Unleashing the power of Azure to enable critical business decision-making for a Travel Advisory.

- using Data Factory, ADLS Gen2, and Databricks

Summary:

In this project, Implemented end-to-end analytics on trip transaction data using Azure services. Utilized Azure Data Factory, ADLS Gen2, and Databricks for data transformation and pipeline resiliency. Explored Delta Lake's features for ACID transactions, data versioning, and schema enforcement. Leveraged PySpark in Databricks notebooks for data transformations. Scheduled pipeline in Azure Data Factory and employed Logic Apps for email triggers, ensuring resiliency.

Problem statement

The project aims to address the limitations and challenges of traditional data lakes for big data processing and analytics. Specifically, the challenges include:

1. **Lack of ACID transactions:** Traditional data lakes often lack support for atomicity, consistency, isolation, and durability (ACID) transactions. This makes it challenging to write reliable data pipelines and maintain data consistency.

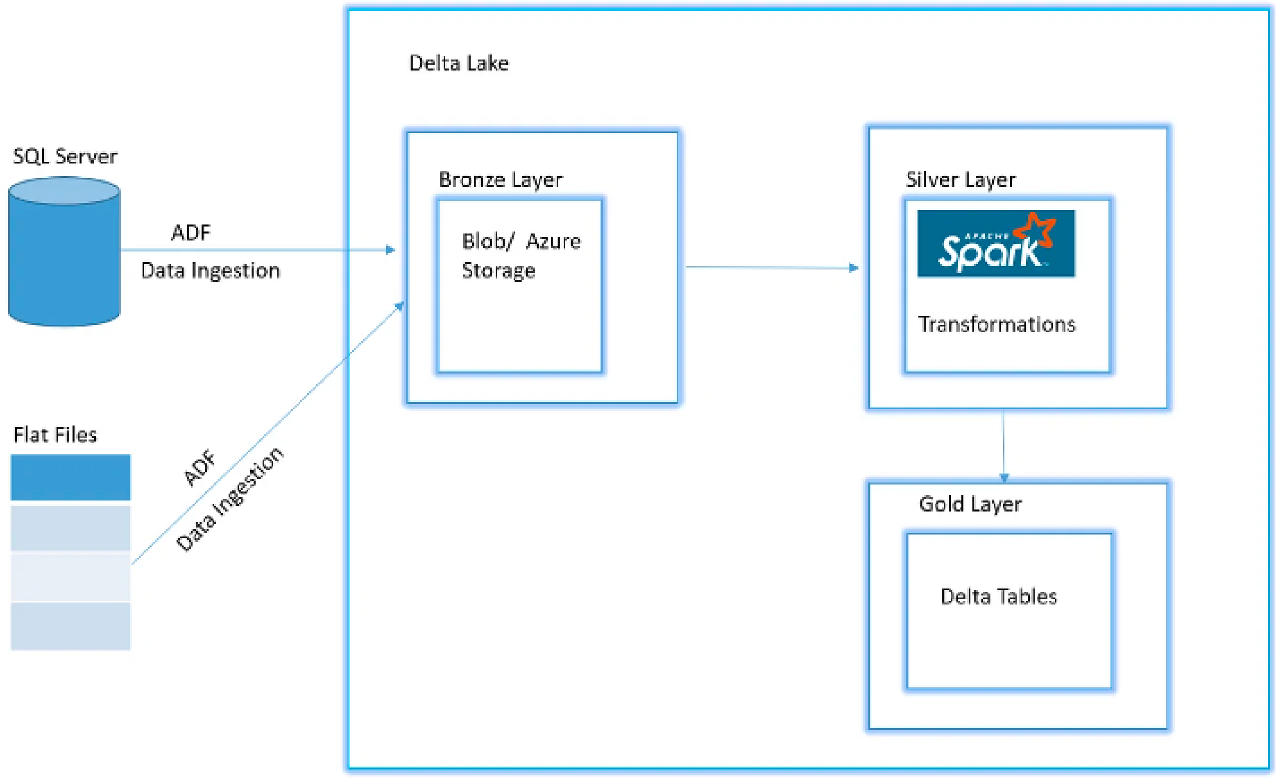
2. **Data versioning difficulties:** Tracking changes made to data over time and reverting to previous versions is a complex task in traditional data lakes. The absence of data versioning capabilities hinders efficient data engineering practices.

3. **Inconsistent data quality:** Without schema enforcement, data ingested into traditional data lakes may lack consistency and cleanliness. This can lead to issues during data processing and analysis.

4. **Limited time travel capabilities:** Traditional data lakes typically do not offer time travel queries, which are essential for examining data at specific points in time. This limitation hampers historical data analysis and decision-making.

5. **Suboptimal query performance:** Querying data from traditional data lakes can suffer from high latency and suboptimal performance due to a lack of advanced indexing and caching techniques.

Architecture Diagram



Tech Stack:

Language: **Python, SQL, Spark**
The project utilizes a combination of Python, SQL, and Spark for implementing various data processing and analytics tasks.

Package: **PySpark**
PySpark is the Python library used for interacting with Apache Spark. It enables data engineers and data scientists to leverage the power of Spark's distributed computing capabilities through Python code.

Services: The project leverages multiple Azure services to create an end-to-end data processing and analytics solution.

Azure Data Factory (ADF) enables the creation and orchestration of data workflows across various sources and destinations.

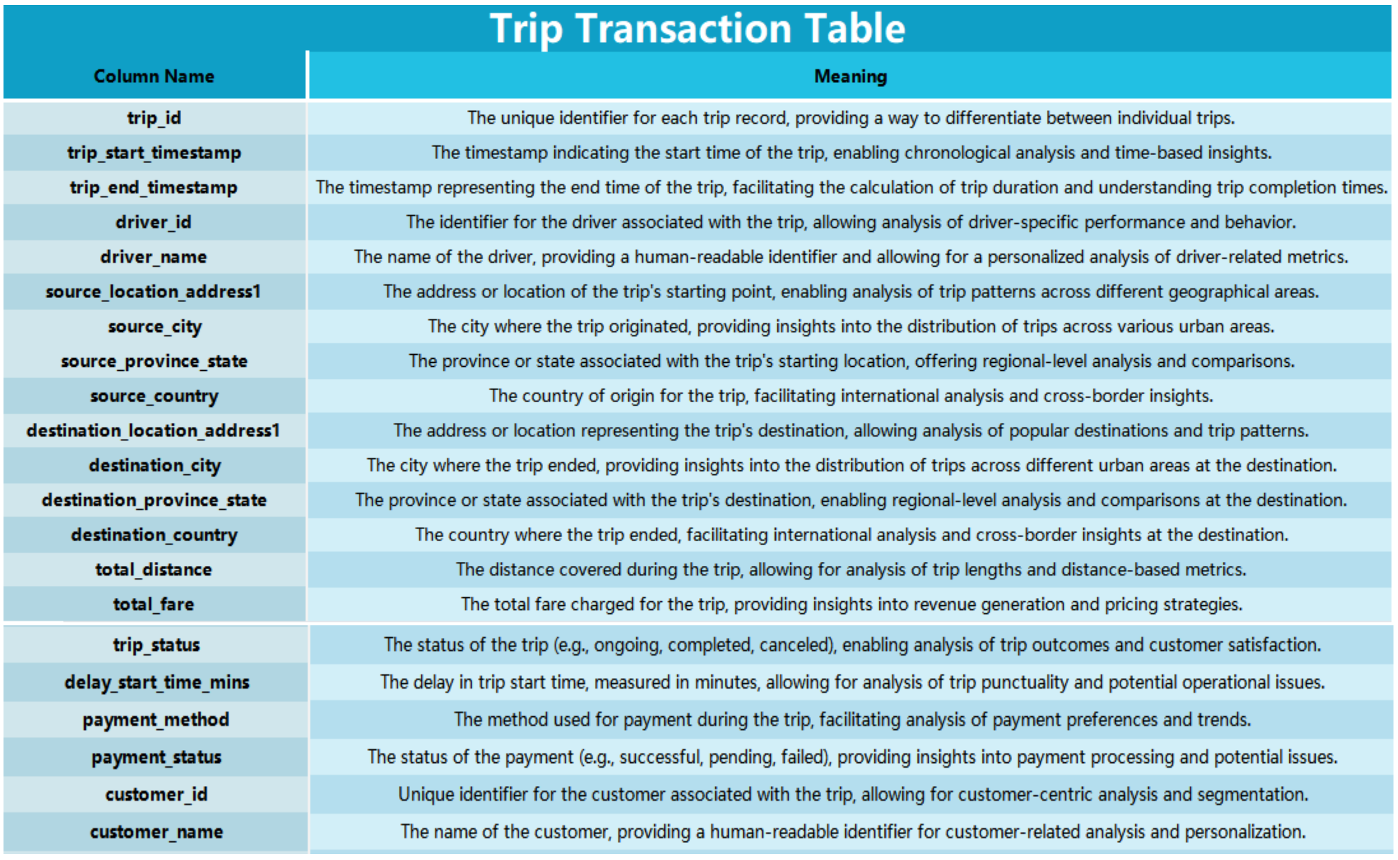
Azure Blob Storage (ADLS Gen2) provides scalable and cost-effective cloud storage for different types of data, including big data.

Azure Databricks offers a collaborative workspace for processing and analyzing large datasets using distributed computing capabilities.

Logic Apps automates business workflows and enhances the resilience of data pipelines.

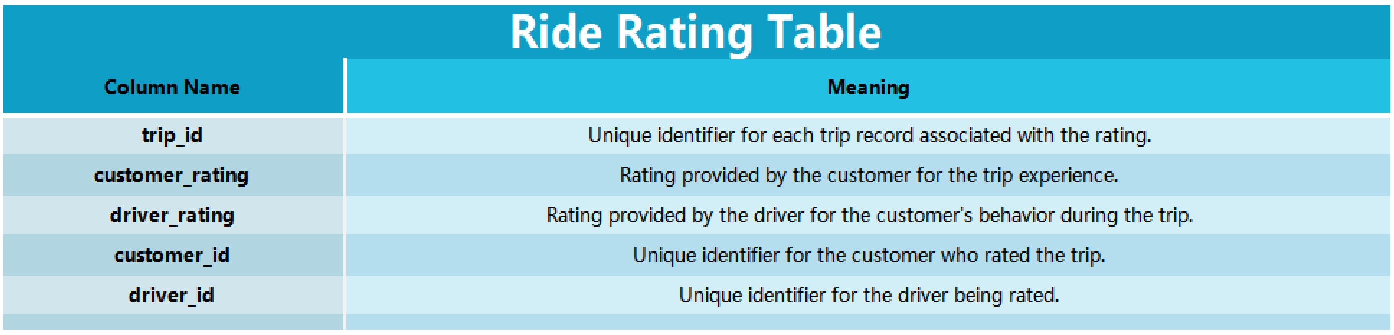
Azure SQL Database provides managed and scalable relational database solutions for efficient data management and analysis.

