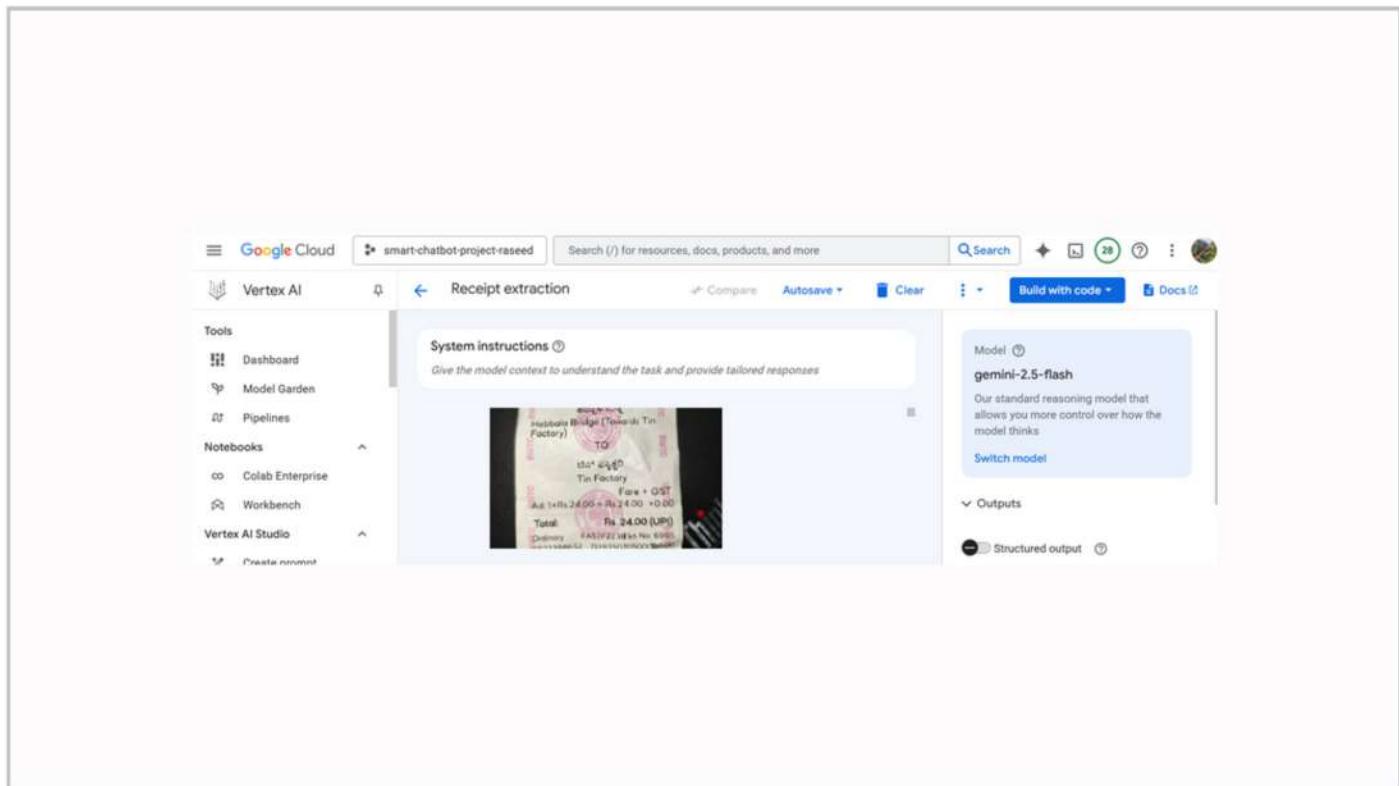
# Set variables
file_path = "/content/1000164010.jpg" # Already uploaded JPG file
bucket_name = "receipt-data-1735" # Replace with your bucket
# destination_blob_name = "receipt-data-1735/upload" # Path in GCS
image_blob_name = "upload/Image File"
json_blob_name = "upload/Json File"

# Upload both files
upload_to_gcs(image_path, bucket_name, image_blob_name)
upload_to_gcs(json_file_name, bucket_name, json_blob_name)

[+] [✓] Uploaded '1000164010.jpg' to 'gs://receipt-data-1735/upload/Image File'
[✓] Uploaded '1000164010.json' to 'gs://receipt-data-1735/upload/Json File'
```

The screenshot shows a Jupyter Notebook interface with a sidebar labeled "Files" containing "sample_data", "1000164010.jpg", and "smart-chatbot-project-raseed-486c4b559e2d.ipynb". The main area displays Python code:

```
[ ] from google.colab import files  
uploaded = files.upload() # Upload your downloaded key.json file here  
[ ] Choose File: smart-chatb...59e2d.json  
• smart-chatbot-project-raseed-486c4b559e2d.json (application/json) - 2422 bytes, last modified: 7/27/2025 - 100% done  
Saving smart-chatbot-project-raseed-486c4b559e2d.json to smart-chatbot-project-raseed-486c4b559e2d.json  
[ ] import os  
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "/content/smart-chatbot-project-raseed-486c4b559e2d.json"  
[ ] from google.cloud import vision  
[ ]  
import io  
import json  
# Initialize Vision client  
client = vision.ImageAnnotatorClient()
```



[Expand to show model's summarized thoughts](#)

Sure, మీరు మొత్తం ఎంత ఖర్చు పెట్టారో నేను మీకు చెప్పగలను.

- బస్ లైఫ్ కోసం మీరు **24** రూపాయలు ఖర్చు చేశారు.
- మరియు కిరాగా వస్తువుల కోసం **384** రూపాయలు ఖర్చు చేశారు.

రెండు కలిపి, మీరు మొత్తం **408** రూపాయలు ఖర్చు చేశారు.

Write your prompt here

apps Details 1 2 3 4 5 6 7 8

Google Cloud smart-chatbot-project-raseed

Cloud Storage Object details

Overview Buckets Monitoring Settings **Intelligence** Insights datasets Configuration

Buckets > receipt-data-1735 > upload > Image File

time
Bucket retention retain until time
None

Hold status
None

Encryption type
Google-managed

Based on the receipts:

- **Time of Purchase:** You made one purchase in the **late morning** (around 9:30 AM) and another in the **mid-morning** (around 11:15 AM).
- **What I understand in crisp:** Your purchases suggest a tendency to shop and travel during **non-peak or less crowded morning hours**, indicating either a flexible schedule or a preference for convenience over rush.

Write your prompt here

