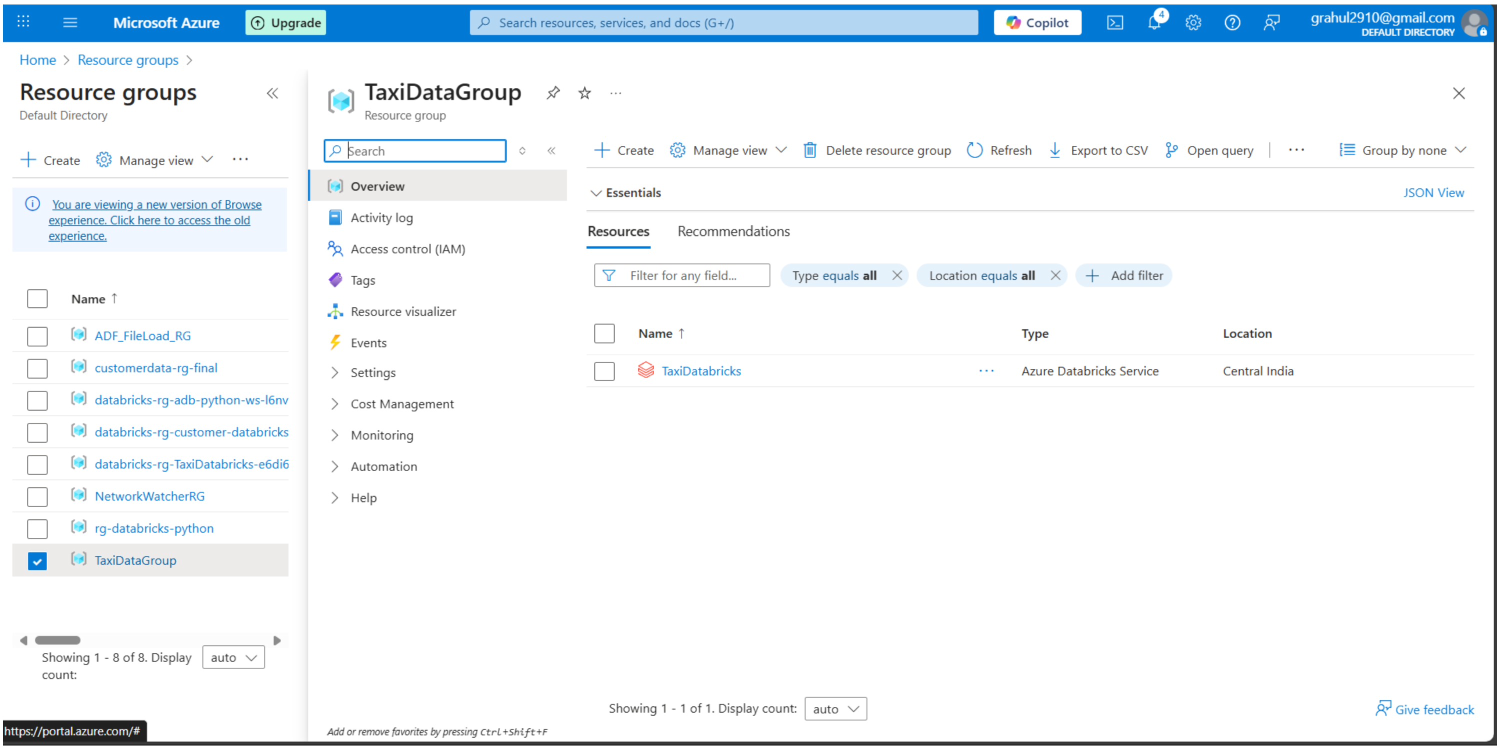
NYC Yellow Taxi Data Analysis using Azure and PySpark

1. Created Azure Resources:
   - Resource Group



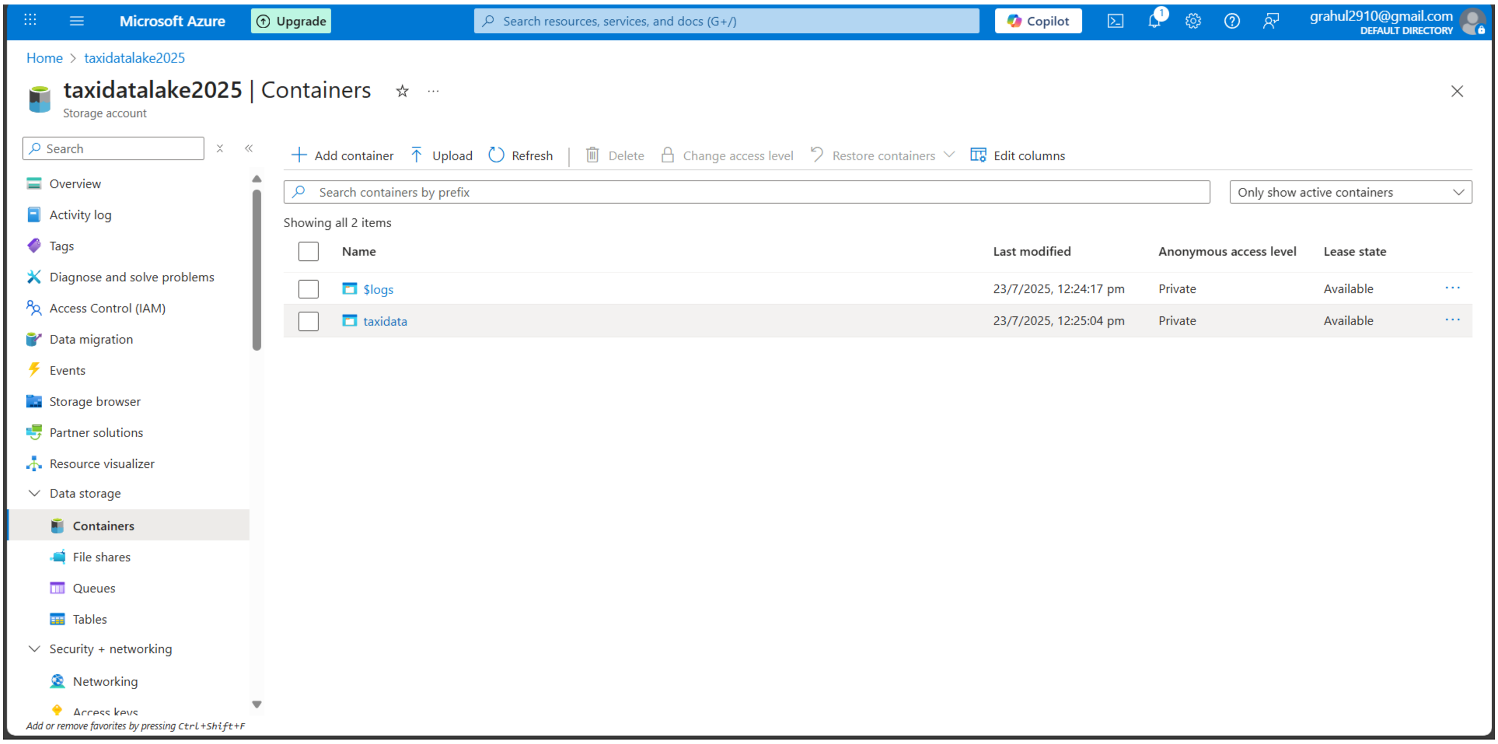
Name ↑
ADF_FileLoad_RG
customerdata-rg-final
databricks-rg-adb-python-ws-l6nv
databricks-rg-customer-databricks
databricks-rg-TaxiDatabricks-e6di6
NetworkWatcherRG
rg-databricks-python
☑ TaxiDataGroup