Understanding the Column Structure



Trip Transaction Table

Column Name	Meaning
trip_id	The unique identifier for each trip record, providing a way to differentiate between individual trips.
trip_start_timestamp	The timestamp indicating the start time of the trip, enabling chronological analysis and time-based insights.
trip_end_timestamp	The timestamp representing the end time of the trip, facilitating the calculation of trip duration and understanding trip completion times.
driver_id	The identifier for the driver associated with the trip, allowing analysis of driver-specific performance and behavior.
driver_name	The name of the driver, providing a human-readable identifier and allowing for a personalized analysis of driver-related metrics.
source_location_address1	The address or location of the trip's starting point, enabling analysis of trip patterns across different geographical areas.
source_city	The city where the trip originated, providing insights into the distribution of trips across various urban areas.
source_province_state	The province or state associated with the trip's starting location, offering regional-level analysis and comparisons.
source_country	The country of origin for the trip, facilitating international analysis and cross-border insights.
destination_location_address1	The address or location representing the trip's destination, allowing analysis of popular destinations and trip patterns.
destination_city	The city where the trip ended, providing insights into the distribution of trips across different urban areas at the destination.
destination_province_state	The province or state associated with the trip's destination, enabling regional-level analysis and comparisons at the destination.
destination_country	The country where the trip ended, facilitating international analysis and cross-border insights at the destination.
total_distance	The distance covered during the trip, allowing for analysis of trip lengths and distance-based metrics.
total_fare	The total fare charged for the trip, providing insights into revenue generation and pricing strategies.
trip_status	The status of the trip (e.g., ongoing, completed, canceled), enabling analysis of trip outcomes and customer satisfaction.
delay_start_time_mins	The delay in trip start time, measured in minutes, allowing for analysis of trip punctuality and potential operational issues.
payment_method	The method used for payment during the trip, facilitating analysis of payment preferences and trends.
payment_status	The status of the payment (e.g., successful, pending, failed), providing insights into payment processing and potential issues.
customer_id	Unique identifier for the customer associated with the trip, allowing for customer-centric analysis and segmentation.
customer_name	The name of the customer, providing a human-readable identifier for customer-related analysis and personalization.



Ride Rating Table

Column Name	Meaning
trip_id	Unique identifier for each trip record associated with the rating.
customer_rating	Rating provided by the customer for the trip experience.
driver_rating	Rating provided by the driver for the customer's behavior during the trip.
customer_id	Unique identifier for the customer who rated the trip.
driver_id	Unique identifier for the driver being rated.

Key features of Delta Lake:

ACID transactions on Spark:

Delta Lake utilizes a transaction log to track all commits made to the table directory, ensuring Atomicity, Consistency, Isolation, and Durability (ACID) transactions. It provides Serializable isolation levels to maintain data consistency across multiple users.

Unified Batch and Stream Processing:

Delta Lake enables the integration of both batch and streaming data processing. It offers a unified view of data from streaming sources (e.g., Kafka) and historical data (e.g., HDFS), allowing seamless operations for streaming data ingestion, batch historic backfill, and interactive queries.

Time Travel:

Delta Lake provides data versioning and snapshot capabilities, allowing users to query data as if a specific snapshot represents the current state of the system. This feature enables easy access to older versions of the data lake, facilitating activities such as audits and rollbacks.

Evolution of Delta Lake from Data Lake:

Delta Lake represents a significant evolution from traditional Data Lake architectures, addressing limitations and introducing advanced features.

Effective Transactional Control:

Delta Lake provides robust transactional control, ensuring reliable and consistent data operations while maintaining ACID compliance.

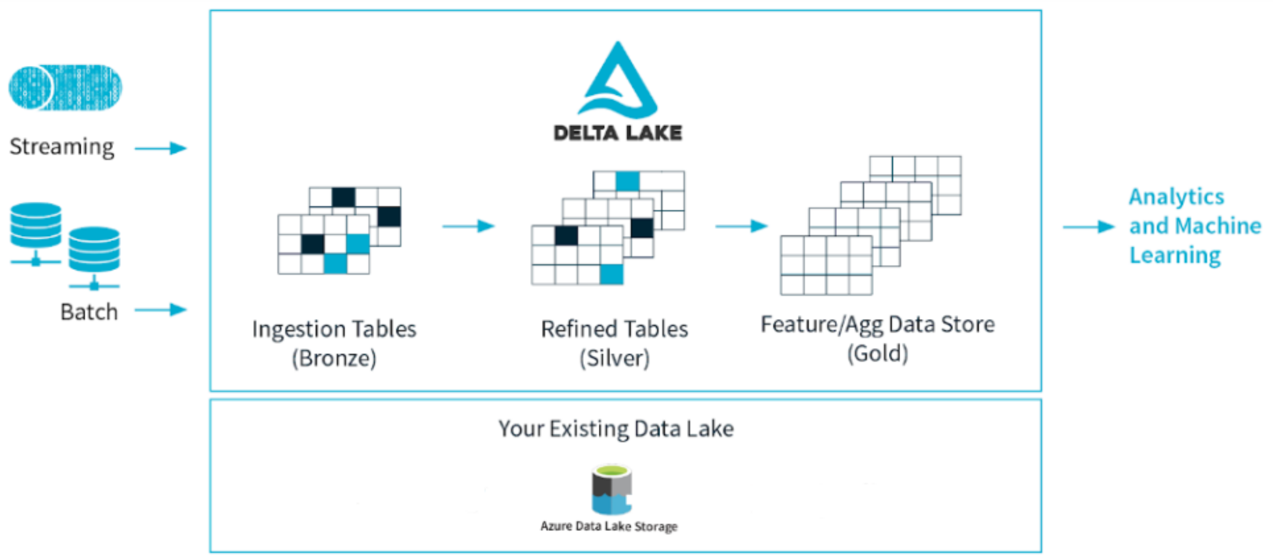
Schema Evolution:

Delta Lake supports seamless schema evolution, enabling easy modification and evolution of table schemas without disrupting existing data.

Delta Format Files vs Parquet:

Delta Lake's Delta format offers advantages over Parquet files, including optimized data storage, improved query performance, and enhanced data management capabilities.

Medallion Architecture:



Medallion Architecture: Empowering Data Excellence with Layers of Trust

Bronze Layer:

The bronze layer in the Medallion Architecture stores un-validated data. It maintains the raw state of the data source, allowing for incremental appends over time. The data in this layer can come from a combination of streaming and batch transactions.

Silver Layer:

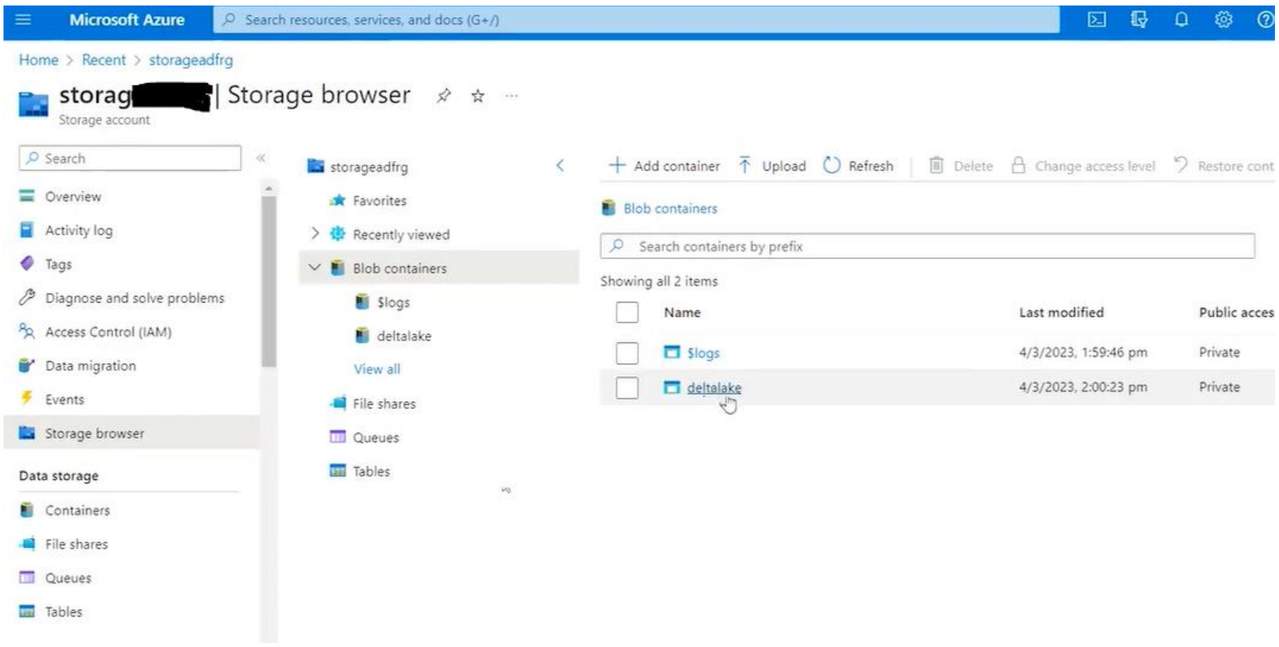
The silver layer represents validated and enriched data that can be trusted for downstream analytics. It contains a refined version of the data compared to the bronze layer and serves as a reliable source for analytics purposes.

Gold Layer:

The gold layer consists of highly refined and aggregated data that powers analytics, machine learning, and production applications. The data in this layer has been transformed into valuable knowledge rather than mere information. Gold tables play a crucial role in delivering insights and driving decision-making processes within the organization.

