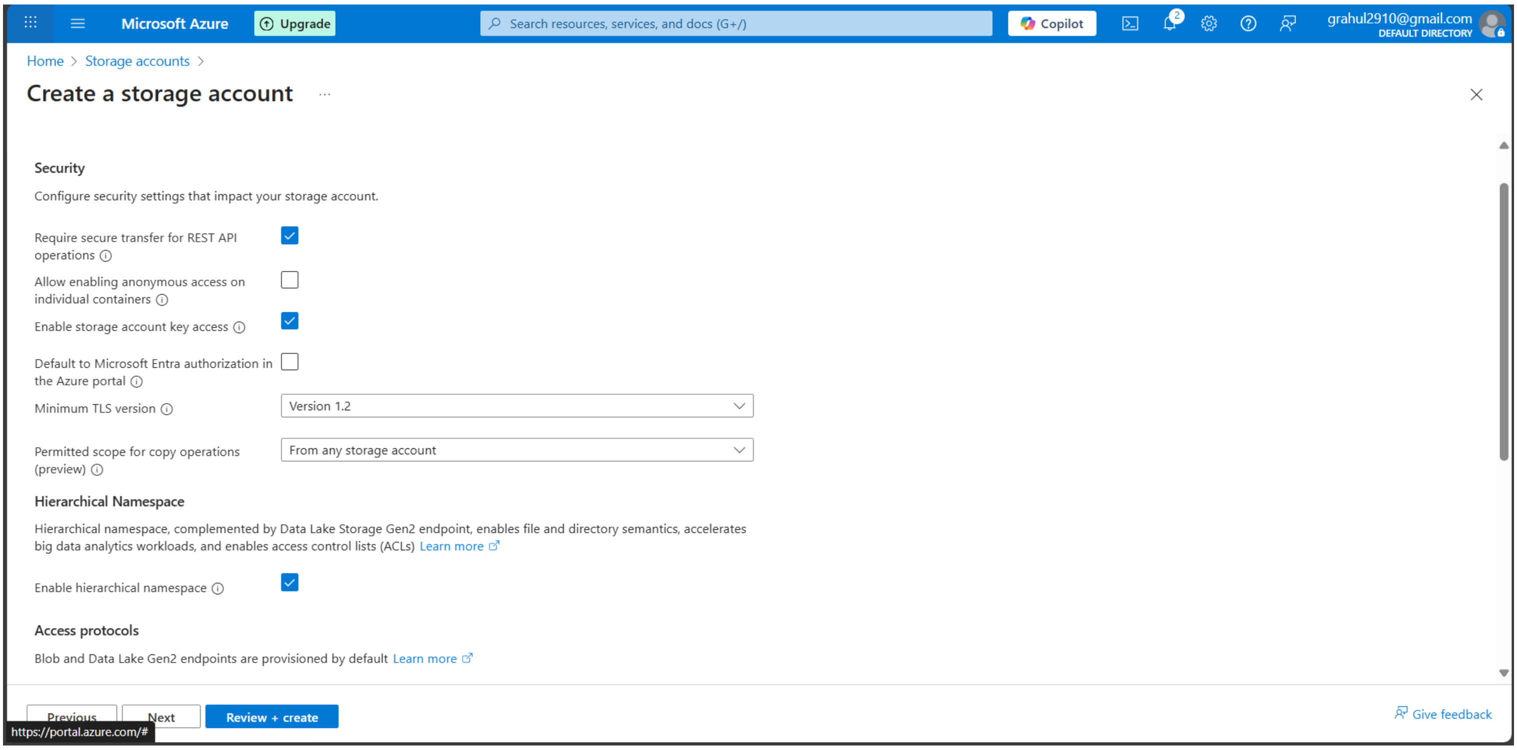
Reading Different File Formats from ADLS Gen2 into Delta Tables using Azure Databricks

□ Setup Azure Resources

- Created a Resource Group in Azure Portal to logically group all resources.
- Created an Azure Data Lake Storage Gen2 (ADLS Gen2) account to store customer data files.


Microsoft Azure
Upgrade
Search resources, services, and docs (G+/)
Copilot
grahul2910@gmail.com
DEFAULT DIRECTORY
Home > Storage accounts >

Create a storage account

Security

Configure security settings that impact your storage account.