Showing 1 - 8 of 8. Display count: auto

Name ↑	Type	Location
TaxiDatabricks	Azure Databricks Service	Central India

Showing 1 - 1 of 1. Display count: auto

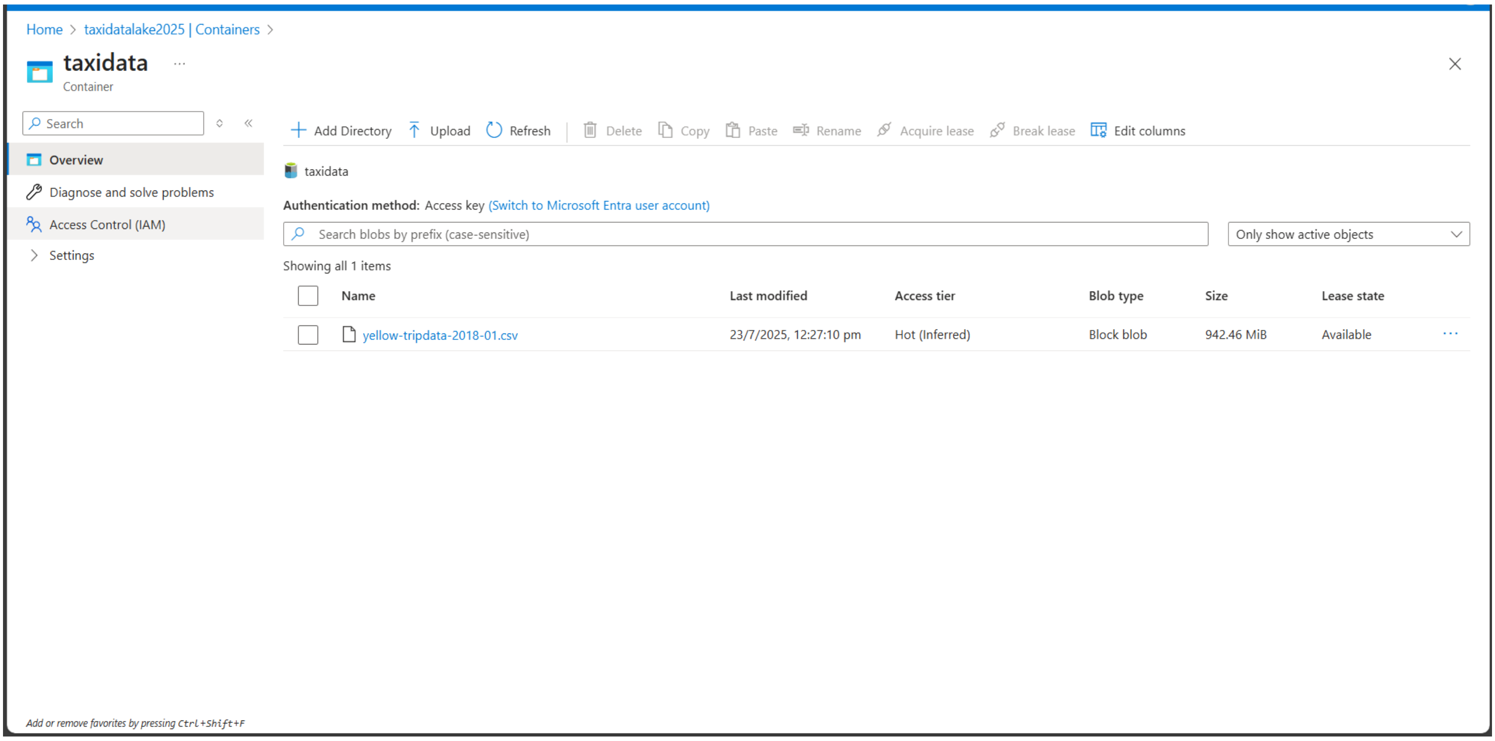
   - Storage Account with ADLS Gen2 enabled



Showing all 2 items

Name	Last modified	Anonymous access level	Lease state
$logs	23/7/2025, 12:24:17 pm	Private	Available
taxidata	23/7/2025, 12:25:04 pm	Private	Available

- Container named taxidata



Home > taxidatalake2025 | Containers >

taxidata

Container

taxidata

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive)

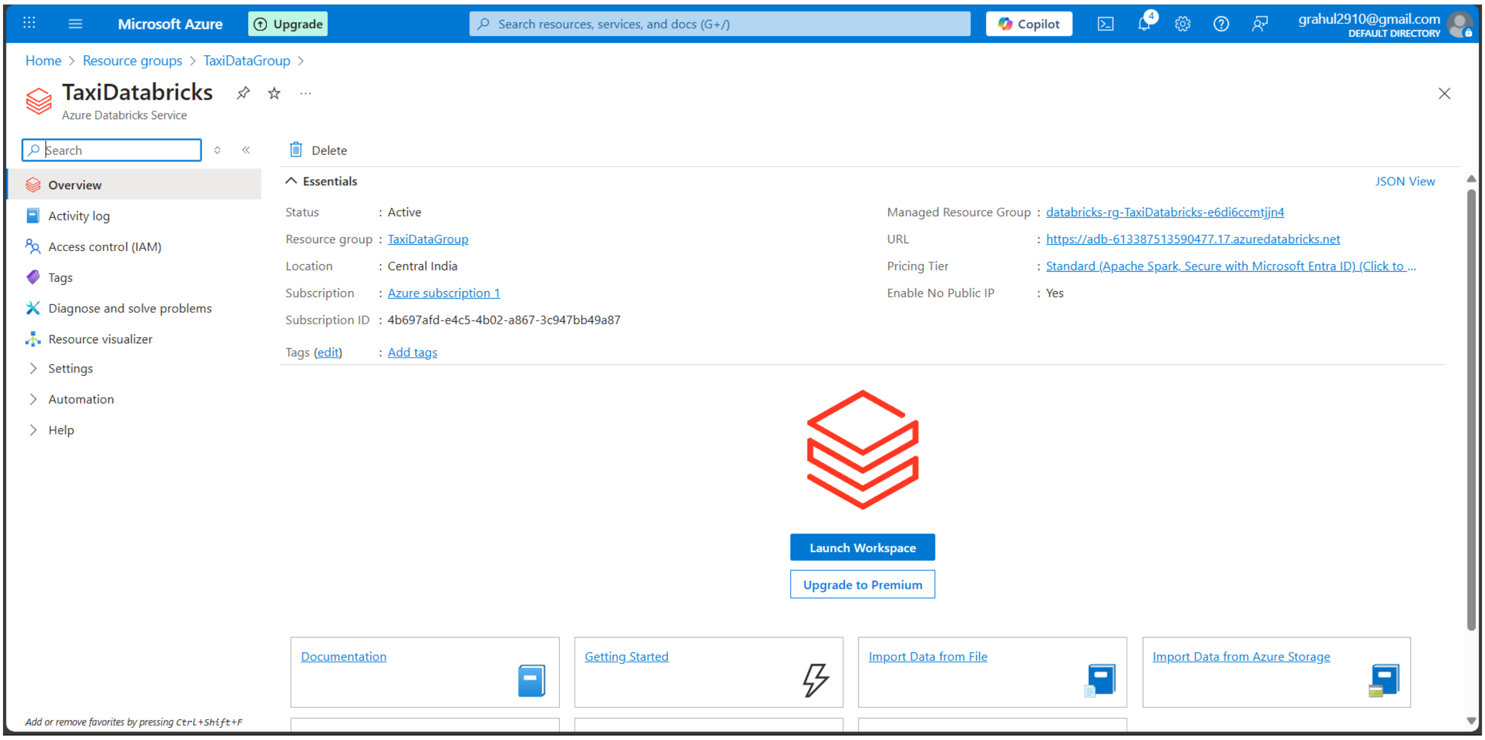
Only show active objects

Showing all 1 items

	Name	Last modified	Access tier	Blob type	Size	Lease state
□	yellow-tripdata-2018-01.csv	23/7/2025, 12:27:10 pm	Hot (Inferred)	Block blob	942.46 MiB	Available