Steps Involved:

1st we have started Storage account with access tier as HOT



storag[redacted] | Storage browser

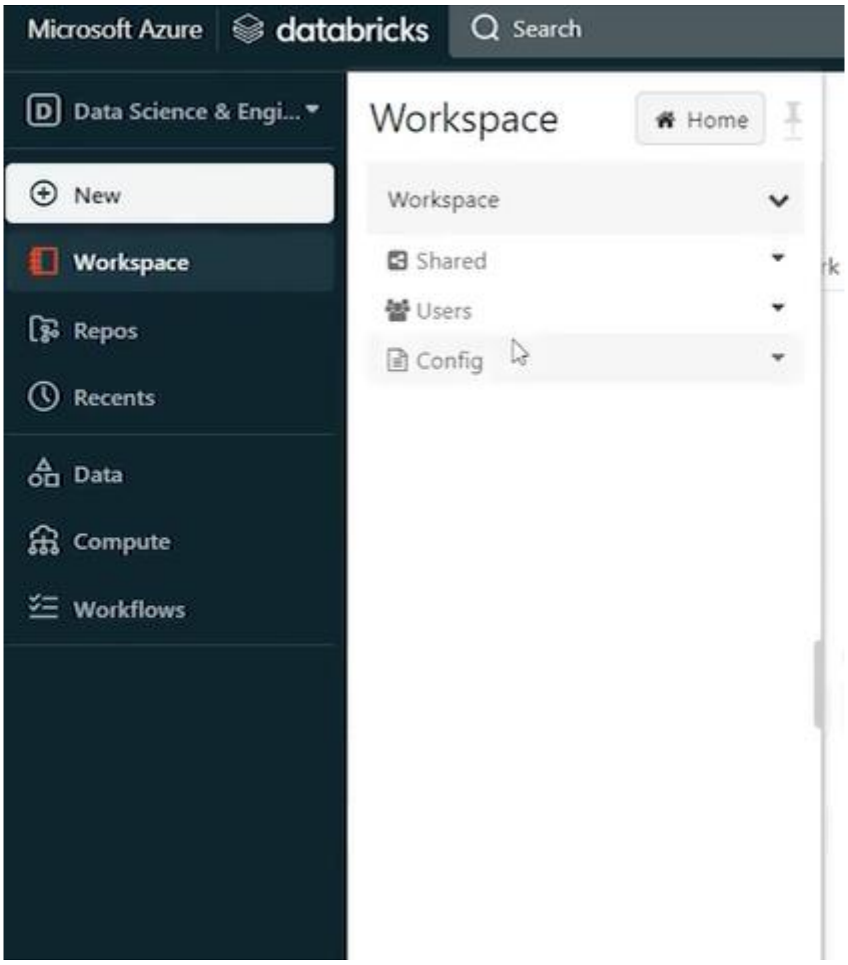
Showing all 2 items

Name	Last modified	Public acces
$logs	4/3/2023, 1:59:46 pm	Private
deltalake	4/3/2023, 2:00:23 pm	Private

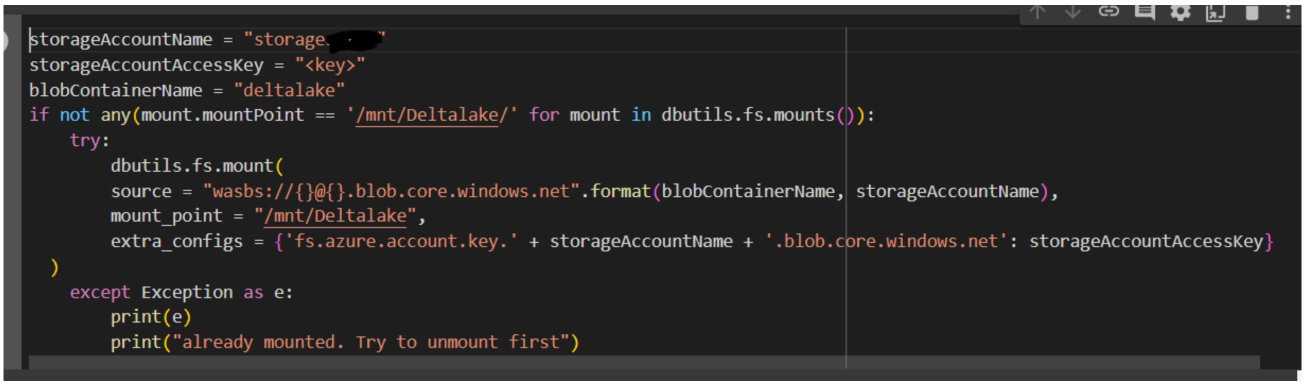
Inside delta lake we have to create Bronze, silver and Gold layer

With min of 8 max of 16 worker nodes.

Worker node memory 20GB with 10 cores each

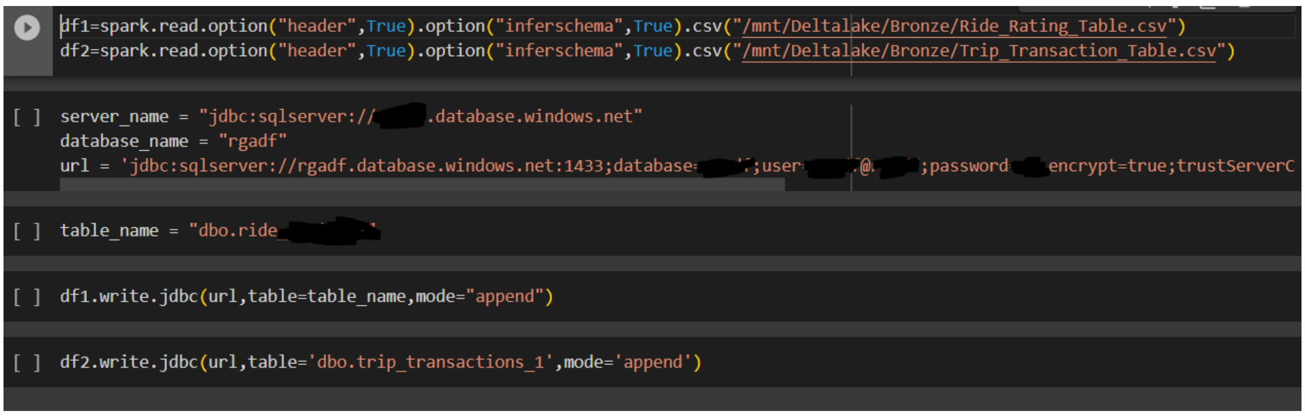


Using below config. file able to mount the storage account with Databricks



```
storageAccountName = "storage[redacted]"
storageAccountAccessKey = "<key>"
blobContainerName = "deltalake"
if not any(mount.mountPoint == '/mnt/Deltalake/' for mount in dbutils.fs.mounts()):
    try:
        dbutils.fs.mount(
        source = "wasbs://{}@{}.blob.core.windows.net".format(blobContainerName, storageAccountName),
        mount_point = "/mnt/Deltalake",
        extra_configs = {'fs.azure.account.key.' + storageAccountName + '.blob.core.windows.net': storageAccountAccessKey}
    )
    except Exception as e:
        print(e)
        print("already mounted. Try to unmount first")
```

Using below scipt we have loaded data from bronze data lake to Databricks.



```
df1=spark.read.option("header",True).option("inferschema",True).csv("/mnt/Deltalake/Bronze/Ride_Rating_Table.csv")
df2=spark.read.option("header",True).option("inferschema",True).csv("/mnt/Deltalake/Bronze/Trip_Transaction_Table.csv")
```

```
[ ]  server_name = "jdbc:sqlserver://[redacted].database.windows.net"
     database_name = "rgadf"
     url = 'jdbc:sqlserver://rgadf.database.windows.net:1433;database=[redacted]';user=[redacted]@[redacted];password[redacted]encrypt=true;trustServerC
```

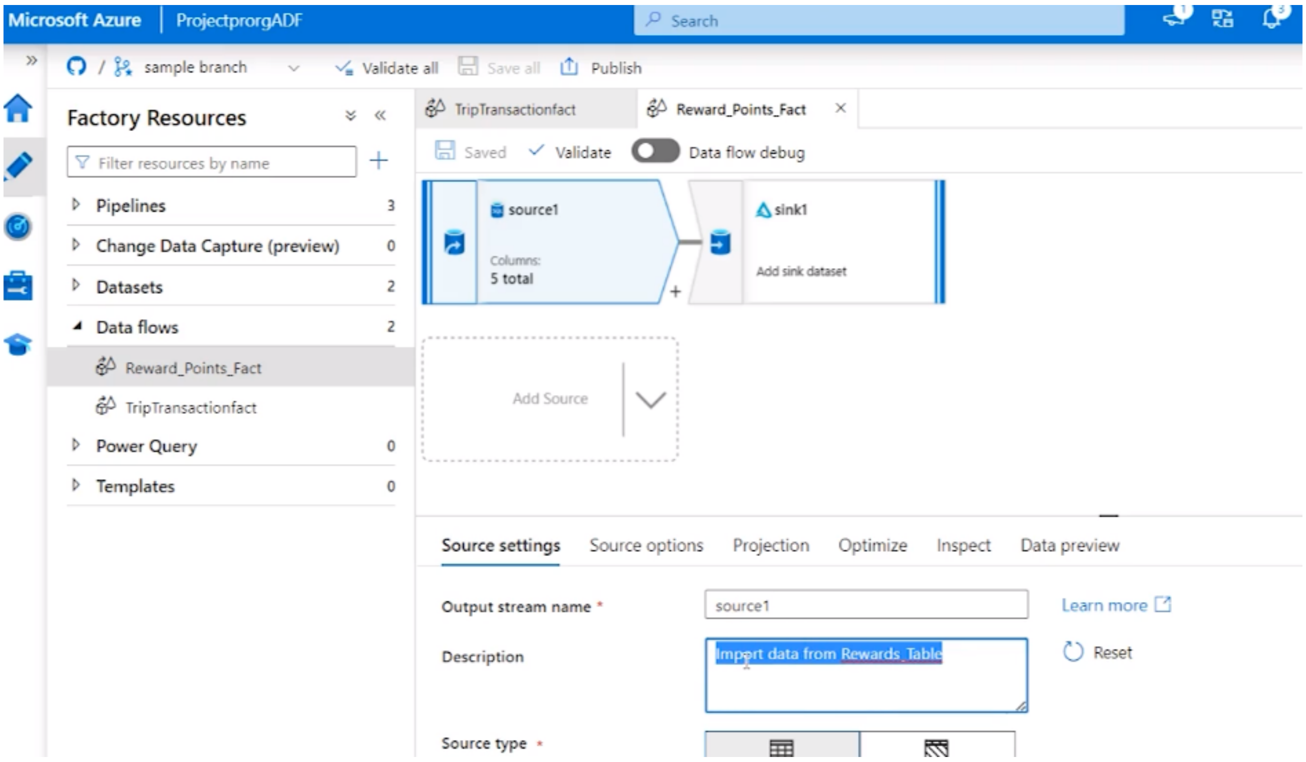
```
[ ]  table_name = "dbo.ride[redacted]"
```

```
[ ]  df1.write.jdbc(url,table=table_name,mode="append")
```

```
[ ]  df2.write.jdbc(url,table='dbo.trip_transactions_1',mode='append')
```

Now we will create DataFlow(Low code no code) in Azure Data factory to load data from SQL Database to bronze layer of delta lake

We also need to create link service which we need to provide in source properties



