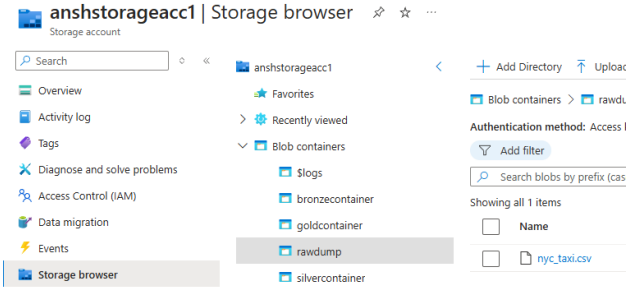
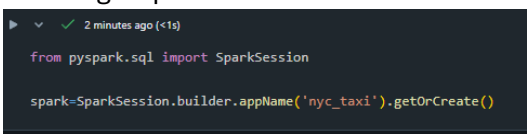
Ansh Ranjan
Databricks Case Study
Exercise 2 – Data Ingestion

TASK 1: Load a dataset into your Databricks using Spark

1. Our data resides in a ‘rawdump’ named container in our storage account.

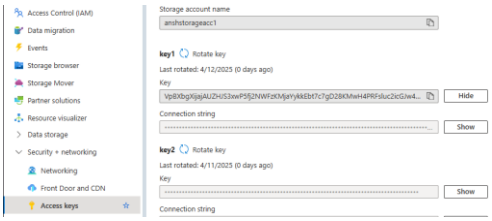


2. Creating a spark session

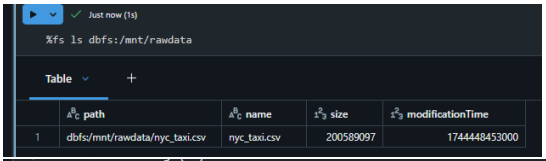


3. Now we have to mount our data point to our workspace.

You can get your access key by going to storage account > Security + networking > Access Keys

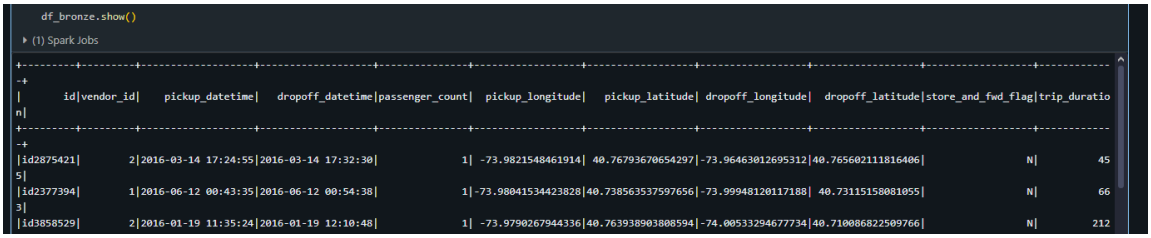


4. Reading data into a dataframe



TASK 2: Inspect the data

1. Displaying the data



2. Dataframe schema

```
df_bronze.printSchema()

root
 |-- id: string (nullable = true)
 |-- vendor_id: integer (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- trip_duration: integer (nullable = true)
```

3. Number of rows

```
df_bronze.count()

▸ (2) Spark Jobs

1458644
```

TASK 3: Perform data cleaning

1. Dropping null values from the df

```
df_bronze.dropna()

DataFrame[id: string, vendor_id: int, p
double, dropoff_latitude: double, store
```

Now there should not be any null values

```
from pyspark.sql.functions import col, sum, when

null_counts = df_bronze.select([
    sum(when(col(c).isNull(), 1).otherwise(0)).alias(c)
    for c in df_bronze.columns
])

null_counts.show()

▸ (2) Spark Jobs

▸ null_counts: pyspark.sql.dataframe.DataFrame = [id: long, vendor_id: long ... 9 more fields]

+---+---------+---------------+----------------+---------------+----------------+---------------+-----------------+----------------+------------------+-------------+
| id|vendor_id|pickup_datetime|dropoff_datetime|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|store_and_fwd_flag|trip_duration|
+---+---------+---------------+----------------+---------------+----------------+---------------+-----------------+----------------+------------------+-------------+
|  0|        0|              0|               0|              0|               0|              0|                0|               0|                 0|            0|
+---+---------+---------------+----------------+---------------+----------------+---------------+-----------------+----------------+------------------+-------------+
```

2. As we saw earlier, our pickup and dropoff columns are of double data type. We need to convert them into standard datetime format

```
from pyspark.sql.functions import to_timestamp

df_bronze_converted = df_bronze.withColumn("pickup_datetime", to_timestamp("pickup_datetime", "yyyy-MM-dd HH:mm:ss")) \
                               .withColumn("dropoff_datetime", to_timestamp("dropoff_datetime", "yyyy-MM-dd HH:mm:ss"))

df_bronze_converted.printSchema()

▸ df_bronze_converted: pyspark.sql.dataframe.DataFrame = [id: string, vendor_id: integer ... 9 more fields]

root
 |-- id: string (nullable = true)
 |-- vendor_id: integer (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- trip_duration: integer (nullable = true)
```

3. Checking for any rows where dropoff time might be before pickup time for data quality

```
count = df_bronze_converted.filter(col("pickup_datetime") > col("dropoff_datetime")).count()
print(f"The number of rows with pickup_datetime > dropoff_datetime is: {count}")

▸ (2) Spark Jobs

The number of rows with pickup_datetime > dropoff_datetime is: 0
```