Add or remove favorites by pressing Ctrl+Shift+F

- Azure Databricks workspace and cluster



Home > Resource groups > TaxiDataGroup >

TaxiDatabricks

Azure Databricks Service

Essentials

Status : Active

Resource group : TaxiDataGroup

Location : Central India

Subscription : Azure subscription 1

Subscription ID : 4b697afd-e4c5-4b02-a867-3c947bb49a87

Tags (edit) : Add tags

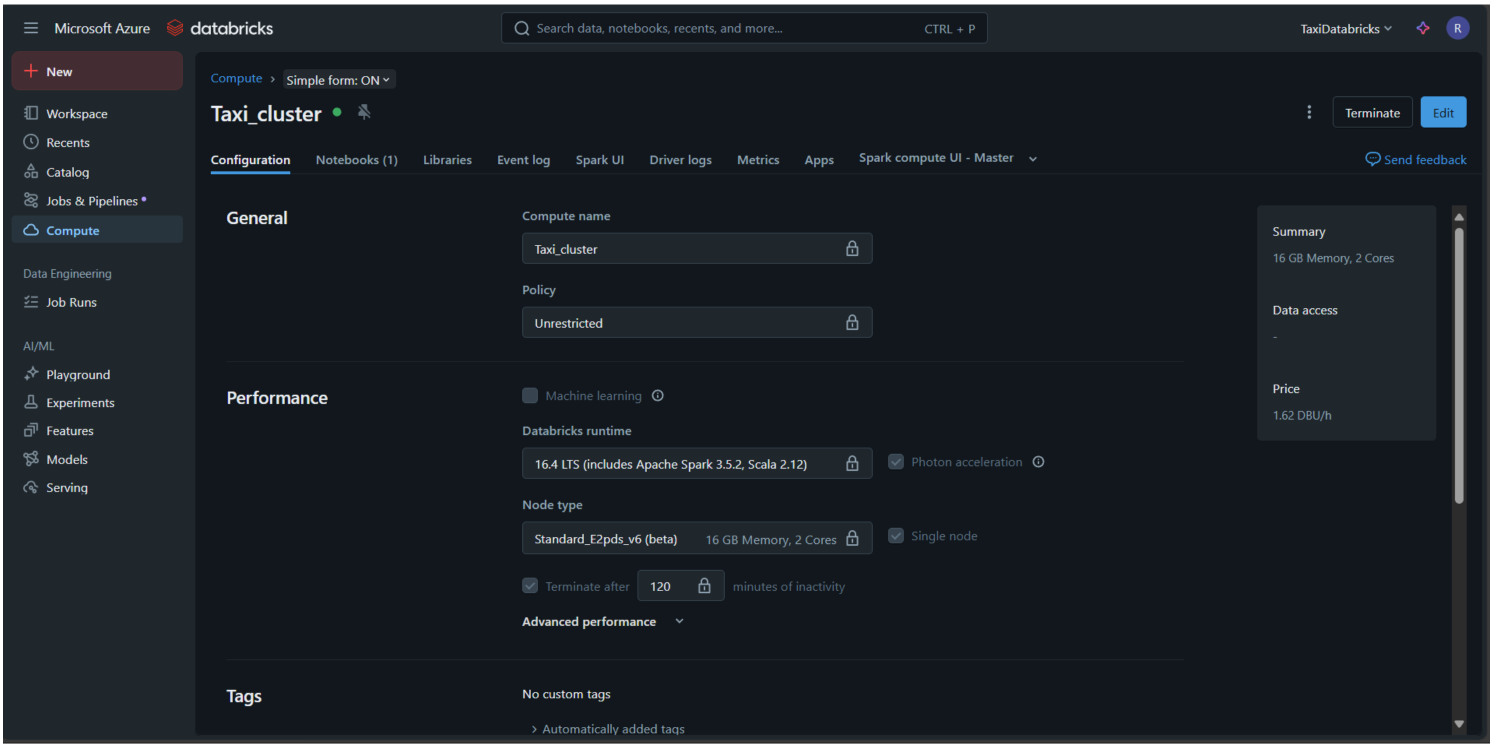
Managed Resource Group : databricks-rg-TaxiDatabricks-e6di6ccmtjin4

URL : https://adb-613387513590477.17.azuredatabricks.net

Pricing Tier : Standard (Apache Spark, Secure with Microsoft Entra ID) (Click to ...

Enable No Public IP : Yes

Add or remove favorites by pressing Ctrl+Shift+F



Taxi_cluster

General

Compute name: Taxi_cluster

Policy: Unrestricted

Performance

□ Machine learning

Databricks runtime: 16.4 LTS (includes Apache Spark 3.5.2, Scala 2.12)

☑ Photon acceleration

Node type: Standard_E2pds_v6 (beta) 16 GB Memory, 2 Cores

☑ Single node

☑ Terminate after 120 minutes of inactivity

Advanced performance

Tags

No custom tags

› Automatically added tags

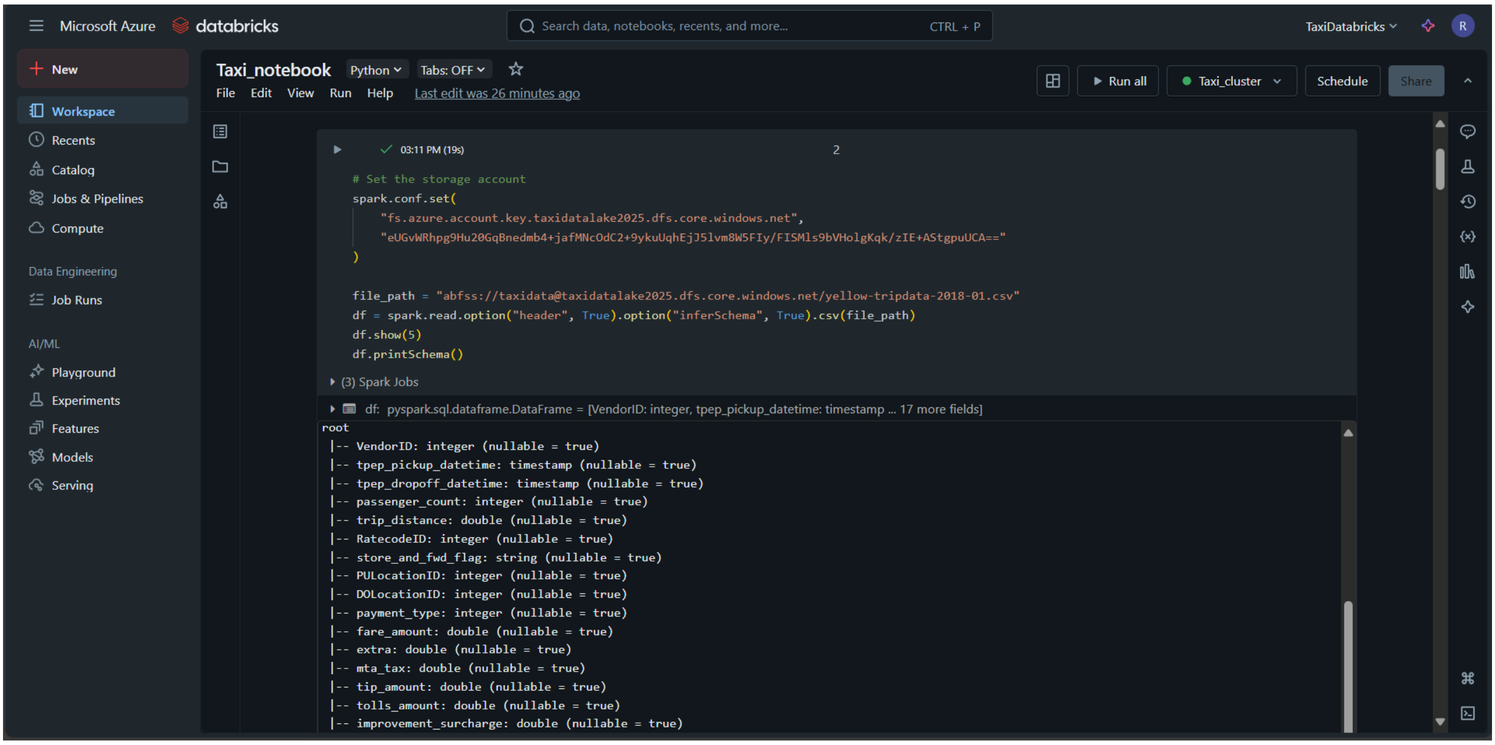
Summary: 16 GB Memory, 2 Cores

Data access: -

Price: 1.62 DBU/h

2. Data Upload:

    ○ Uploaded the dataset yellow_tripdata_2018-01.csv into the taxidata container in the storage account.



Taxi_notebook

03:11 PM (19s) 2

```
# Set the storage account
spark.conf.set(
    "fs.azure.account.key.taxidatalake2025.dfs.core.windows.net",
    "eUGvWRhpg9Hu20GqBnedmb4+jafMNcOdC2+9ykuUqhEjJ5lvm8W5FIy/FISMls9bVHolgKqk/zIE+AStgpuUCA=="
)

file_path = "abfss://taxidata@taxidatalake2025.dfs.core.windows.net/yellow-tripdata-2018-01.csv"
df = spark.read.option("header", True).option("inferSchema", True).csv(file_path)
df.show(5)
df.printSchema()
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [VendorID: integer, tpep_pickup_datetime: timestamp ... 17 more fields]