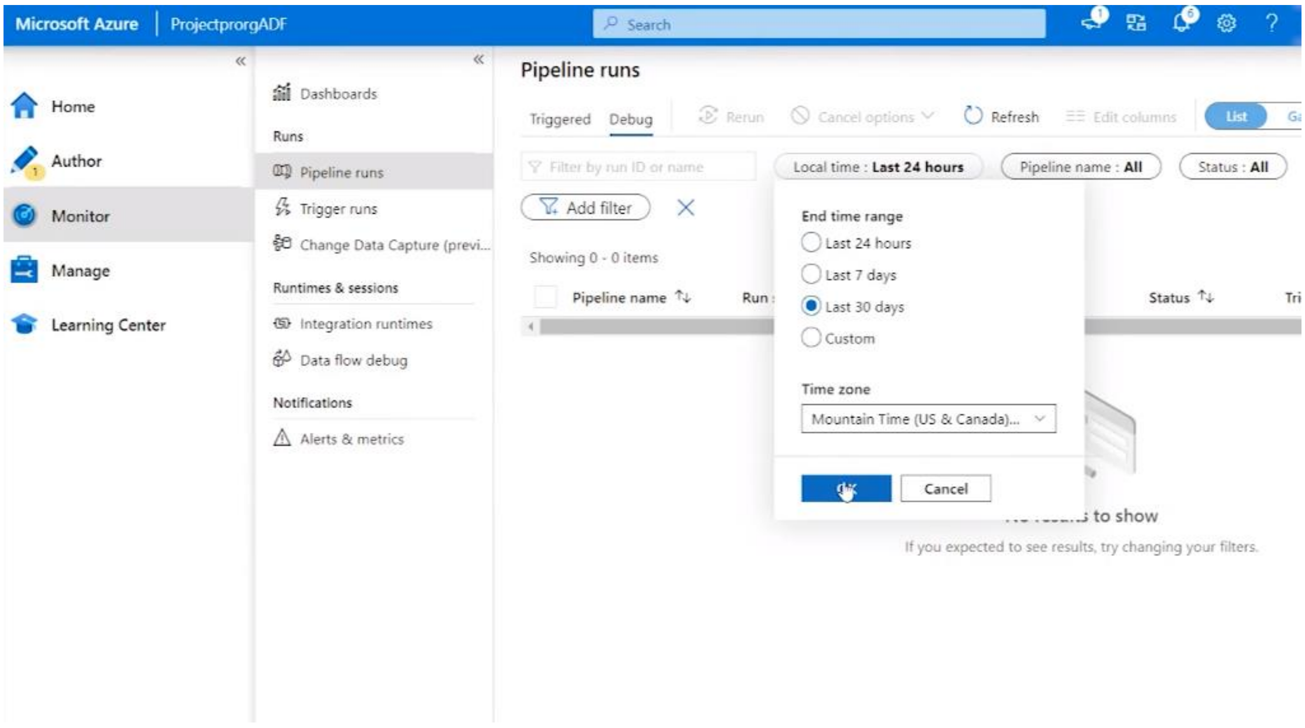
Output stream name *: source1

Description: Import data from Rewards_Table

In sink we will select blob storage and default format that we would like to load.

Once done we have to save and validate if all the components are fine, then we can publish the dataset, it will publish to git branc also we can trigger this pipeline once we are done with publish.

When we trigger our pipeline or debug we need to monitor our logs.



Microsoft Azure | ProjectprorgADF

Search

Home
Author
Monitor
Manage
Learning Center

Dashboards
Runs
Pipeline runs
Trigger runs
Change Data Capture (previ...
Runtimes & sessions
Integration runtimes
Data flow debug
Notifications
Alerts & metrics

Pipeline runs

Triggered | Debug | Rerun | Cancel options | Refresh | Edit columns | List

Filter by run ID or name | Local time : **Last 24 hours** | Pipeline name : **All** | Status : **All**

Add filter

Showing 0 - 0 items

Pipeline name | Run | Status | Tri

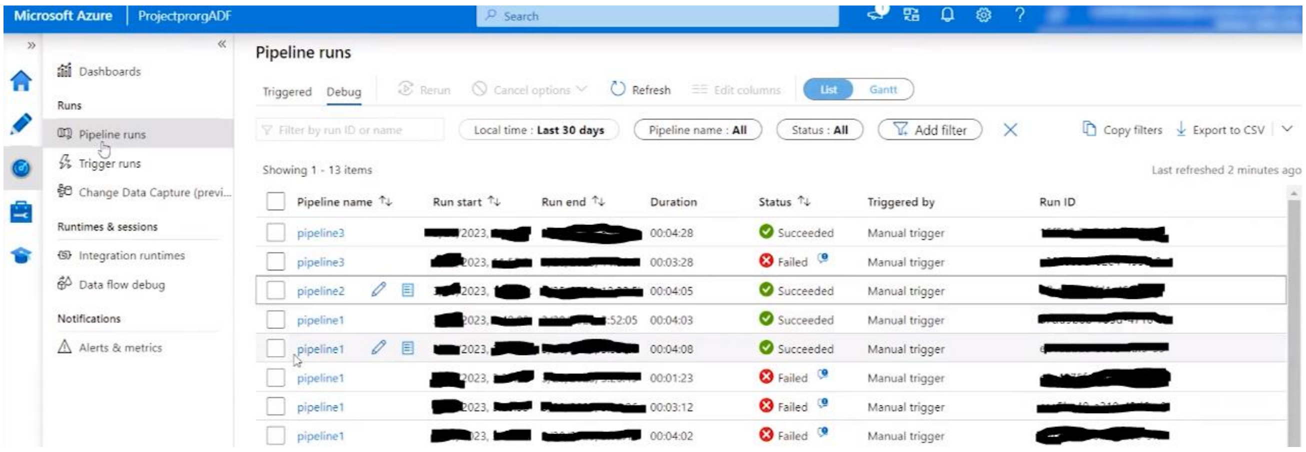
End time range
- Last 24 hours
- Last 7 days
- [✓ Last 30 days]
- Custom

Time zone
Mountain Time (US & Canada)...

OK | Cancel

... to show

If you expected to see results, try changing your filters.



Microsoft Azure | ProjectprorgADF

Search

Dashboards
Runs
Pipeline runs
Trigger runs
Change Data Capture (previ...
Runtimes & sessions
Integration runtimes
Data flow debug
Notifications
Alerts & metrics

Pipeline runs

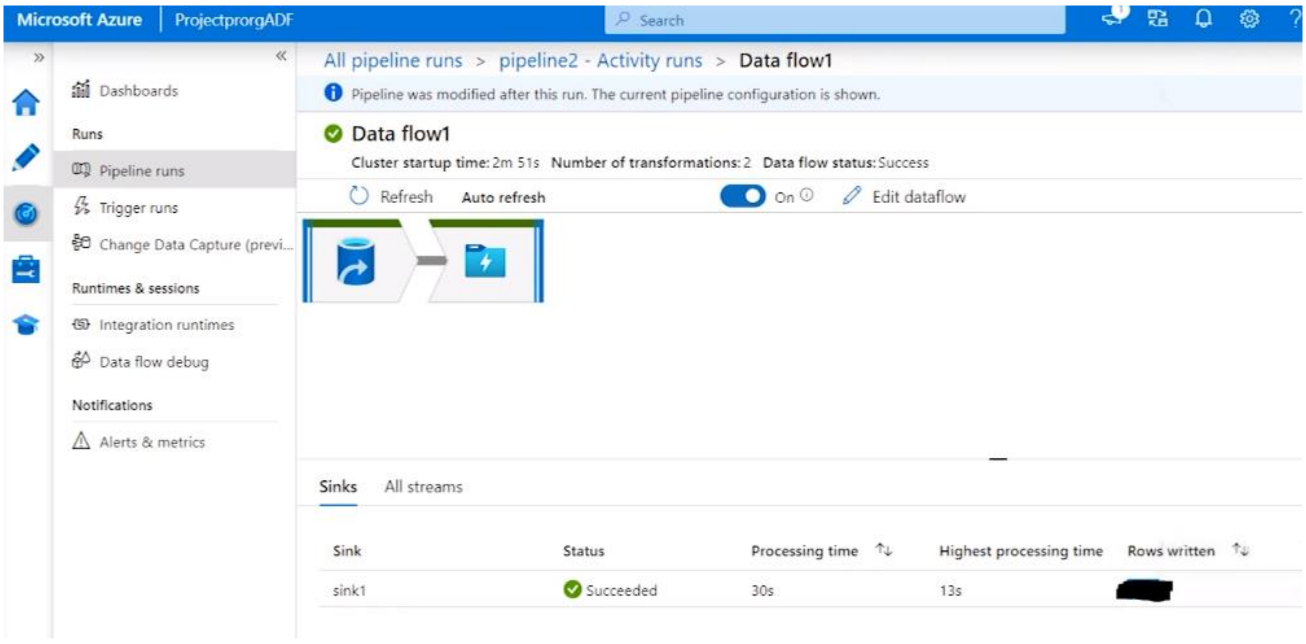
Triggered | Debug | Rerun | Cancel options | Refresh | Edit columns | List | Gantt

Filter by run ID or name | Local time : **Last 30 days** | Pipeline name : **All** | Status : **All** | Add filter | Copy filters | Export to CSV

Showing 1 - 13 items | Last refreshed 2 minutes ago

Pipeline name	Run start	Run end	Duration	Status	Triggered by	Run ID
pipeline3	[redacted]2023, [redacted]	[redacted]	00:04:28	✓ Succeeded	Manual trigger	[redacted]
pipeline3	[redacted]2023, [redacted]	[redacted]	00:03:28	Failed	Manual trigger	[redacted]
pipeline2	[redacted]2023, [redacted]	[redacted]	00:04:05	✓ Succeeded	Manual trigger	[redacted]
pipeline1	[redacted]2023, [redacted]	[redacted]:52:05	00:04:03	✓ Succeeded	Manual trigger	[redacted]
pipeline1	[redacted]2023, [redacted]	[redacted]	00:04:08	✓ Succeeded	Manual trigger	[redacted]
pipeline1	[redacted]2023, [redacted]	[redacted]	00:01:23	Failed	Manual trigger	[redacted]
pipeline1	[redacted]2023, [redacted]	[redacted]	00:03:12	Failed	Manual trigger	[redacted]
pipeline1	[redacted]23, [redacted]	[redacted]	00:04:02	Failed	Manual trigger	[redacted]

Below shows data received at sink



Microsoft Azure | ProjectprorgADF

Search

Dashboards
Runs
Pipeline runs
Trigger runs
Change Data Capture (previ...
Runtimes & sessions
Integration runtimes
Data flow debug
Notifications
Alerts & metrics

All pipeline runs > pipeline2 - Activity runs > Data flow1

Pipeline was modified after this run. The current pipeline configuration is shown.