Require secure transfer for REST API operations ☑

Allow enabling anonymous access on individual containers □

Enable storage account key access ☑

Default to Microsoft Entra authorization in the Azure portal □

Minimum TLS version: Version 1.2

Permitted scope for copy operations (preview): From any storage account

Hierarchical Namespace

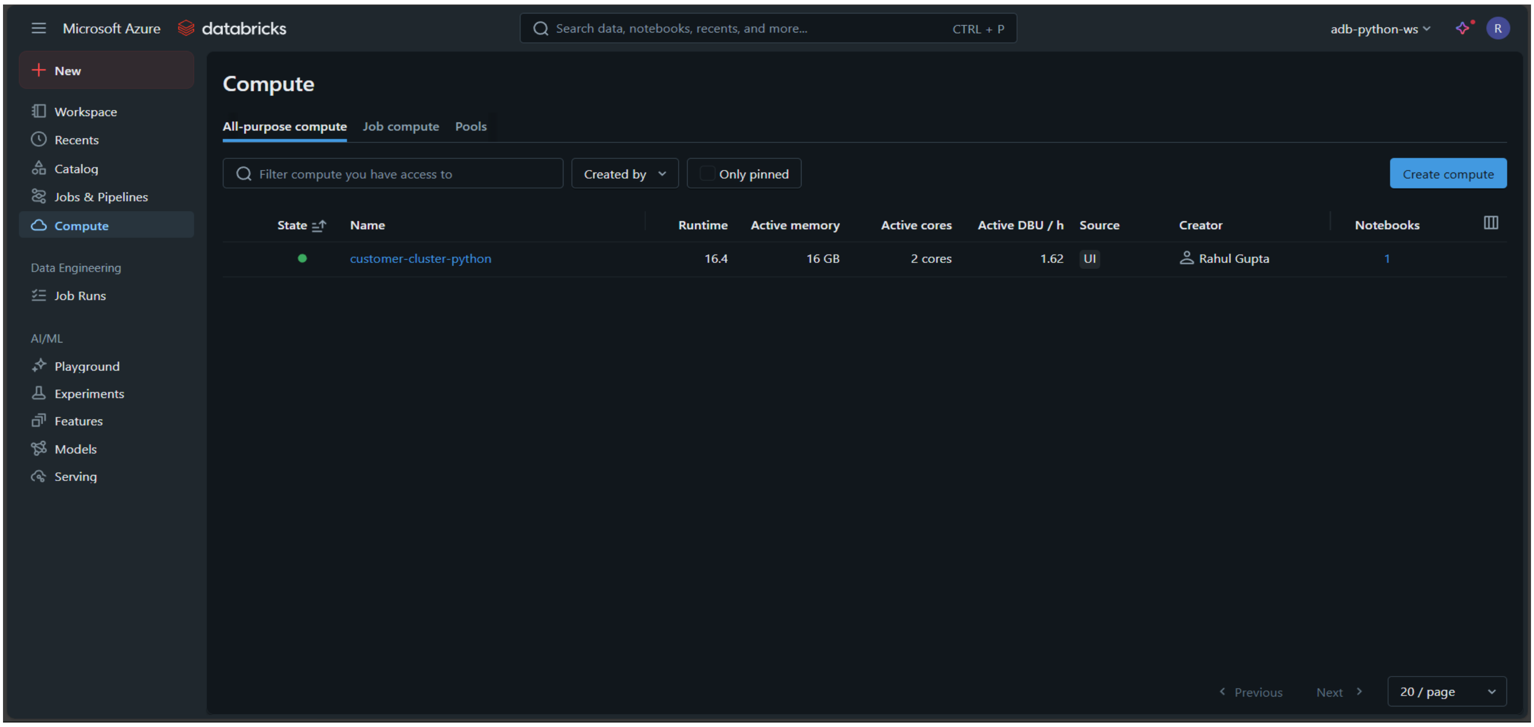
Hierarchical namespace, complemented by Data Lake Storage Gen2 endpoint, enables file and directory semantics, accelerates big data analytics workloads, and enables access control lists (ACLs) Learn more

Enable hierarchical namespace ☑

Access protocols

Blob and Data Lake Gen2 endpoints are provisioned by default Learn more

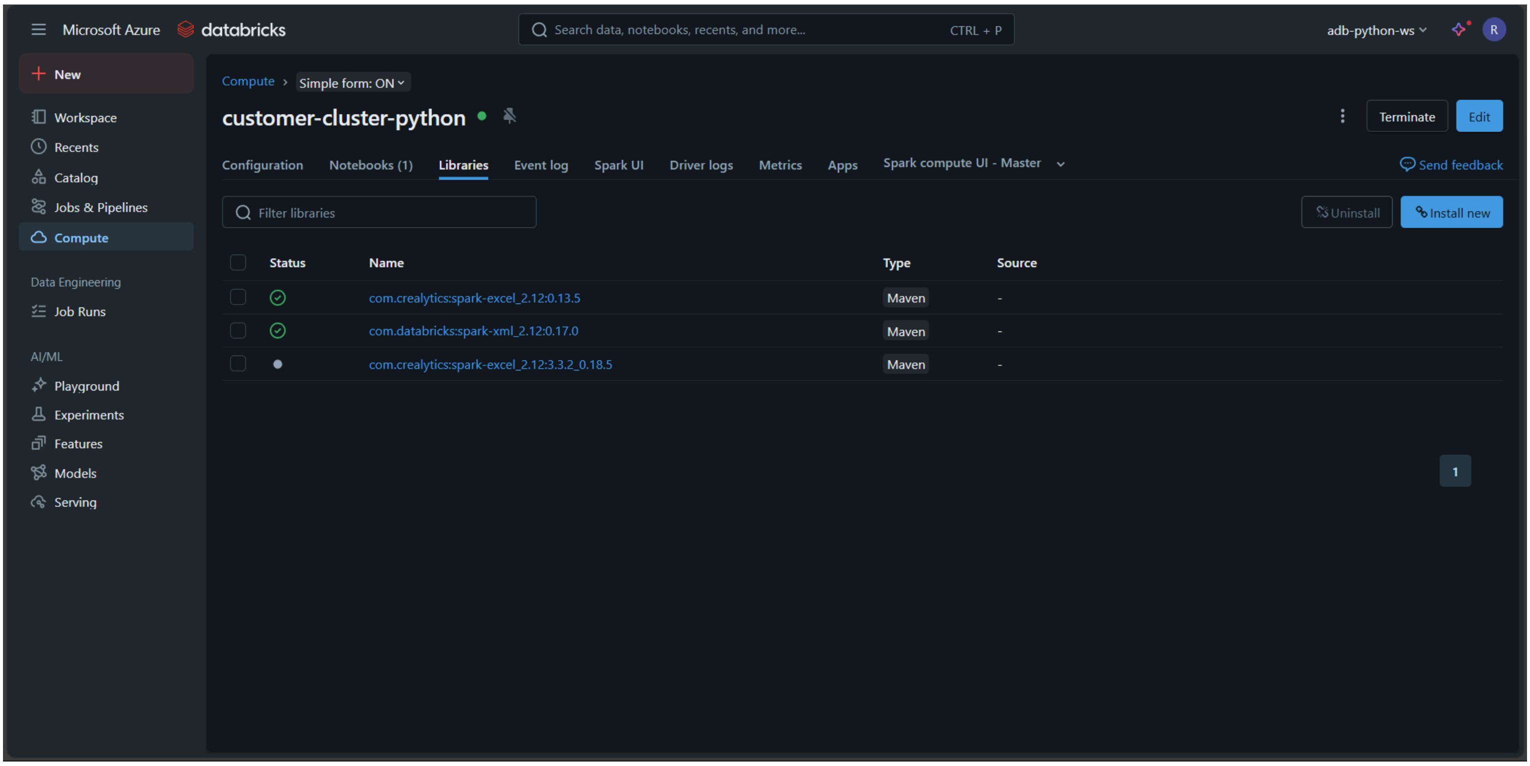
- Created an Azure Databricks workspace to run Spark clusters and process data.