```
root
 |-- VendorID: integer (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- tpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: integer (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: integer (nullable = true)
 |-- DOLocationID: integer (nullable = true)
 |-- payment_type: integer (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
```

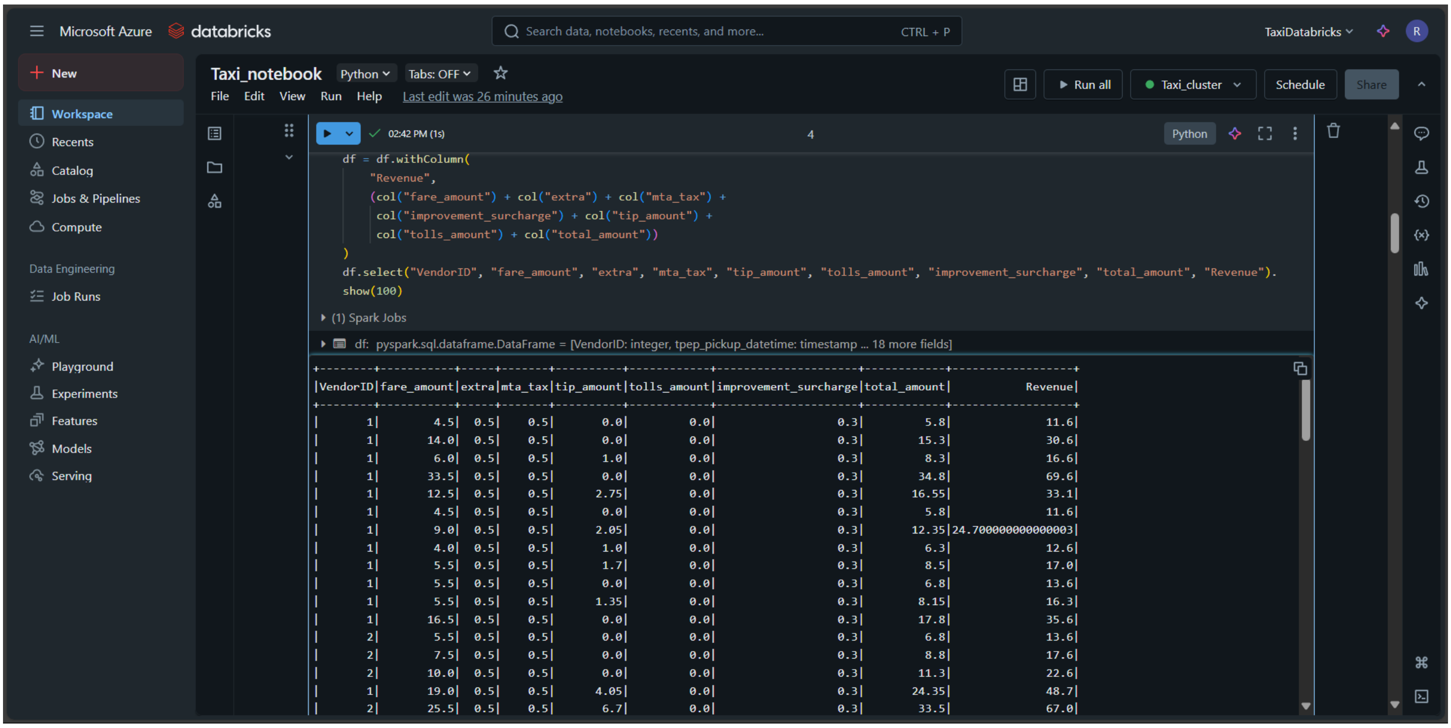
3. Databricks Notebook Setup:

    ○ Connected Databricks to the storage using ABFSS protocol and OAuth credentials.

- Read the CSV file using an explicitly defined schema and converted date columns (tpep_pickup_datetime, tpep_dropoff_datetime) to timestamps.

Query 1: Top Pickup Locations

- Grouped data by PULocationID.
- Summed the number of passengers per pickup location.

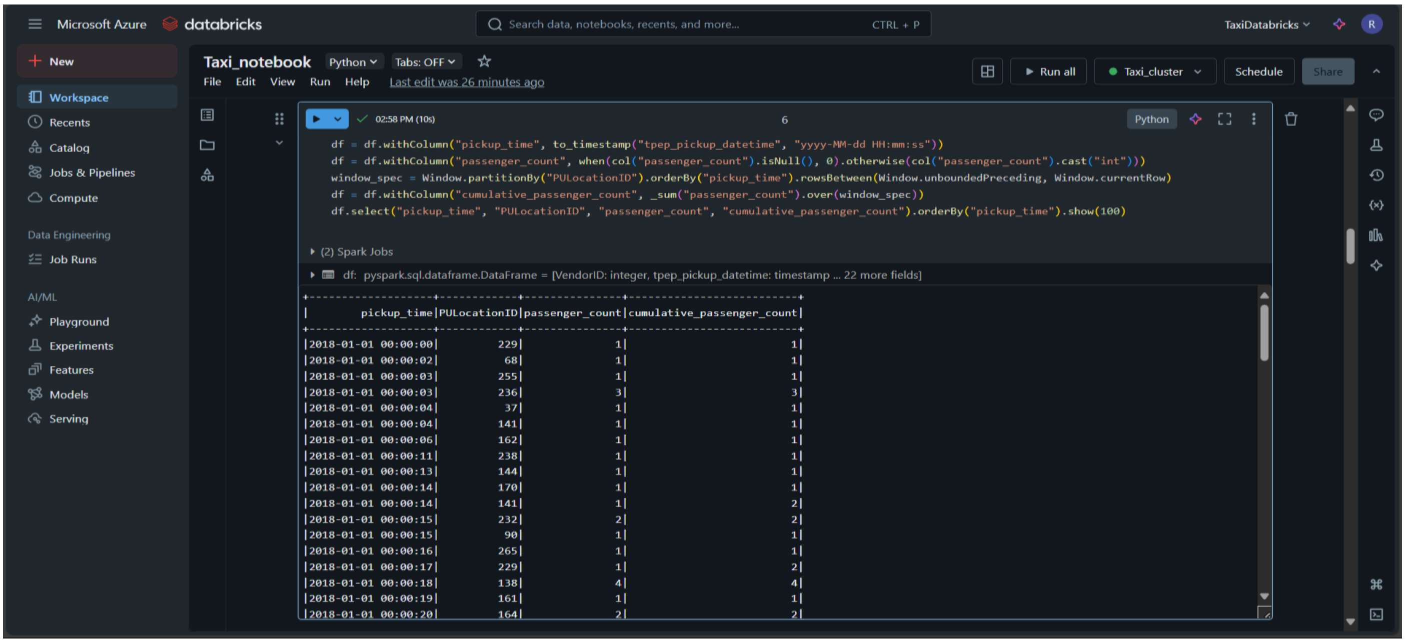


```
df = df.withColumn(
    "Revenue",
    (col("fare_amount") + col("extra") + col("mta_tax") +
     col("improvement_surcharge") + col("tip_amount") +
     col("tolls_amount") + col("total_amount"))
)
df.select("VendorID", "fare_amount", "extra", "mta_tax", "tip_amount", "tolls_amount", "improvement_surcharge", "total_amount", "Revenue").
show(100)
```

```
▸ (1) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [VendorID: integer, tpep_pickup_datetime: timestamp ... 18 more fields]
+--------+-----------+-----+-------+----------+------------+---------------------+------------+------------------+
|VendorID|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|           Revenue|
+--------+-----------+-----+-------+----------+------------+---------------------+------------+------------------+
|       1|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|              11.6|
|       1|       14.0|  0.5|    0.5|       0.0|         0.0|                  0.3|        15.3|              30.6|
|       1|        6.0|  0.5|    0.5|       1.0|         0.0|                  0.3|         8.3|              16.6|
|       1|       33.5|  0.5|    0.5|       0.0|         0.0|                  0.3|        34.8|              69.6|
|       1|       12.5|  0.5|    0.5|      2.75|         0.0|                  0.3|       16.55|              33.1|
|       1|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|              11.6|
|       1|        9.0|  0.5|    0.5|      2.05|         0.0|                  0.3|       12.35|24.700000000000003|
|       1|        4.0|  0.5|    0.5|       1.0|         0.0|                  0.3|         6.3|              12.6|
|       1|        5.5|  0.5|    0.5|       1.7|         0.0|                  0.3|         8.5|              17.0|
|       1|        5.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         6.8|              13.6|
|       1|        5.5|  0.5|    0.5|      1.35|         0.0|                  0.3|        8.15|              16.3|
|       1|       16.5|  0.5|    0.5|       0.0|         0.0|                  0.3|        17.8|              35.6|
|       2|        5.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         6.8|              13.6|
|       2|        7.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         8.8|              17.6|
|       2|       10.0|  0.5|    0.5|       0.0|         0.0|                  0.3|        11.3|              22.6|
|       1|       19.0|  0.5|    0.5|      4.05|         0.0|                  0.3|       24.35|              48.7|
|       2|       25.5|  0.5|    0.5|       6.7|         0.0|                  0.3|        33.5|              67.0|
```

Query 2: Top Dropoff Locations

- Grouped data by DOLocationID.
- Aggregated the number of passengers per dropoff location.
- Sorted to identify most common drop-off zones.

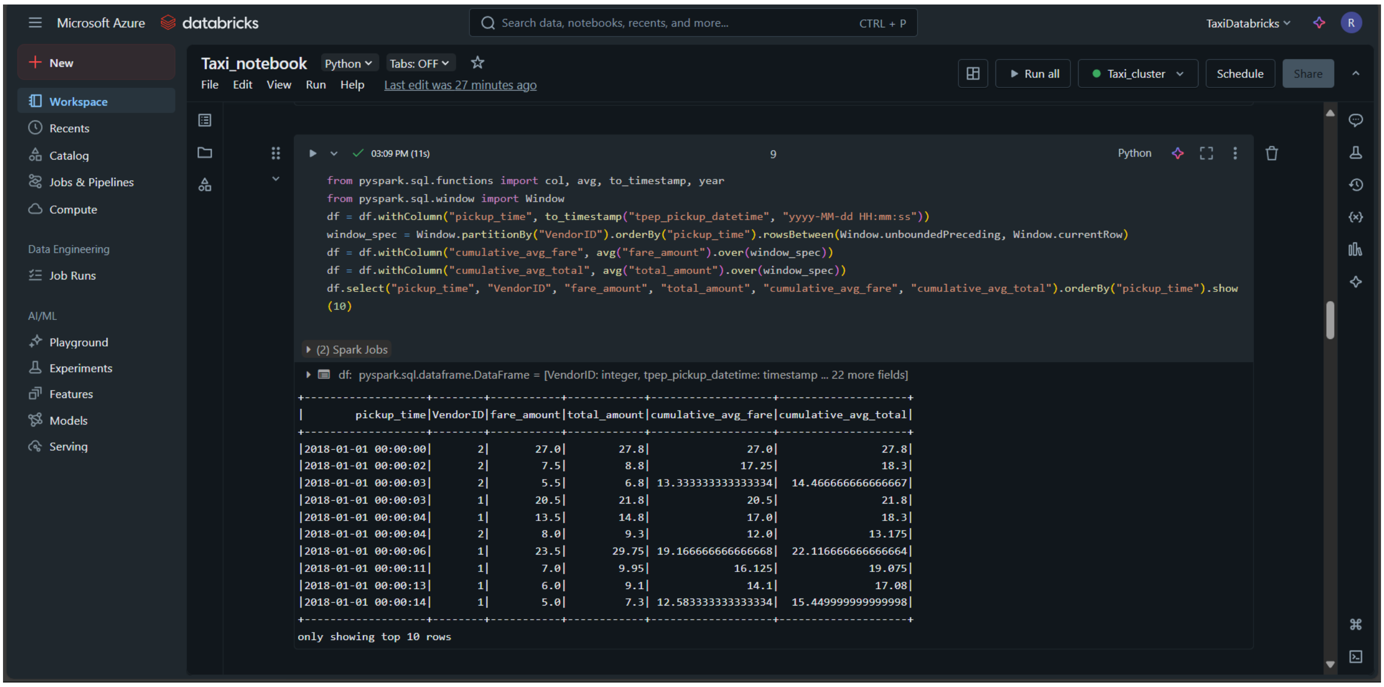