Data flow1

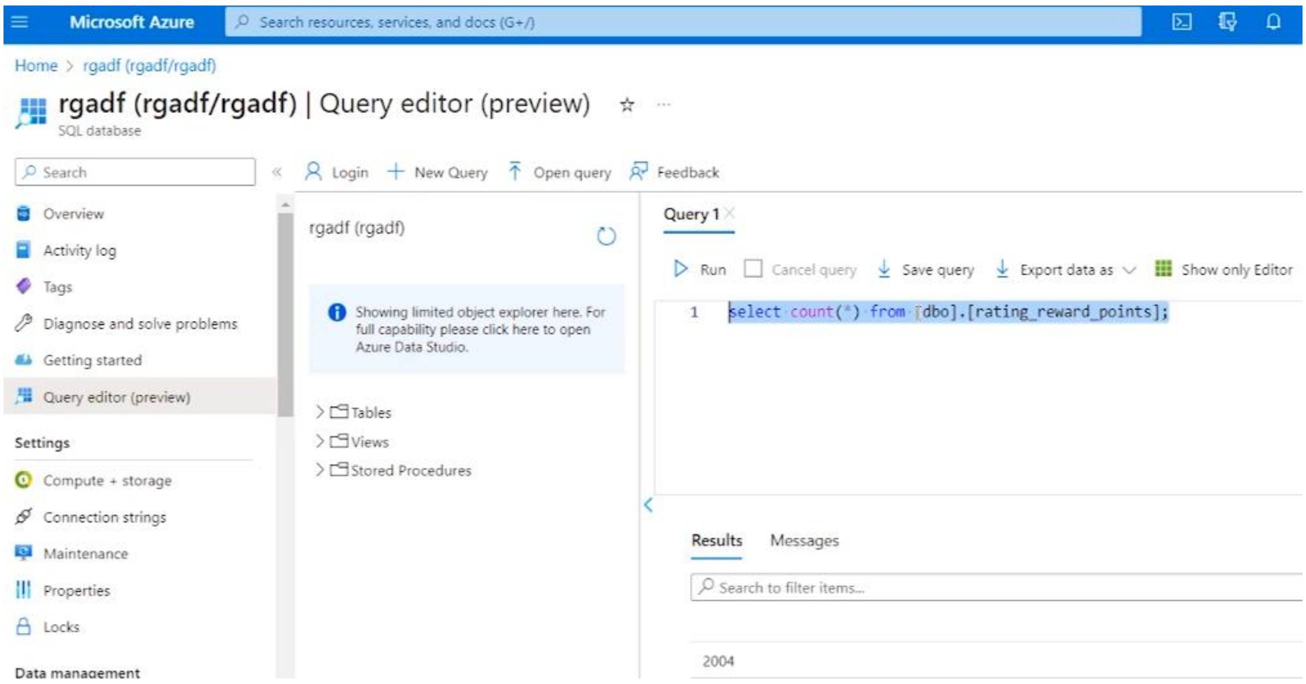
Cluster startup time: 2m 51s Number of transformations: 2 Data flow status: Success

Refresh | Auto refresh On | Edit dataflow

Sinks | All streams

Sink	Status	Processing time	Highest processing time	Rows written
sink1	✓ Succeeded	30s	13s	[redacted]

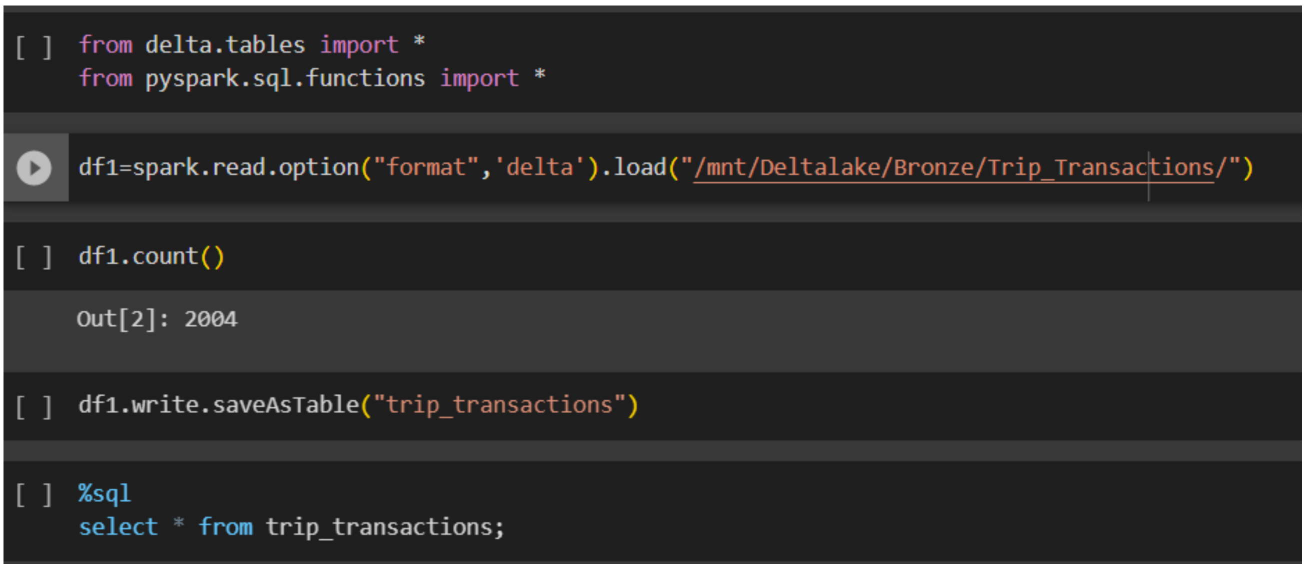
Can validate via query editor.



```sql
select count(*) from [dbo].[rating_reward_points];
```

```
2004
```

Now we will do processing of data via databricks.



```python
from delta.tables import *
from pyspark.sql.functions import *
```

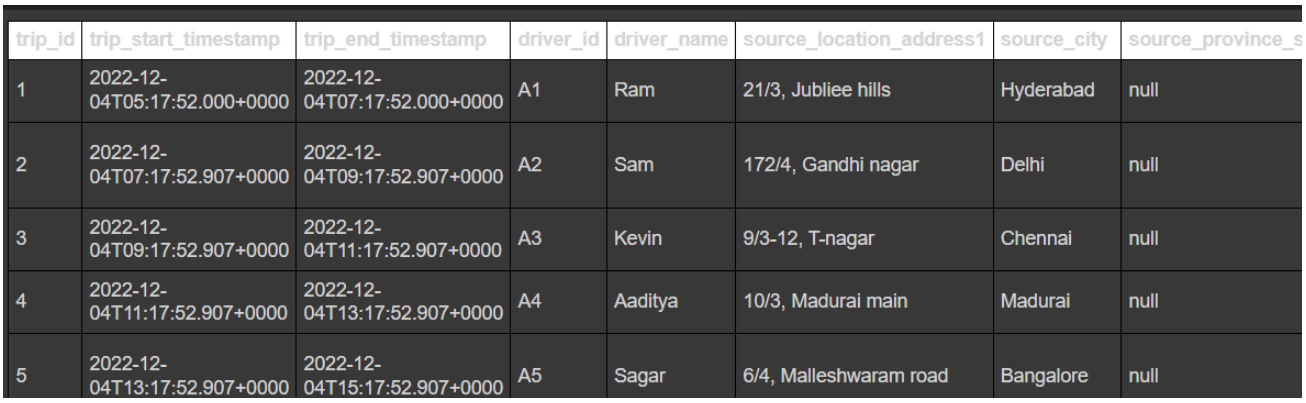
```python
df1=spark.read.option("format",'delta').load("/mnt/Deltalake/Bronze/Trip_Transactions/")
```

```python
df1.count()
```

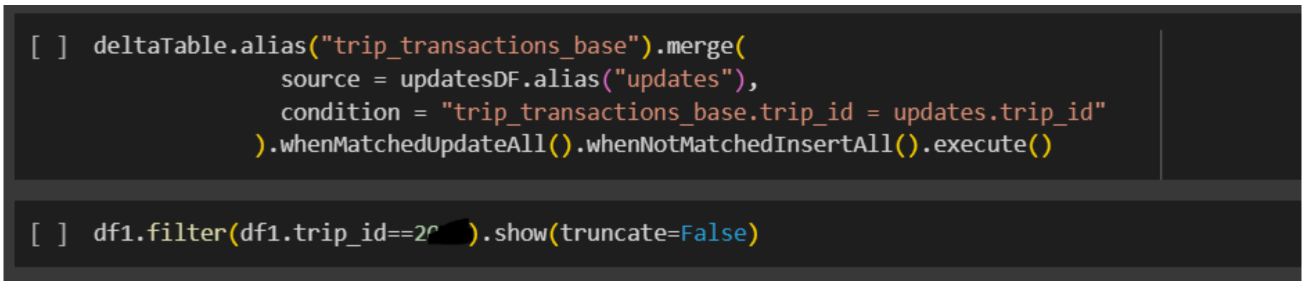
```
Out[2]: 2004
```

```python
df1.write.saveAsTable("trip_transactions")
```

```sql
%sql
select * from trip_transactions;
```



