Capstone Project Submission Report

E-Commerce Data Engineering Pipeline with Azure and Databricks

Name: **Jatin Midha**
Employee ID: **127287**
Course/Program: **Data Engineering**
Instructor Name: **Piyush Raj Kataya**

Table of Contents

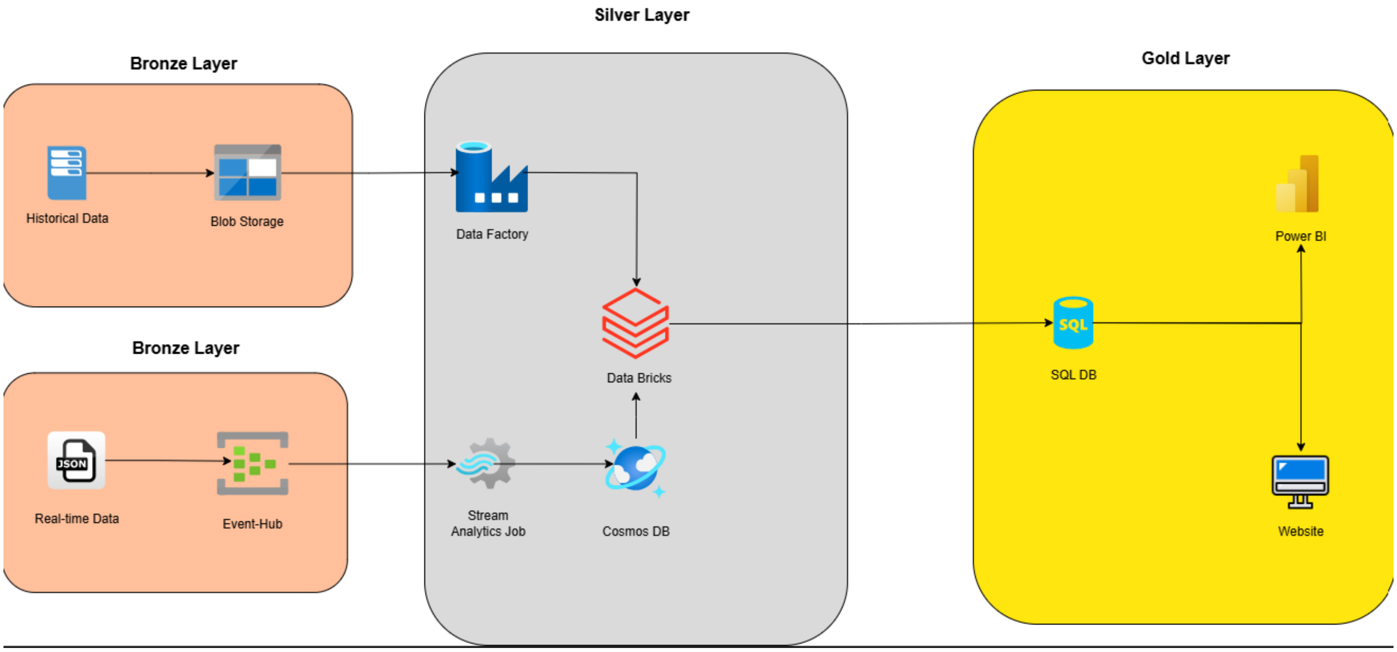
Problem Statement

The task involves developing a comprehensive data processing and analytics solution for a fictional e-commerce company. This company processes millions of transactions daily, necessitating the analysis of this data to extract valuable business insights. The solution must be capable of ingesting raw transactional data from various sources, including APIs, structured databases, and event-driven streams. Subsequently, the data must be transformed and loaded into a database for reporting and analysis. The solution must be highly available, scalable, secure, and optimized for cost efficiency.

Objective

Implement an ETL pipeline using Azure services and Databricks to process, analyze, and visualize large datasets, adhering to production-level requirements. We will design, deploy, and present an end-to-end solution that integrates various Azure services, focusing on real-world data engineering challenges.