```
df = df.withColumn("pickup_time", to_timestamp("tpep_pickup_datetime", "yyyy-MM-dd HH:mm:ss"))
df = df.withColumn("passenger_count", when(col("passenger_count").isNull(), 0).otherwise(col("passenger_count").cast("int")))
window_spec = Window.partitionBy("PULocationID").orderBy("pickup_time").rowsBetween(Window.unboundedPreceding, Window.currentRow)
df = df.withColumn("cumulative_passenger_count", _sum("passenger_count").over(window_spec))
df.select("pickup_time", "PULocationID", "passenger_count", "cumulative_passenger_count").orderBy("pickup_time").show(100)
```

```
▸ (2) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [VendorID: integer, tpep_pickup_datetime: timestamp ... 22 more fields]
+-------------------+------------+---------------+--------------------------+
|        pickup_time|PULocationID|passenger_count|cumulative_passenger_count|
+-------------------+------------+---------------+--------------------------+
|2018-01-01 00:00:00|         229|              1|                         1|
|2018-01-01 00:00:02|          68|              1|                         1|
|2018-01-01 00:00:03|         255|              1|                         1|
|2018-01-01 00:00:03|         236|              3|                         3|
|2018-01-01 00:00:04|          37|              1|                         1|
|2018-01-01 00:00:04|         141|              1|                         1|
|2018-01-01 00:00:06|         162|              1|                         1|
|2018-01-01 00:00:11|         238|              1|                         1|
|2018-01-01 00:00:13|         144|              1|                         1|
|2018-01-01 00:00:14|         170|              1|                         1|
|2018-01-01 00:00:14|         141|              1|                         2|
|2018-01-01 00:00:15|         232|              2|                         2|
|2018-01-01 00:00:15|          90|              1|                         1|
|2018-01-01 00:00:16|         265|              1|                         1|
|2018-01-01 00:00:17|         229|              1|                         2|
|2018-01-01 00:00:18|         138|              4|                         4|
|2018-01-01 00:00:19|         161|              1|                         1|
|2018-01-01 00:00:20|         164|              2|                         2|
```

Query 3: Revenue Generated by Each Vendor

- Created a derived column Revenue by summing:
    - fare_amount + extra + mta_tax + tip_amount + tolls_amount + improvement_surcharge + congestion_surcharge
- Grouped the dataset by VendorID.
- Aggregated total revenue per vendor.

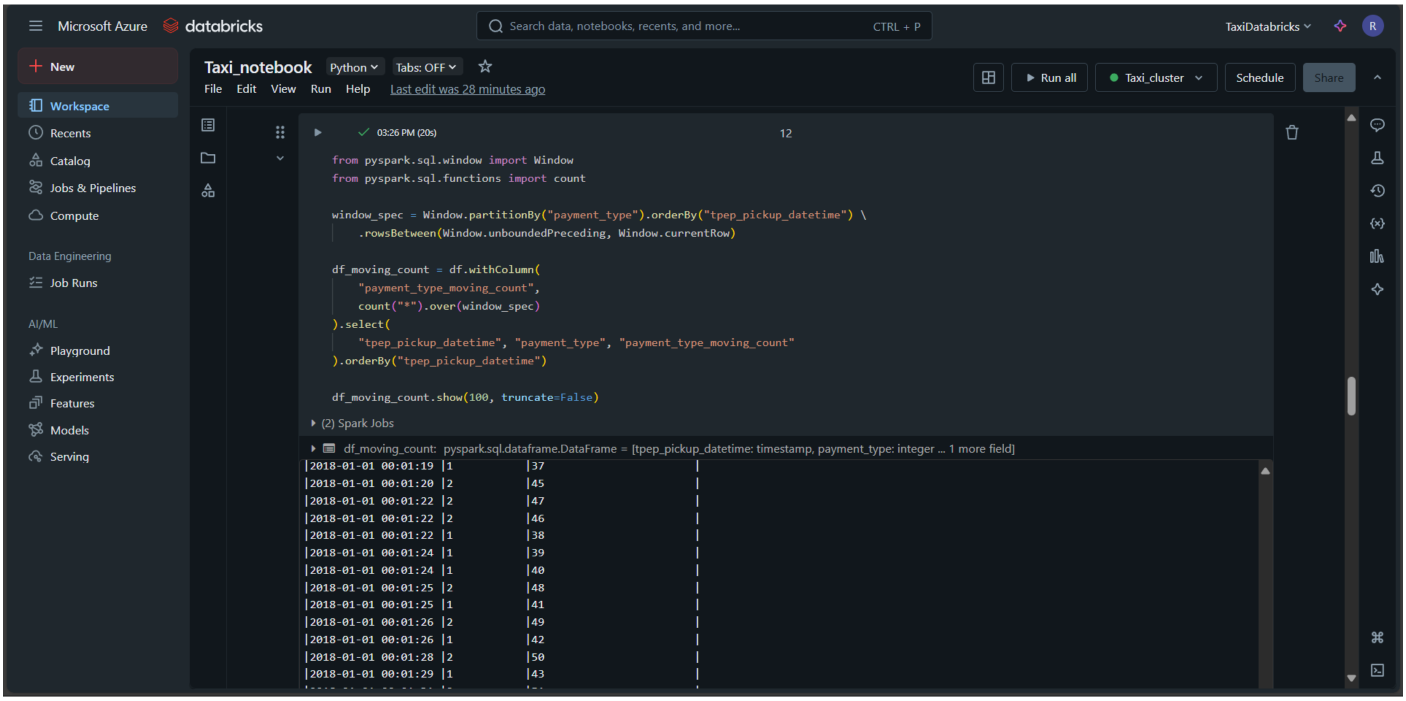


```
from pyspark.sql.functions import col, avg, to_timestamp, year
from pyspark.sql.window import Window
df = df.withColumn("pickup_time", to_timestamp("tpep_pickup_datetime", "yyyy-MM-dd HH:mm:ss"))
window_spec = Window.partitionBy("VendorID").orderBy("pickup_time").rowsBetween(Window.unboundedPreceding, Window.currentRow)
df = df.withColumn("cumulative_avg_fare", avg("fare_amount").over(window_spec))
df = df.withColumn("cumulative_avg_total", avg("total_amount").over(window_spec))
df.select("pickup_time", "VendorID", "fare_amount", "total_amount", "cumulative_avg_fare", "cumulative_avg_total").orderBy("pickup_time").show
(10)
```