Compute

State	Name	Runtime	Active memory	Active cores	Active DBU / h	Source	Creator	Notebooks
●	customer-cluster-python	16.4	16 GB	2 cores	1.62	UI	Rahul Gupta	1

- Configured Linked Services in Azure Databricks to securely connect to the ADLS Gen2 storage.
- Install the required Libraries

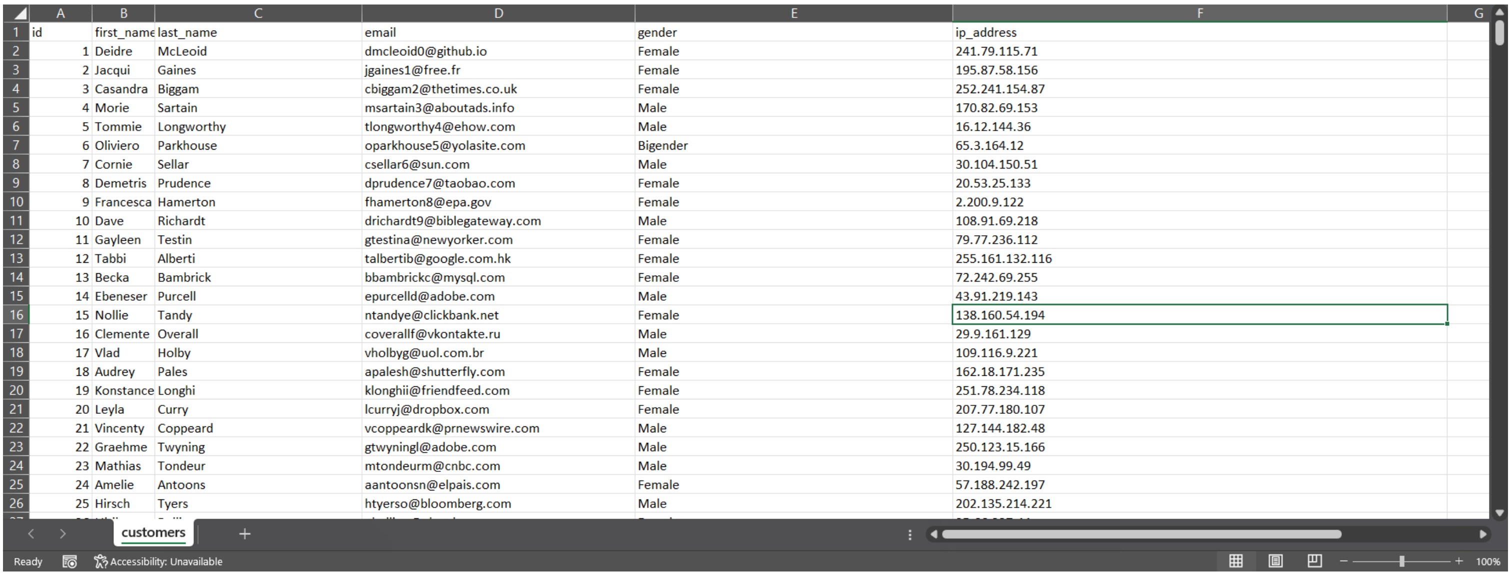


customer-cluster-python

Status	Name	Type	Source
✓	com.crealytics:spark-excel_2.12:0.13.5	Maven	-
✓	com.databricks:spark-xml_2.12:0.17.0	Maven	-
●	com.crealytics:spark-excel_2.12:3.3.2_0.18.5	Maven	-