Workflow Architecture



Bronze Layer

- **Data Sources**: Historical data is stored in Blob Storage, and real-time data is ingested through Event Hub.
- **Data Processing**: Stream Analytics Job is used to process real-time data as it arrives.
- **Purpose**: This layer captures raw data from various sources and performs initial processing on real-time data. It serves as the landing zone for all incoming data.

Silver Layer

- **Data Processing**: Data Factory is used to further process and transform the data.
- **Data Storage**: The processed data is stored in SQL DB and Cosmos DB.
- **Purpose**: This layer cleanses, validates, and enriches the data. It involves deduplication, transformation, and integration of data from different sources.

Gold Layer

- **Data Consumption**: The refined data is made available for business intelligence and analytics through Power BI and a website.

- **Purpose**: This layer contains highly curated, business-ready data optimized for reporting, dashboards, and advanced analytics.

Data Collection, Exploratory Data Analysis(EDA), and Data Preprocessing

Data Collection:

The provided datasets are:

Historical Data

1. **customer_data.csv**: Contains customer information such as names, contact details, and demographic data.

Table: Data Description

Attribute Name	Data Type	Description	Missing Values (%)
Customer_id	Integer	Id of Customers	0%
First_name	String	Customer’s first name	0%
Last_name	String	Customer’s last name	0%
Signup_date	Date	Signup date of cust.	0%
Address	String	Customers address	0%
Email	String	Customers email	0%

2. **inventory_data.csv**: Includes details about the products in stock, such as item names, quantities, and prices.

Table: Data Description

Attribute Name	Data Type	Description	Missing Values (%)
Product_id	Integer	Id of product	0%
Current_stock	Integer	Stock Available	0%
Reorder_level	Integer	Items used	0%

3. **Product_data.csv**: Similar to the previous file, contains data of product inventory with price, categories.

Table: Data Description

Attribute Name	Data Type	Description	Missing Values (%)
Product_id	Integer	Id of product	0%
Product_name	String	Name of Product	0%
Stock_Quantity	Integer	Items available	0%
Category	String	Categories of product	0%
Price	Integer	Price of Product	0%

4. **Reviews_data.csv**: Holds customer reviews and ratings for various products or services.

Table: Data Description

Attribute Name	Data Type	Description	Missing Values (%)
Review_id	String	Review ID	0%
Customer_id	String	Customer ID	0%
Product_id	String	Product ID	0%
Rating	Integer	Product Rating	0%
Review_Text	String	Customer Feedback	0%
Review_date	Date	Date of review	0%

5. **Transaction_data.csv**: Records transaction details, including purchase dates, amounts, and customer IDs.

Attribute Name	Data Type	Description	Missing Values (%)
Transaction_id	String	Review ID	0%
Customer_id	String	Customer ID	0%
Transaction_date	date	Transaction date	0%

Product_id	String	Product ID	0%
Quantity	Integer	Product Quantity	0%
Payment_Type	String	Mode of Payment	0%
Transaction_amount	Float	Transaction amount	0%

Real-Time Data

1. **Realtime transactions.json:** Contains real-time transaction data including transaction IDs, timestamps, amounts, and customer details.

Attribute Name	Data Type	Description	Missing Values (%)
Transaction_id	String	Review ID	0%
Customer_id	String	Customer ID	0%
Transaction_date	date	Transaction date	0%
Product_id	String	Product ID	0%
Quantity	Integer	Product Quantity	0%
Payment_Type	String	Mode of Payment	0%
Transaction_amount	Float	Transaction amount	0%

Data Preprocessing

The data preprocessing phase involved cleaning the raw transactional data using PySpark in Databricks. This process is crucial for ensuring the quality and reliability of the data before it is used for analysis and reporting. Here is a detailed explanation of the steps involved:

1. **Data Ingestion**:

   The raw data, stored in CSV files in Azure Blob Storage, was ingested into Databricks using PySpark. Databricks provides a collaborative environment for data engineering and data science, making it an ideal platform for preprocessing large datasets.