```
df:  pyspark.sql.dataframe.DataFrame = [VendorID: integer, tpep_pickup_datetime: timestamp ... 22 more fields]
+-------------------+--------+-----------+------------+-------------------+--------------------+
|        pickup_time|VendorID|fare_amount|total_amount|cumulative_avg_fare|cumulative_avg_total|
+-------------------+--------+-----------+------------+-------------------+--------------------+
|2018-01-01 00:00:00|       2|       27.0|        27.8|               27.0|                27.8|
|2018-01-01 00:00:02|       2|        7.5|         8.8|              17.25|                18.3|
|2018-01-01 00:00:03|       2|        5.5|         6.8| 13.333333333333334|  14.466666666666667|
|2018-01-01 00:00:03|       1|       20.5|        21.8|               20.5|                21.8|
|2018-01-01 00:00:04|       1|       13.5|        14.8|               17.0|                18.3|
|2018-01-01 00:00:04|       2|        8.0|         9.3|               12.0|              13.175|
|2018-01-01 00:00:06|       1|       23.5|       29.75| 19.166666666666668|  22.116666666666664|
|2018-01-01 00:00:11|       1|        7.0|        9.95|             16.125|              19.075|
|2018-01-01 00:00:13|       1|        6.0|         9.1|               14.1|               17.08|
|2018-01-01 00:00:14|       1|        5.0|         7.3| 12.583333333333334|  15.449999999999998|
+-------------------+--------+-----------+------------+-------------------+--------------------+
only showing top 10 rows
```

Query 4: Moving Count of Payments by Payment Mode

- Used a Window function to calculate a rolling count of rides per payment_type, ordered by pickup timestamp.
- Enabled tracking of trends in payment methods over time.

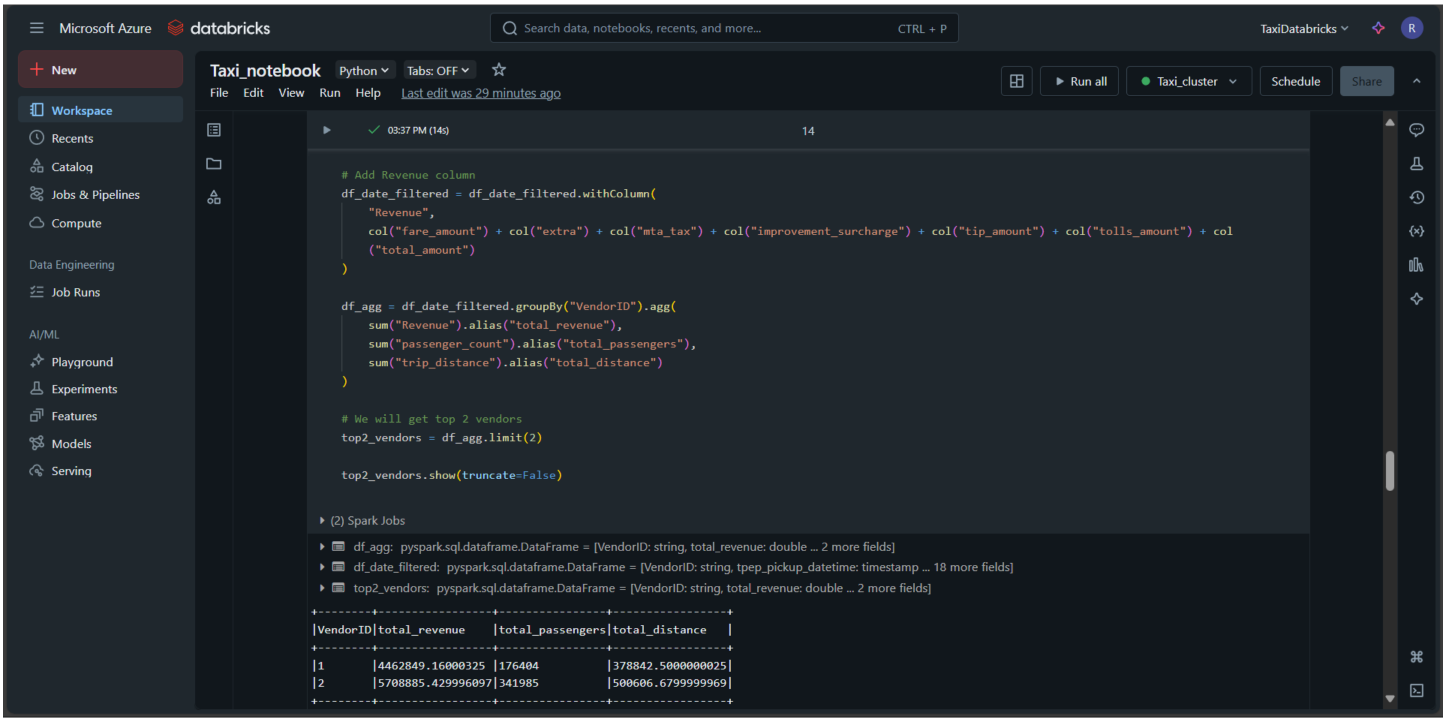


```
from pyspark.sql.window import Window
from pyspark.sql.functions import count

window_spec = Window.partitionBy("payment_type").orderBy("tpep_pickup_datetime") \
    .rowsBetween(Window.unboundedPreceding, Window.currentRow)

df_moving_count = df.withColumn(
    "payment_type_moving_count",
    count("*").over(window_spec)
).select(
    "tpep_pickup_datetime", "payment_type", "payment_type_moving_count"
).orderBy("tpep_pickup_datetime")

df_moving_count.show(100, truncate=False)
```

```
df_moving_count:  pyspark.sql.dataframe.DataFrame = [tpep_pickup_datetime: timestamp, payment_type: integer ... 1 more field]
|2018-01-01 00:01:19 |1           |37                       |
|2018-01-01 00:01:20 |2           |45                       |
|2018-01-01 00:01:22 |2           |47                       |
|2018-01-01 00:01:22 |2           |46                       |
|2018-01-01 00:01:22 |1           |38                       |
|2018-01-01 00:01:24 |1           |39                       |
|2018-01-01 00:01:24 |1           |40                       |
|2018-01-01 00:01:25 |2           |48                       |
|2018-01-01 00:01:25 |1           |41                       |
|2018-01-01 00:01:26 |2           |49                       |
|2018-01-01 00:01:26 |1           |42                       |
|2018-01-01 00:01:28 |2           |50                       |
|2018-01-01 00:01:29 |1           |43                       |
```

Query 5: Top 2 Highest Earning Vendors on a Particular Date

- Filtered rides from January 15, 2018.
- Computed Revenue per row as in Query 3.
- Aggregated total revenue, passenger count, and trip distance by vendor.
- Sorted and selected the top 2 vendors by revenue.

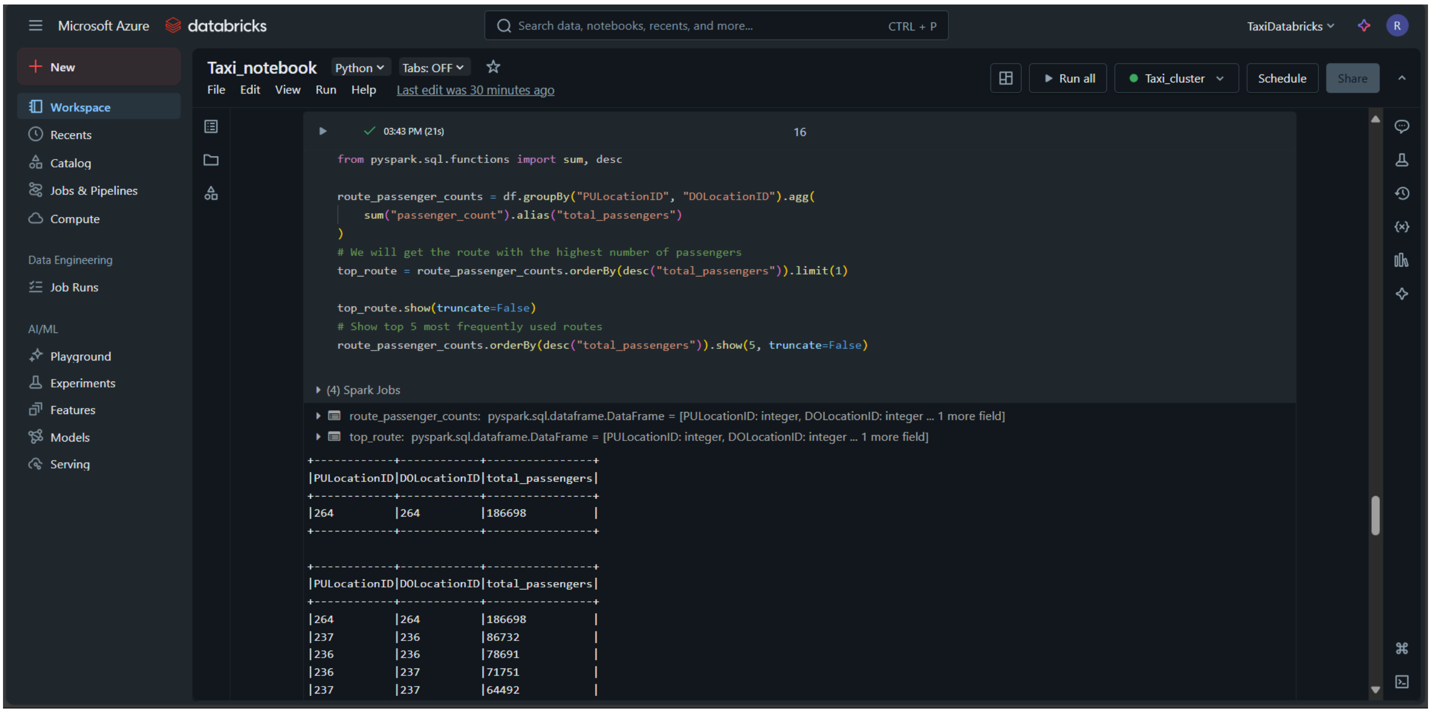