□ Upload Sample Customer Data

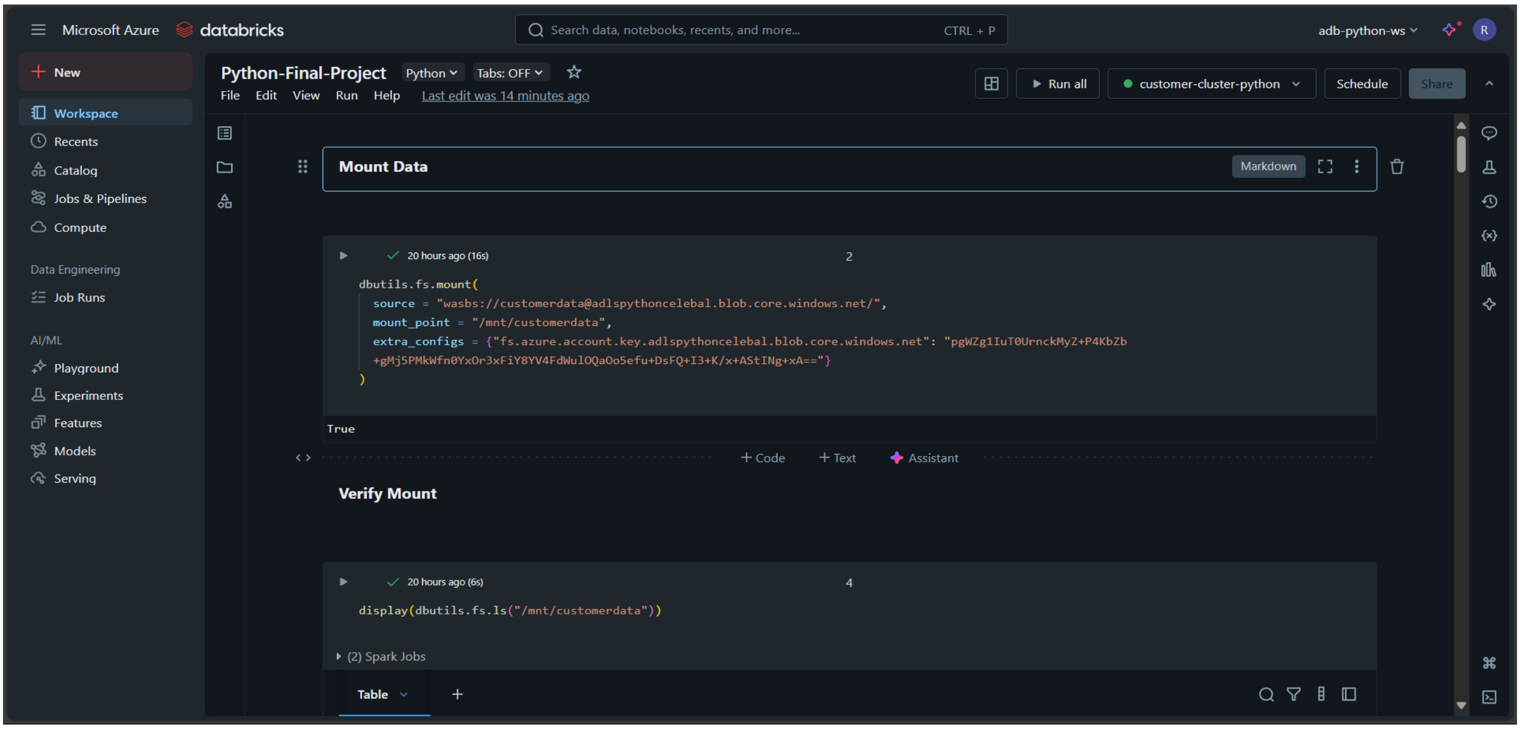
- Generated random customer data with 500 records in



id	first_name	last_name	email	gender	ip_address
1	Deidre	McLeoid	dmcleoid0@github.io	Female	241.79.115.71
2	Jacqui	Gaines	jgaines1@free.fr	Female	195.87.58.156
3	Casandra	Biggam	cbiggam2@thetimes.co.uk	Female	252.241.154.87
4	Morie	Sartain	msartain3@aboutads.info	Male	170.82.69.153
5	Tommie	Longworthy	tlongworthy4@ehow.com	Male	16.12.144.36
6	Oliviero	Parkhouse	oparkhouse5@yolasite.com	Bigender	65.3.164.12
7	Cornie	Sellar	csellar6@sun.com	Male	30.104.150.51
8	Demetris	Prudence	dprudence7@taobao.com	Female	20.53.25.133
9	Francesca	Hamerton	fhamerton8@epa.gov	Female	2.200.9.122
10	Dave	Richardt	drichardt9@biblegateway.com	Male	108.91.69.218
11	Gayleen	Testin	gtestina@newyorker.com	Female	79.77.236.112
12	Tabbi	Alberti	talbertib@google.com.hk	Female	255.161.132.116
13	Becka	Bambrick	bbambrickc@mysql.com	Female	72.242.69.255
14	Ebeneser	Purcell	epurcelld@adobe.com	Male	43.91.219.143
15	Nollie	Tandy	ntandye@clickbank.net	Female	138.160.54.194
16	Clemente	Overall	coverallf@vkontakte.ru	Male	29.9.161.129
17	Vlad	Holby	vholbyg@uol.com.br	Male	109.116.9.221
18	Audrey	Pales	apalesh@shutterfly.com	Female	162.18.171.235
19	Konstance	Longhi	klonghii@friendfeed.com	Female	251.78.234.118
20	Leyla	Curry	lcurryj@dropbox.com	Female	207.77.180.107
21	Vincenty	Coppeard	vcoppeardk@prnewswire.com	Male	127.144.182.48
22	Graehme	Twyning	gtwyningl@adobe.com	Male	250.123.15.166
23	Mathias	Tondeur	mtondeurm@cnbc.com	Male	30.194.99.49
24	Amelie	Antoons	aantoonsn@elpais.com	Female	57.188.242.197
25	Hirsch	Tyers	htyerso@bloomberg.com	Male	202.135.214.221

□ Connecting Databricks to ADLS Gen2

- Mount ADLS Gen2 storage in Databricks.