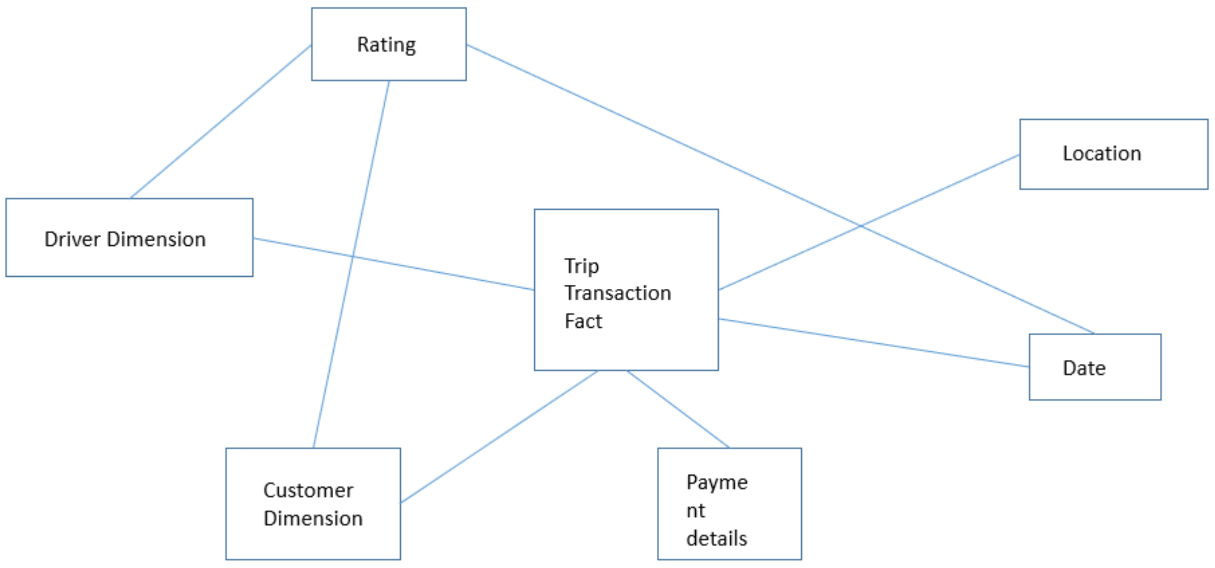
trip_id	trip_start_timestamp	trip_end_timestamp	driver_id	driver_name	source_location_address1	source_city	source_province_s
1	2022-12-04T05:17:52.000+0000	2022-12-04T07:17:52.000+0000	A1	Ram	21/3, Jubliee hills	Hyderabad	null
2	2022-12-04T07:17:52.907+0000	2022-12-04T09:17:52.907+0000	A2	Sam	172/4, Gandhi nagar	Delhi	null
3	2022-12-04T09:17:52.907+0000	2022-12-04T11:17:52.907+0000	A3	Kevin	9/3-12, T-nagar	Chennai	null
4	2022-12-04T11:17:52.907+0000	2022-12-04T13:17:52.907+0000	A4	Aaditya	10/3, Madurai main	Madurai	null
5	2022-12-04T13:17:52.907+0000	2022-12-04T15:17:52.907+0000	A5	Sagar	6/4, Malleshwaram road	Bangalore	null

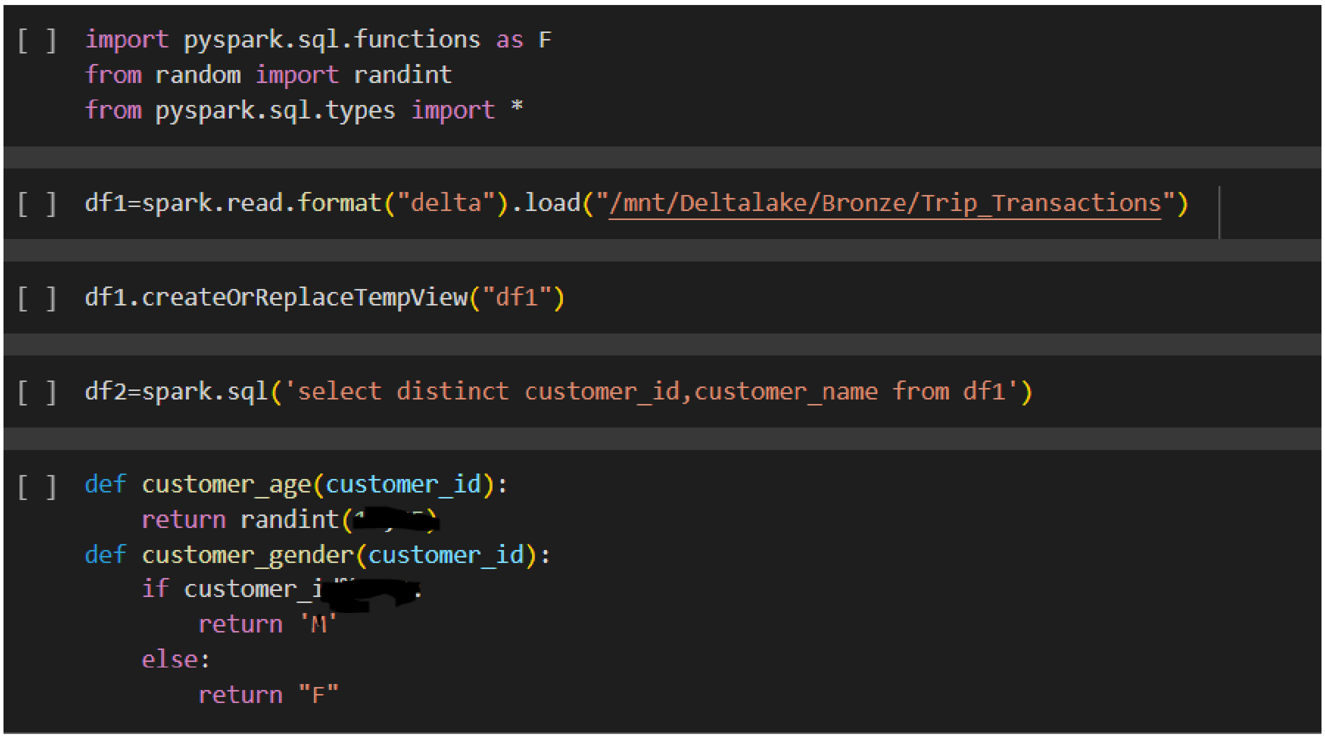


```
[ ]  deltaTable.alias("trip_transactions_base").merge(
                  source = updatesDF.alias("updates"),
                  condition = "trip_transactions_base.trip_id = updates.trip_id"
                ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()

[ ]  df1.filter(df1.trip_id==2[redacted]).show(truncate=False)
```

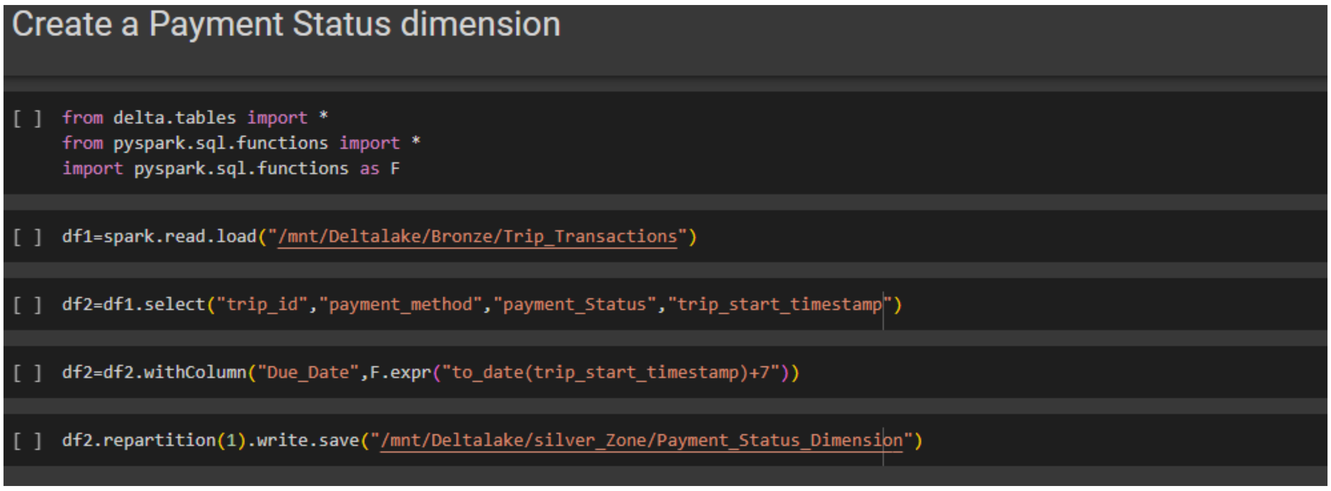
No we will work on silver layer, where we will transform our data to make facts dimension data modelling.

Fact- Dimension Data Modelling





```
[ ]  import pyspark.sql.functions as F
     from random import randint
     from pyspark.sql.types import *

[ ]  df1=spark.read.format("delta").load("/mnt/Deltalake/Bronze/Trip_Transactions")

[ ]  df1.createOrReplaceTempView("df1")

[ ]  df2=spark.sql('select distinct customer_id,customer_name from df1')

[ ]  def customer_age(customer_id):
         return randint([redacted])
     def customer_gender(customer_id):
         if customer_i[redacted]
             return 'M'
         else:
             return "F"
```