```
# Add Revenue column
df_date_filtered = df_date_filtered.withColumn(
    "Revenue",
    col("fare_amount") + col("extra") + col("mta_tax") + col("improvement_surcharge") + col("tip_amount") + col("tolls_amount") + col
    ("total_amount")
)

df_agg = df_date_filtered.groupBy("VendorID").agg(
    sum("Revenue").alias("total_revenue"),
    sum("passenger_count").alias("total_passengers"),
    sum("trip_distance").alias("total_distance")
)

# We will get top 2 vendors
top2_vendors = df_agg.limit(2)

top2_vendors.show(truncate=False)
```

```
(2) Spark Jobs
df_agg:  pyspark.sql.dataframe.DataFrame = [VendorID: string, total_revenue: double ... 2 more fields]
df_date_filtered:  pyspark.sql.dataframe.DataFrame = [VendorID: string, tpep_pickup_datetime: timestamp ... 18 more fields]
top2_vendors:  pyspark.sql.dataframe.DataFrame = [VendorID: string, total_revenue: double ... 2 more fields]
+--------+-----------------+----------------+-----------------+
|VendorID|total_revenue    |total_passengers|total_distance   |
+--------+-----------------+----------------+-----------------+
|1       |4462849.16000325 |176404          |378842.5000000025|
|2       |5708885.429996097|341985          |500606.6799999969|
+--------+-----------------+----------------+-----------------+
```

Query 6: Route with Most Passengers

- Grouped by both PULocationID and DOLocationID to define a “route.”
- Summed passenger_count for each route.
- Identified the route with the highest total passengers.

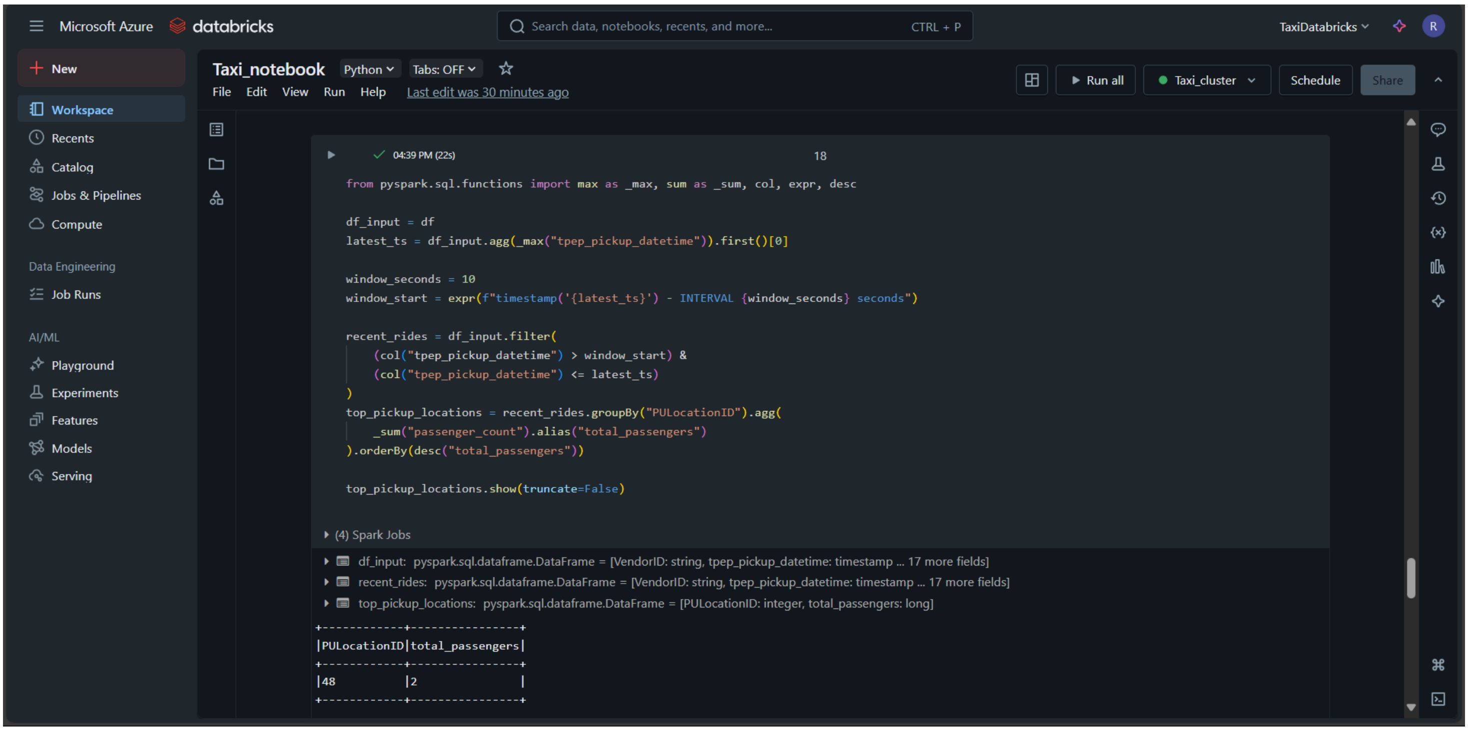


```
from pyspark.sql.functions import sum, desc

route_passenger_counts = df.groupBy("PULocationID", "DOLocationID").agg(
    sum("passenger_count").alias("total_passengers")
)
# We will get the route with the highest number of passengers
top_route = route_passenger_counts.orderBy(desc("total_passengers")).limit(1)

top_route.show(truncate=False)
# Show top 5 most frequently used routes
route_passenger_counts.orderBy(desc("total_passengers")).show(5, truncate=False)
```

```
(4) Spark Jobs
route_passenger_counts:  pyspark.sql.dataframe.DataFrame = [PULocationID: integer, DOLocationID: integer ... 1 more field]
top_route:  pyspark.sql.dataframe.DataFrame = [PULocationID: integer, DOLocationID: integer ... 1 more field]
+------------+------------+----------------+
|PULocationID|DOLocationID|total_passengers|
+------------+------------+----------------+
|264         |264         |186698          |
+------------+------------+----------------+


+------------+------------+----------------+
|PULocationID|DOLocationID|total_passengers|
+------------+------------+----------------+
|264         |264         |186698          |
|237         |236         |86732           |
|236         |236         |78691           |
|236         |237         |71751           |
|237         |237         |64492           |
```

Query 7: Top Pickup Locations in the Last N Seconds

- Retrieved the maximum pickup timestamp from the dataset.
- Defined a time window (last 5 or 10 seconds).
- Filtered data to include only trips within that time frame.
- Aggregated and sorted pickup locations by passenger count to detect demand surges.



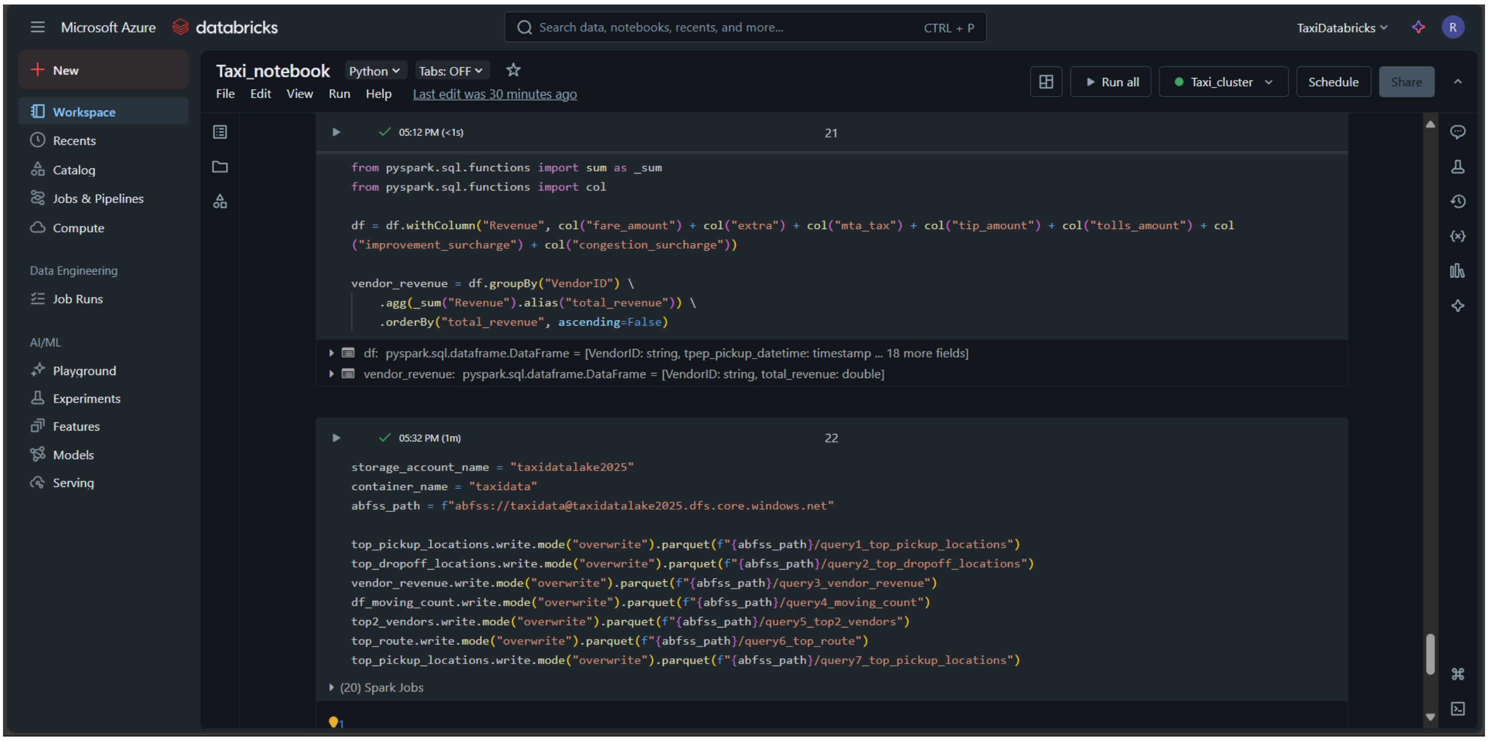
04:39 PM (22s) 18