Mount Data

20 hours ago (16s)

```
dbutils.fs.mount(
  source = "wasbs://customerdata@adlspythoncelebal.blob.core.windows.net/",
  mount_point = "/mnt/customerdata",
  extra_configs = {"fs.azure.account.key.adlspythoncelebal.blob.core.windows.net": "pgWZg1IuT0UrnckMyZ+P4KbZb
  +gMj5PMkWfn0YxOr3xFiY8YV4FdWulOQaOo5efu+DsFQ+I3+K/x+AStINg+xA=="}
)
```

```
True
```

Verify Mount

20 hours ago (6s)

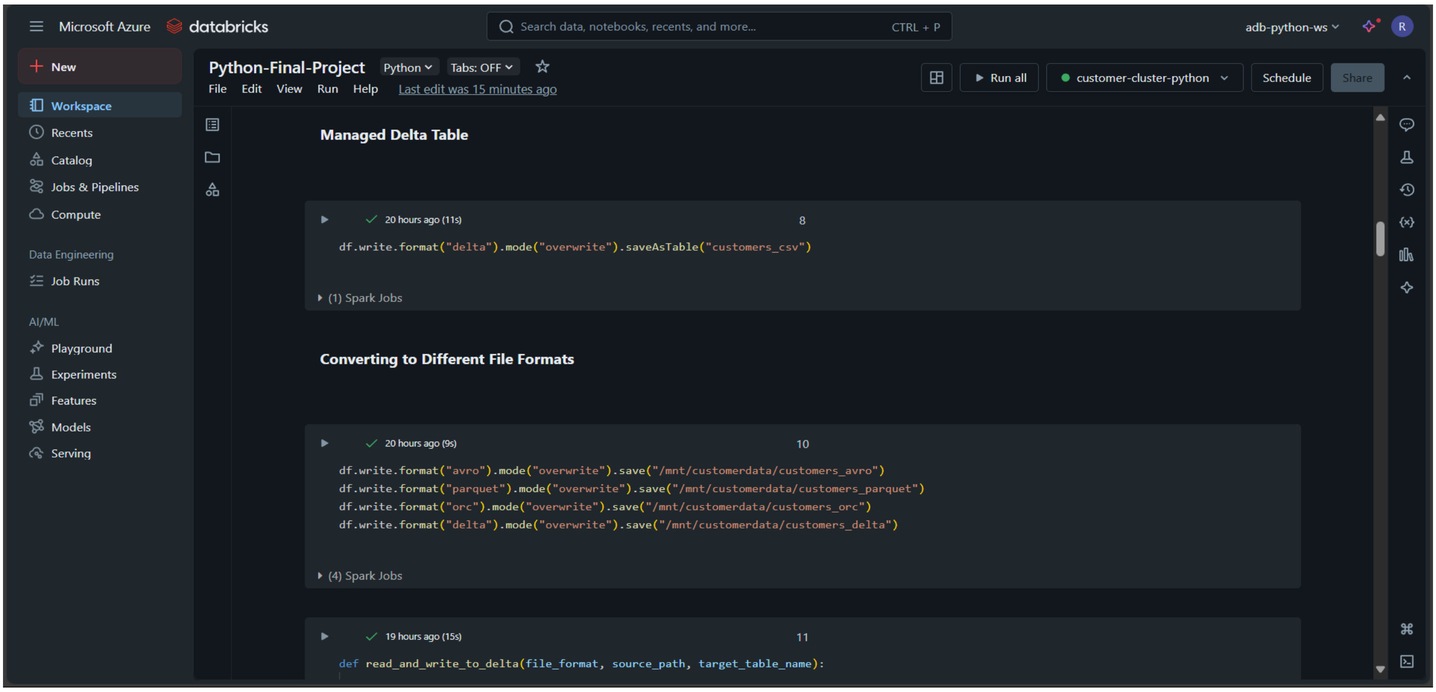
```
display(dbutils.fs.ls("/mnt/customerdata"))
```

(2) Spark Jobs

- Convert the sample data into various file formats including CSV, TSV, JSON, XML, XLSX, TXT, Avro, Parquet, ORC, and Delta.

□ Creating Managed Delta Tables for Various File Formats

- For Avro, Parquet, ORC, and Delta file formats, directly created managed Delta tables using the CTAS (Create Table As Select) SQL command to read from files and save as Delta tables.