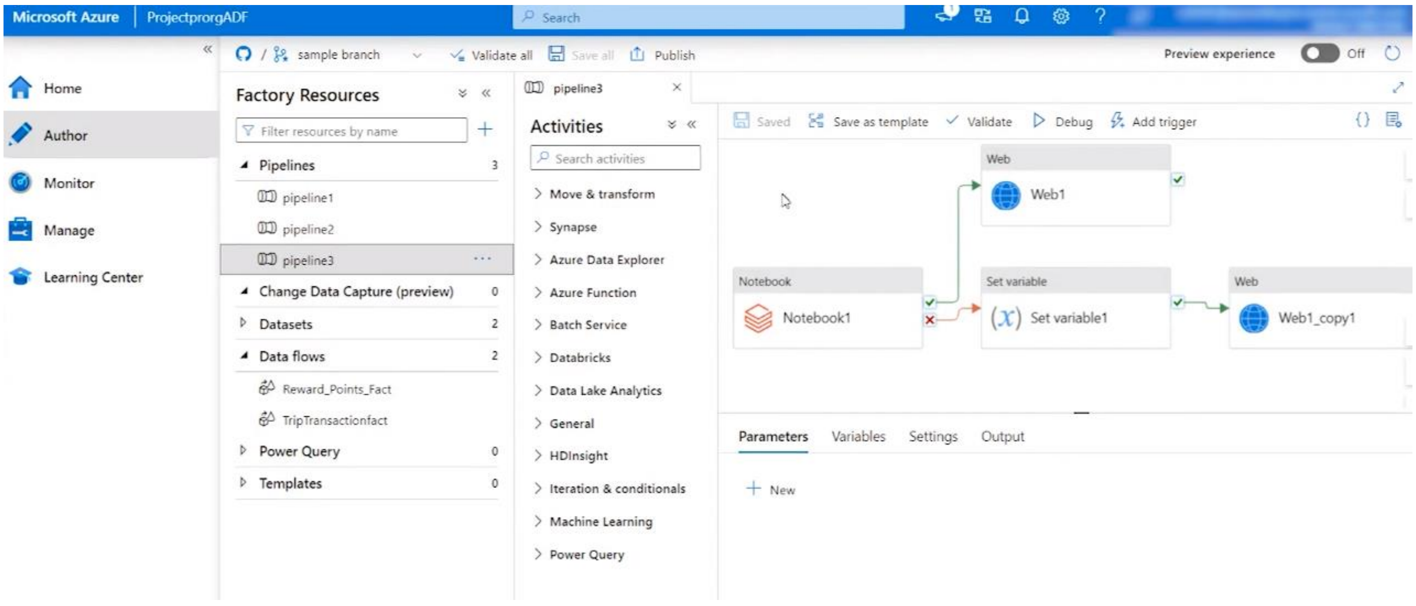


Create a Payment Status dimension

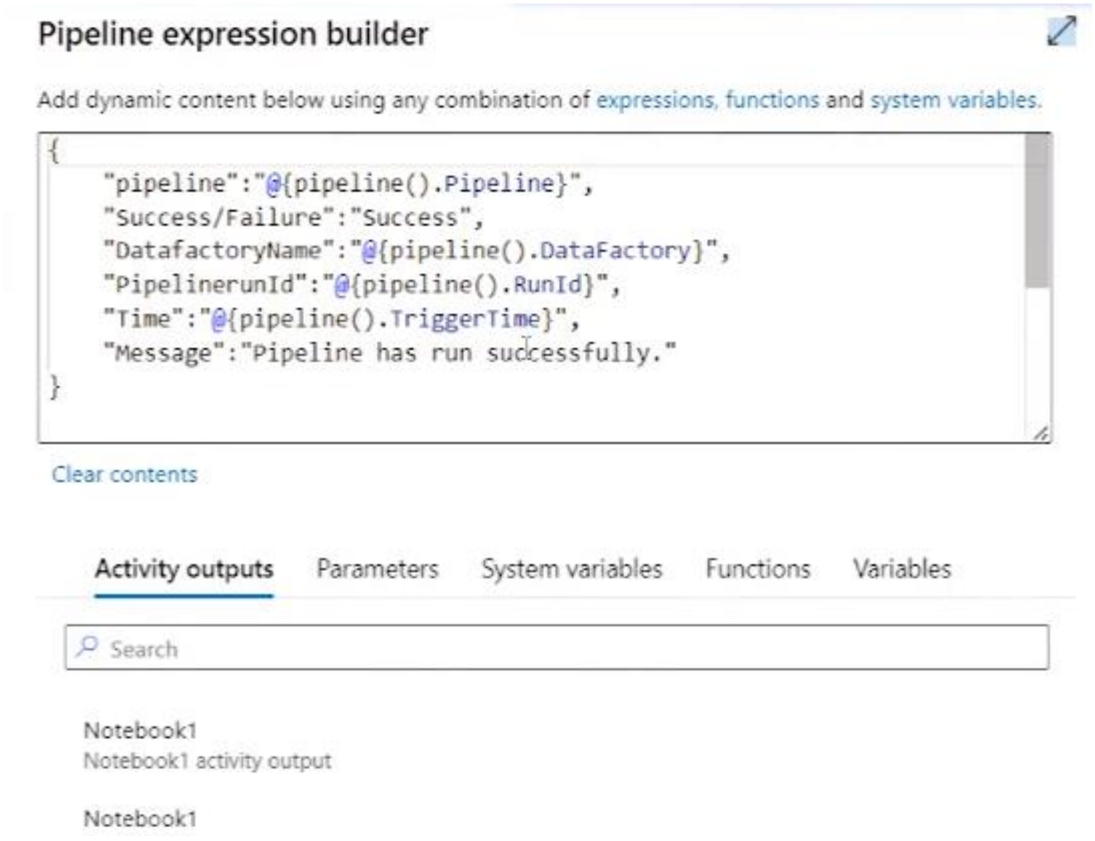
```
[ ]  from delta.tables import *
     from pyspark.sql.functions import *
     import pyspark.sql.functions as F

[ ]  df1=spark.read.load("/mnt/Deltalake/Bronze/Trip_Transactions")

[ ]  df2=df1.select("trip_id","payment_method","payment_Status","trip_start_timestamp")

[ ]  df2=df2.withColumn("Due_Date",F.expr("to_date(trip_start_timestamp)+7"))

[ ]  df2.repartition(1).write.save("/mnt/Deltalake/silver_Zone/Payment_Status_Dimension")
```

Like wise we will create all dimension table

Now for email or notification we need to create ADF pipeline

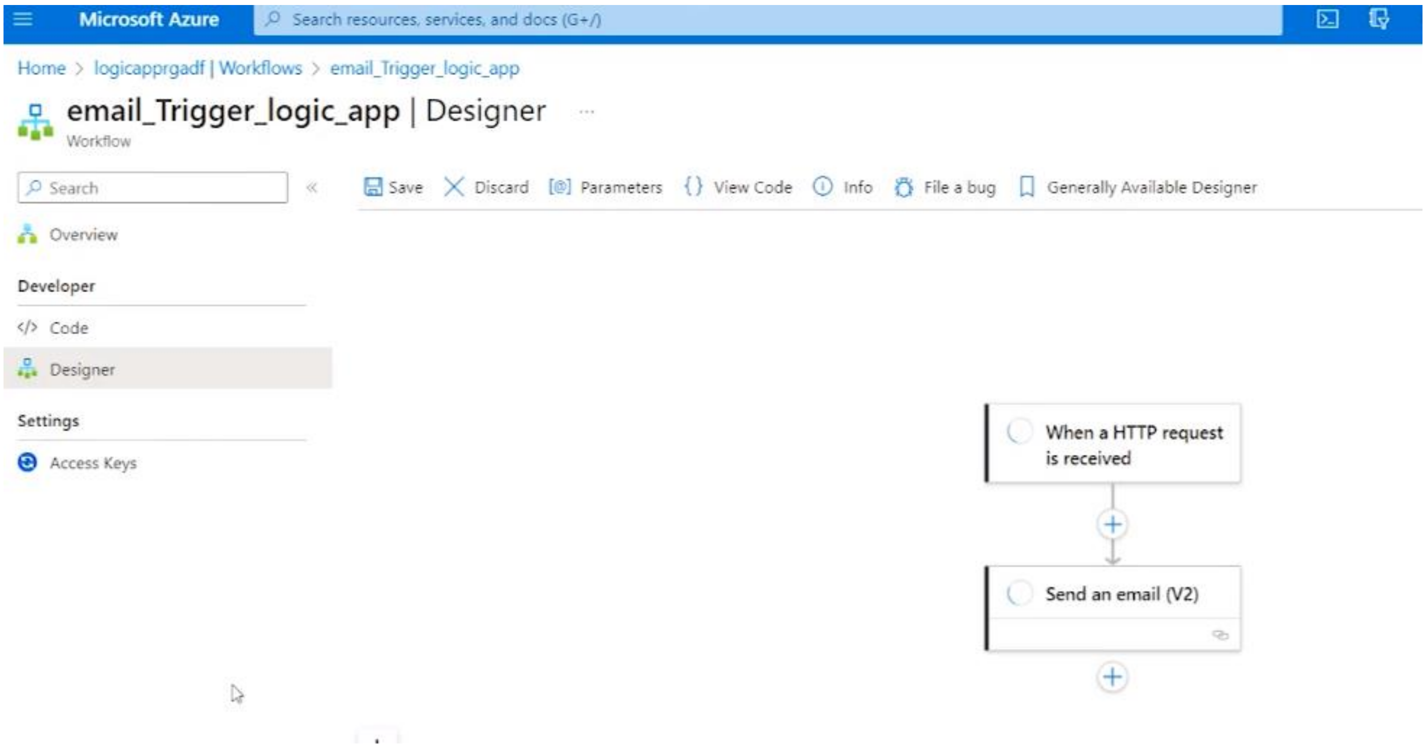


In web activity properties we need to provide logic app URL, once trigger it will post a API request with message in body, providing parameters


Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

```
{
    "pipeline":"@{pipeline().Pipeline}",
    "Success/Failure":"Success",
    "DatafactoryName":"@{pipeline().DataFactory}",
    "PipelinerunId":"@{pipeline().RunId}",
    "Time":"@{pipeline().TriggerTime}",
    "Message":"Pipeline has run successfully."
}
```

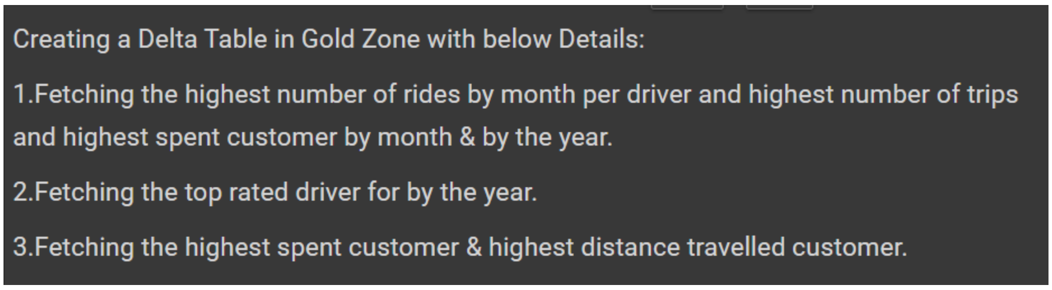

Microsoft Azure

Home > logicapprgadf | Workflows > email_Trigger_logic_app

email_Trigger_logic_app | Designer

Workflow

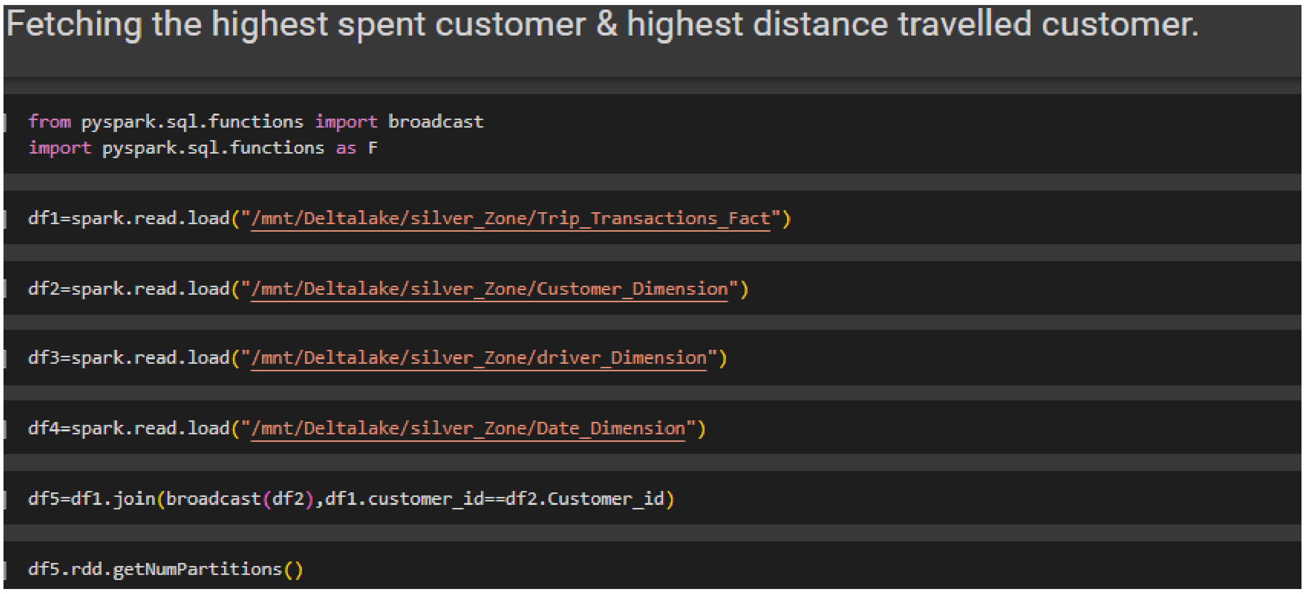
Data bricks code to write gold layer of storage account


Creating a Delta Table in Gold Zone with below Details:

1.Fetching the highest number of rides by month per driver and highest number of trips and highest spent customer by month & by the year.