```
from pyspark.sql.functions import max as _max, sum as _sum, col, expr, desc

df_input = df
latest_ts = df_input.agg(_max("tpep_pickup_datetime")).first()[0]

window_seconds = 10
window_start = expr(f"timestamp('{latest_ts}') - INTERVAL {window_seconds} seconds")

recent_rides = df_input.filter(
    (col("tpep_pickup_datetime") > window_start) &
    (col("tpep_pickup_datetime") <= latest_ts)
)
top_pickup_locations = recent_rides.groupBy("PULocationID").agg(
    _sum("passenger_count").alias("total_passengers")
).orderBy(desc("total_passengers"))

top_pickup_locations.show(truncate=False)
```

(4) Spark Jobs

```
df_input:  pyspark.sql.dataframe.DataFrame = [VendorID: string, tpep_pickup_datetime: timestamp ... 17 more fields]
recent_rides:  pyspark.sql.dataframe.DataFrame = [VendorID: string, tpep_pickup_datetime: timestamp ... 17 more fields]
top_pickup_locations:  pyspark.sql.dataframe.DataFrame = [PULocationID: integer, total_passengers: long]
+------------+----------------+
|PULocationID|total_passengers|
+------------+----------------+
|48          |2               |
+------------+----------------+
```

Output-



05:12 PM (<1s) 21

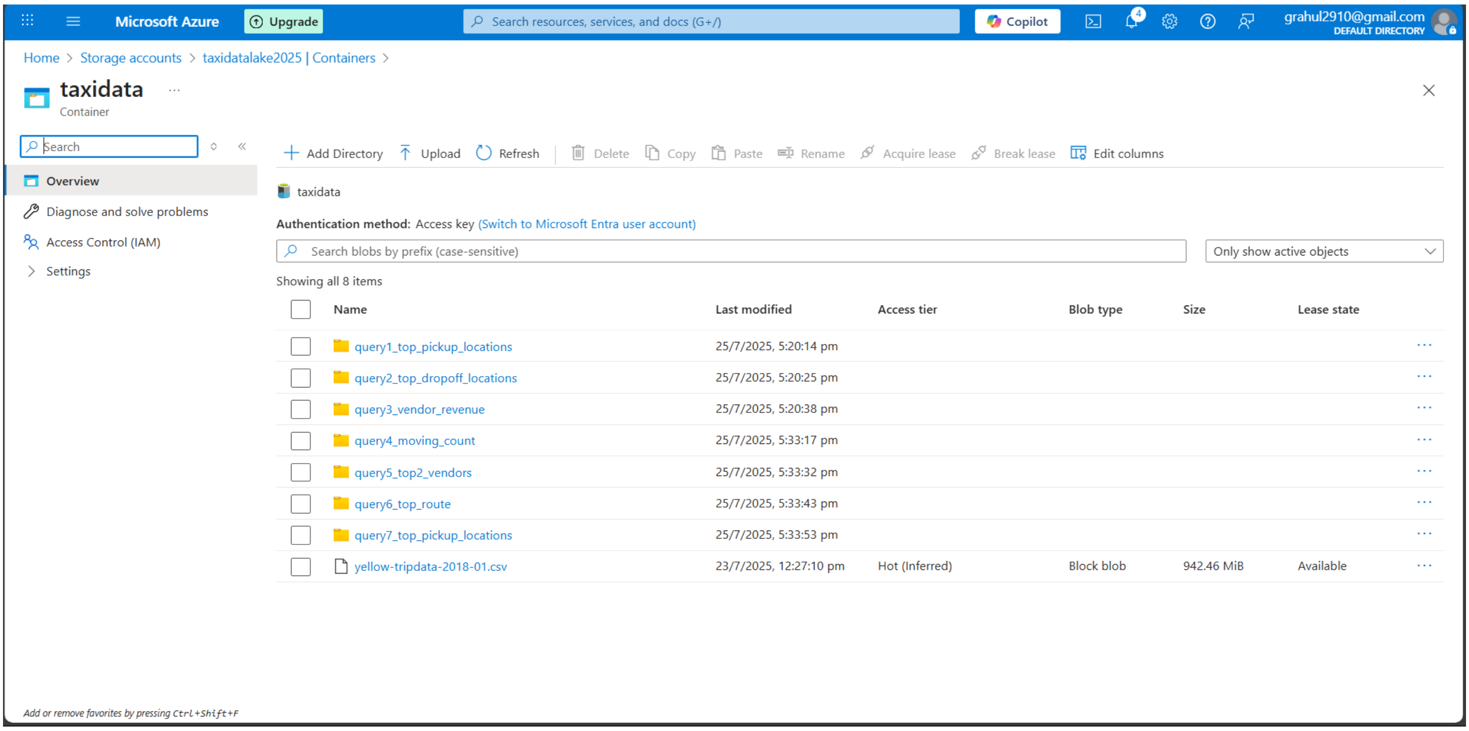
```
from pyspark.sql.functions import sum as _sum
from pyspark.sql.functions import col

df = df.withColumn("Revenue", col("fare_amount") + col("extra") + col("mta_tax") + col("tip_amount") + col("tolls_amount") + col
("improvement_surcharge") + col("congestion_surcharge"))

vendor_revenue = df.groupBy("VendorID") \
    .agg(_sum("Revenue").alias("total_revenue")) \
    .orderBy("total_revenue", ascending=False)
```

```
df:  pyspark.sql.dataframe.DataFrame = [VendorID: string, tpep_pickup_datetime: timestamp ... 18 more fields]
vendor_revenue:  pyspark.sql.dataframe.DataFrame = [VendorID: string, total_revenue: double]
```

05:32 PM (1m) 22

```
storage_account_name = "taxidatalake2025"
container_name = "taxidata"
abfss_path = f"abfss://taxidata@taxidatalake2025.dfs.core.windows.net"

top_pickup_locations.write.mode("overwrite").parquet(f"{abfss_path}/query1_top_pickup_locations")
top_dropoff_locations.write.mode("overwrite").parquet(f"{abfss_path}/query2_top_dropoff_locations")
vendor_revenue.write.mode("overwrite").parquet(f"{abfss_path}/query3_vendor_revenue")
df_moving_count.write.mode("overwrite").parquet(f"{abfss_path}/query4_moving_count")
top2_vendors.write.mode("overwrite").parquet(f"{abfss_path}/query5_top2_vendors")
top_route.write.mode("overwrite").parquet(f"{abfss_path}/query6_top_route")
top_pickup_locations.write.mode("overwrite").parquet(f"{abfss_path}/query7_top_pickup_locations")
```

(20) Spark Jobs



Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 8 items

Name	Last modified	Access tier	Blob type	Size	Lease state
query1_top_pickup_locations	25/7/2025, 5:20:14 pm				
query2_top_dropoff_locations	25/7/2025, 5:20:25 pm				
query3_vendor_revenue	25/7/2025, 5:20:38 pm				
query4_moving_count	25/7/2025, 5:33:17 pm				
query5_top2_vendors	25/7/2025, 5:33:32 pm				
query6_top_route	25/7/2025, 5:33:43 pm				
query7_top_pickup_locations	25/7/2025, 5:33:53 pm				
yellow-tripdata-2018-01.csv	23/7/2025, 12:27:10 pm	Hot (Inferred)	Block blob	942.46 MiB	Available