Managed Delta Table

20 hours ago (11s)

```
df.write.format("delta").mode("overwrite").saveAsTable("customers_csv")
```

(1) Spark Jobs

Converting to Different File Formats

20 hours ago (9s)

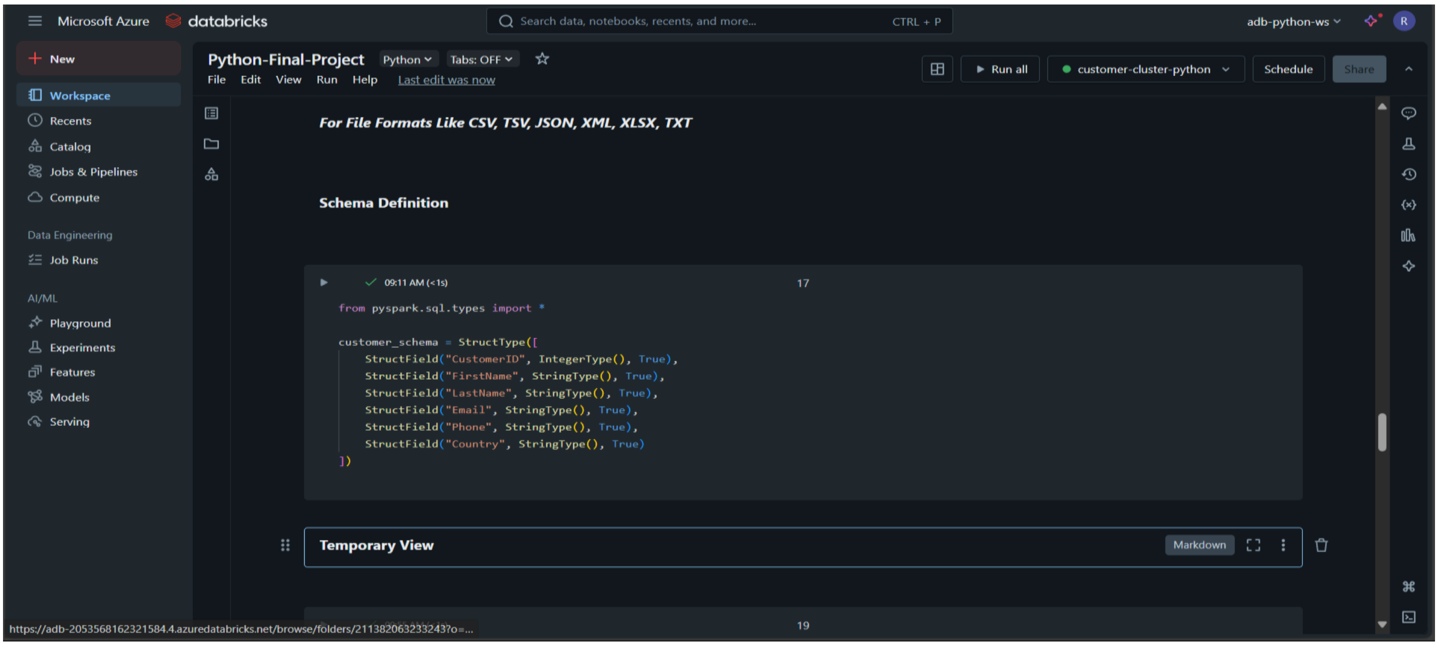
```
df.write.format("avro").mode("overwrite").save("/mnt/customerdata/customers_avro")
df.write.format("parquet").mode("overwrite").save("/mnt/customerdata/customers_parquet")
df.write.format("orc").mode("overwrite").save("/mnt/customerdata/customers_orc")
df.write.format("delta").mode("overwrite").save("/mnt/customerdata/customers_delta")
```

(4) Spark Jobs

19 hours ago (15s)

```
def read_and_write_to_delta(file_format, source_path, target_table_name):
```

- For CSV, TSV, JSON, XML, XLSX, and TXT file formats:
    - Read files using Spark with this schema and created temporary views.
    - Created managed Delta tables from these temporary views using CTAS, ensuring consistent structure.



For File Formats Like CSV, TSV, JSON, XML, XLSX, TXT

Schema Definition

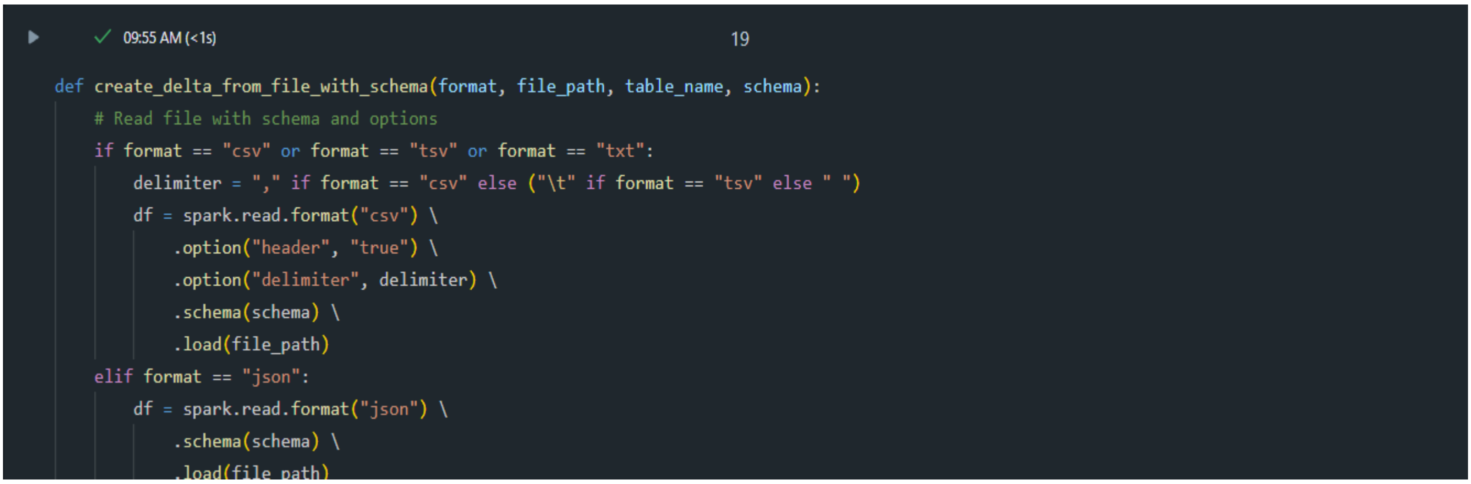
```python
from pyspark.sql.types import *

customer_schema = StructType([
    StructField("CustomerID", IntegerType(), True),
    StructField("FirstName", StringType(), True),
    StructField("LastName", StringType(), True),
    StructField("Email", StringType(), True),
    StructField("Phone", StringType(), True),
    StructField("Country", StringType(), True)
])
```