2.Fetching the top rated driver for by the year.

3.Fetching the highest spent customer & highest distance travelled customer.


Fetching the highest spent customer & highest distance travelled customer.

```
from pyspark.sql.functions import broadcast
import pyspark.sql.functions as F
```

```
df1=spark.read.load("/mnt/Deltalake/silver_Zone/Trip_Transactions_Fact")
```

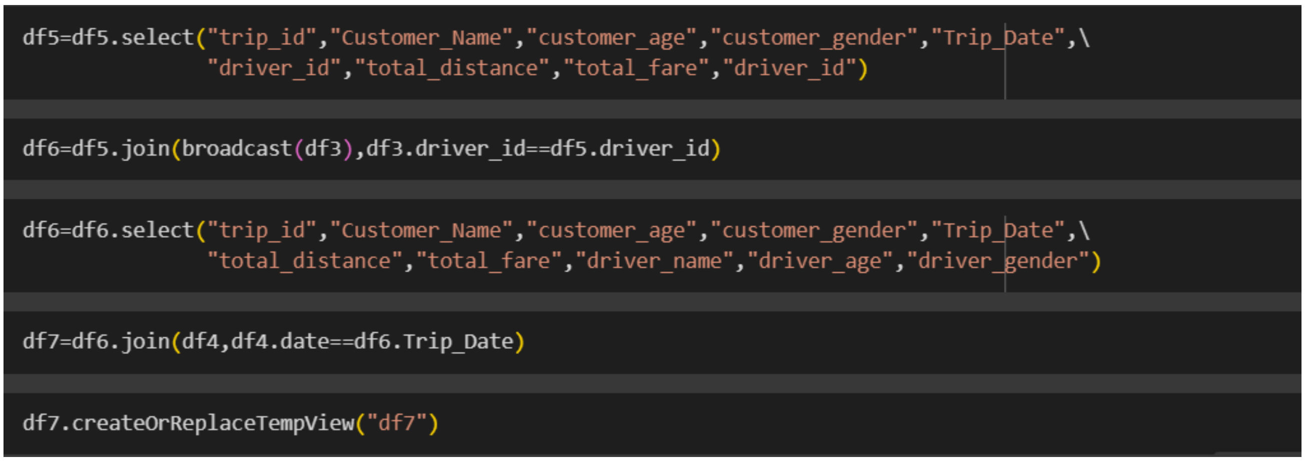
```
df2=spark.read.load("/mnt/Deltalake/silver_Zone/Customer_Dimension")
```

```
df3=spark.read.load("/mnt/Deltalake/silver_Zone/driver_Dimension")
```

```
df4=spark.read.load("/mnt/Deltalake/silver_Zone/Date_Dimension")
```

```
df5=df1.join(broadcast(df2),df1.customer_id==df2.Customer_id)
```

```
df5.rdd.getNumPartitions()
```



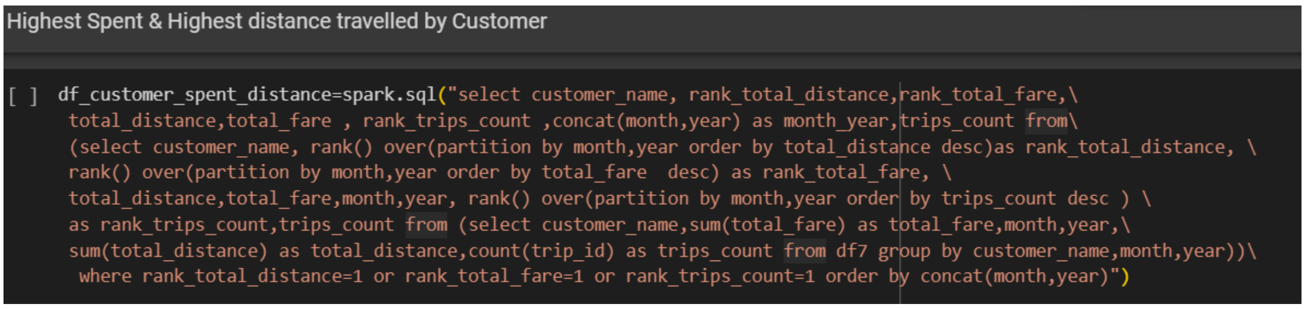
```
df5=df5.select("trip_id","Customer_Name","customer_age","customer_gender","Trip_Date",\
                 "driver_id","total_distance","total_fare","driver_id")
```

```
df6=df5.join(broadcast(df3),df3.driver_id==df5.driver_id)
```

```
df6=df6.select("trip_id","Customer_Name","customer_age","customer_gender","Trip_Date",\
                 "total_distance","total_fare","driver_name","driver_age","driver_gender")
```

```
df7=df6.join(df4,df4.date==df6.Trip_Date)
```

```
df7.createOrReplaceTempView("df7")
```

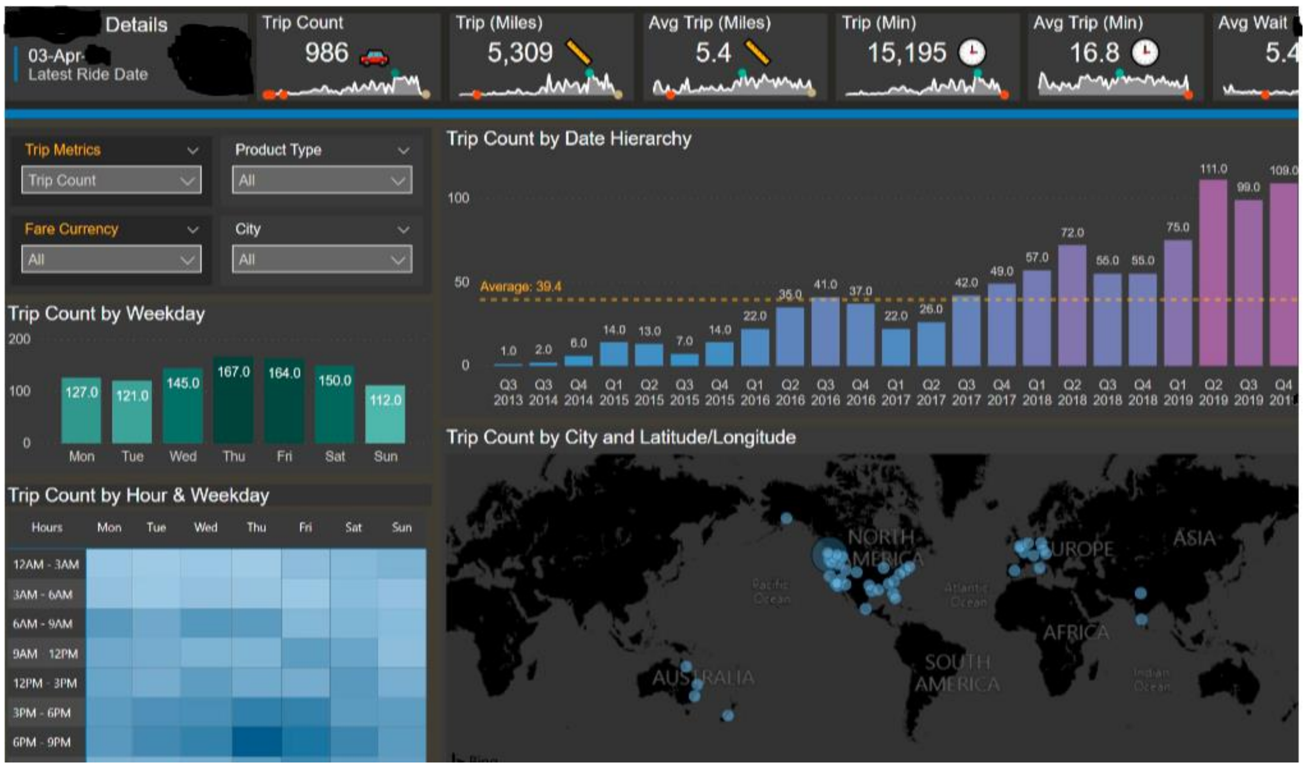


Highest Spent & Highest distance travelled by Customer

```
[ ]  df_customer_spent_distance=spark.sql("select customer_name, rank_total_distance,rank_total_fare,\
      total_distance,total_fare , rank_trips_count ,concat(month,year) as month_year,trips_count from\
      (select customer_name, rank() over(partition by month,year order by total_distance desc)as rank_total_distance, \
      rank() over(partition by month,year order by total_fare  desc) as rank_total_fare, \
      total_distance,total_fare,month,year, rank() over(partition by month,year order by trips_count desc ) \
      as rank_trips_count,trips_count from (select customer_name,sum(total_fare) as total_fare,month,year,\
      sum(total_distance) as total_distance,count(trip_id) as trips_count from df7 group by customer_name,month,year))\
       where rank_total_distance=1 or rank_total_fare=1 or rank_trips_count=1 order by concat(month,year)")
```

Writing to delta table which is hive meta store

We Can connect these Hive meta table to Power BI to get insight from these tables.



Challenge Faced

- Data quality: Ensuring the accuracy, completeness, and consistency of data from diverse sources.
- Scalability: Handling large volumes of data and accommodating future growth.
- Data integration: Integrating data from various systems and sources to create a unified view.
- Real-time processing: Performing timely data processing and analytics to enable real-time decision-making.

Business Benefit

- Improved data analysis and decision-making.
- Optimized data processing efficiency.
- Streamlined and organized data architecture.
- Enhanced data workflow efficiency.
- Advanced data transformation capabilities.
- Automated data processing and increased resilience.
- Proactive monitoring and quality assurance.

By: Abhishek Verma
Senior Associate- Cognizant
vermabhi.90@gmail.com