Temporary View

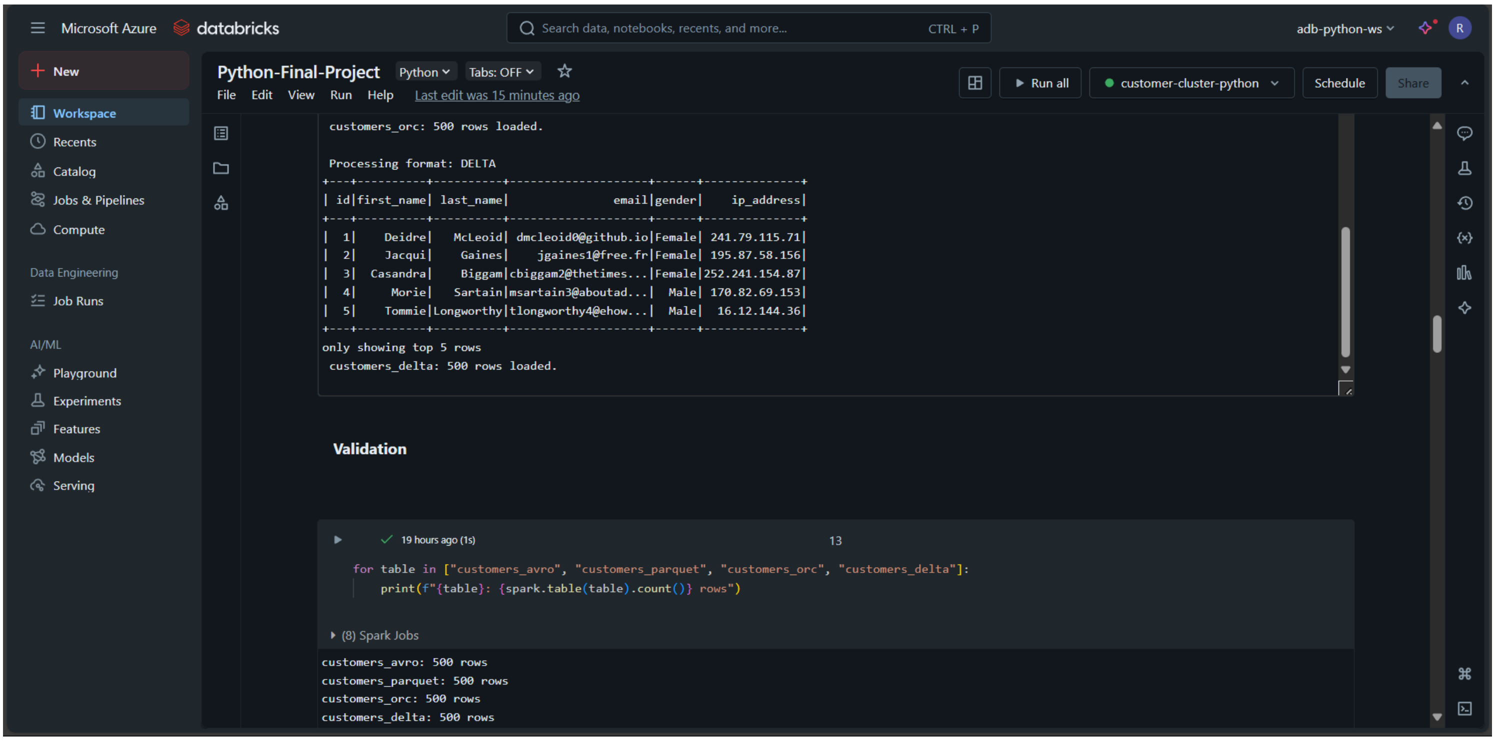
□ Data Validation and Code Optimization

- Implemented a generic Python function to automate reading, view creation, Delta table creation, and validation of row counts for each file format.



```python
def create_delta_from_file_with_schema(format, file_path, table_name, schema):
    # Read file with schema and options
    if format == "csv" or format == "tsv" or format == "txt":
        delimiter = "," if format == "csv" else ("\t" if format == "tsv" else " ")
        df = spark.read.format("csv") \
            .option("header", "true") \
            .option("delimiter", delimiter) \
            .schema(schema) \
            .load(file_path)
    elif format == "json":
        df = spark.read.format("json") \
            .schema(schema) \
            .load(file_path)
```

- Verified that each Delta table contains exactly 500 customer records to ensure data consistency and completeness.



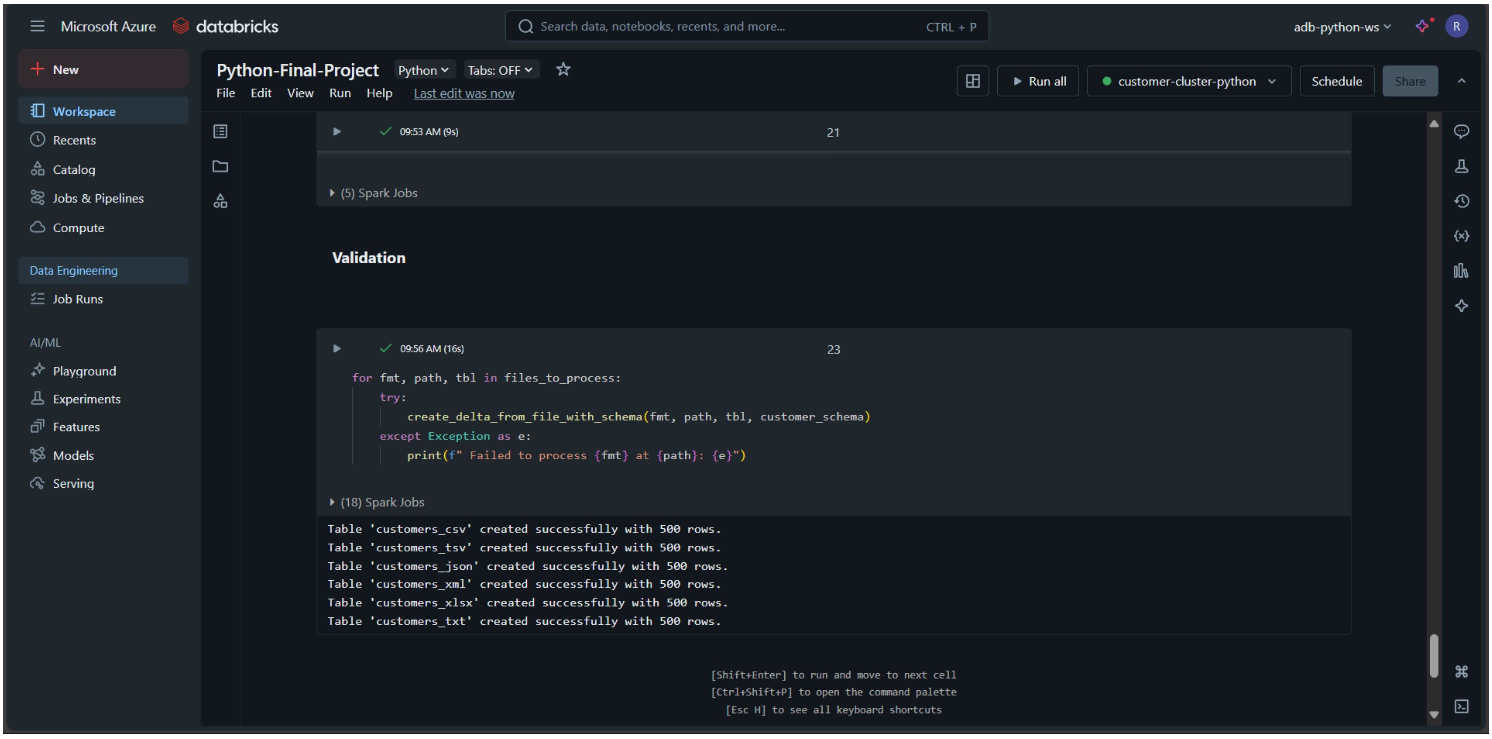
```
customers_orc: 500 rows loaded.

Processing format: DELTA
+---+----------+----------+--------------------+------+--------------+
| id|first_name| last_name|               email|gender|    ip_address|
+---+----------+----------+--------------------+------+--------------+
|  1|    Deidre|   McLeoid| dmcleoid0@github.io|Female| 241.79.115.71|
|  2|    Jacqui|    Gaines|    jgaines1@free.fr|Female| 195.87.58.156|
|  3|  Casandra|    Biggam|cbiggam2@thetimes...|Female|252.241.154.87|
|  4|     Morie|   Sartain|msartain3@aboutad...|  Male| 170.82.69.153|
|  5|    Tommie|Longworthy|tlongworthy4@ehow...|  Male|  16.12.144.36|
+---+----------+----------+--------------------+------+--------------+
only showing top 5 rows
customers_delta: 500 rows loaded.
```

Validation

```
for table in ["customers_avro", "customers_parquet", "customers_orc", "customers_delta"]:
    print(f"{table}: {spark.table(table).count()} rows")
```

```
customers_avro: 500 rows
customers_parquet: 500 rows
customers_orc: 500 rows
customers_delta: 500 rows
```



Validation

```
for fmt, path, tbl in files_to_process:
    try:
        create_delta_from_file_with_schema(fmt, path, tbl, customer_schema)
    except Exception as e:
        print(f" Failed to process {fmt} at {path}: {e}")
```

```
Table 'customers_csv' created successfully with 500 rows.
Table 'customers_tsv' created successfully with 500 rows.
Table 'customers_json' created successfully with 500 rows.
Table 'customers_xml' created successfully with 500 rows.
Table 'customers_xlsx' created successfully with 500 rows.
Table 'customers_txt' created successfully with 500 rows.
```

```
[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts
```