2. **Data Cleaning**:

   Removing Duplicates: Duplicate records were identified and removed to ensure that each transaction is unique. This was achieved using the dropDuplicates() function in PySpark.

   Handling Missing Values: Missing values were handled by either filling them with appropriate default values or by removing the records with missing critical information. The dropna() functions in PySpark was used for this purpose.

Table: Data Preprocessing Summary

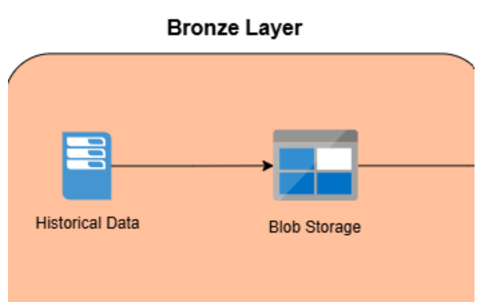
Step	Technique	Result
Checked Null Values	Counting each column null	Consistent Data
Checked Duplicates	Using filter and count together	Clean data without duplicates

5. Data Storage and Optimization

For Historical Data

Azure Blob Storage:

Purpose: Used for extracting and storing raw transactional data from CSV files.
Reason for Choice: Azure Blob Storage provides scalable and cost-effective storage for unstructured data. It supports a wide range of data formats and integrates seamlessly with other Azure services. Blob Storage offers high durability and availability, making it an ideal choice for storing large volumes of raw data.



For Real Time Data Storage

Azure Cosmos DB:

Purpose: Used for storing real-time data from Event Hub.
Reason for Choice: Azure Cosmos DB is a globally distributed, multi-model database service that provides low-latency and high-throughput performance. It is designed to handle real-time data streams and supports automatic scaling to accommodate varying workloads. Cosmos DB offers comprehensive security features, including encryption and fine-grained access control, ensuring the protection of real-time data.

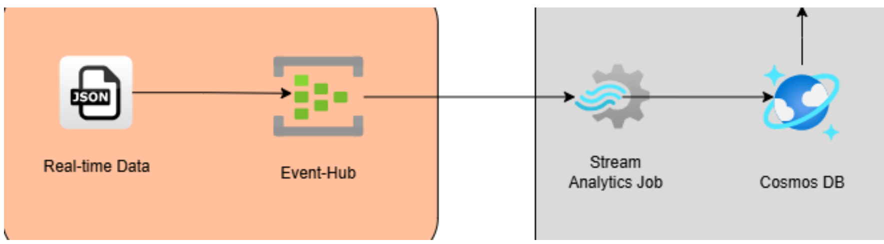


Fig. Real time data storage

Azure SQL Database:

Purpose: Used for storing transformed data for reporting and analysis. Reason for Choice: Azure SQL Database is a fully managed relational database service that offers high performance, scalability, and security. It supports automated backups, patching, and scaling, ensuring high availability and reliability. The service also provides advanced security features, including encryption and threat detection, making it suitable for storing sensitive business data.

Partitioning & Indexing Strategies:

Composite Indexing:

- **Technique**: Composite Indexing
- **Description**: Composite indexes were created on multiple columns that are often used together in queries, such as customer ID and transaction date. This technique helps in optimizing complex queries that involve multiple columns.
- **Result**: Improved performance for multi-column queries, as the composite index allows for efficient data retrieval based on multiple criteria.

6. Real-Time Data Processing and Streaming

Real-Time Data Processing:

Azure Event Hubs for Data Ingestion – Real-time transaction data is streamed into Azure Event Hubs, acting as a highly scalable event ingestion service.

Azure Stream Analytics for Data Processing – Incoming transaction data is processed using Azure Stream Analytics (ASA) to perform transformations, aggregations, and anomaly detection in real time.

Apache Spark on Azure Databricks for Advanced Processing – Real-time fraud detection, risk assessment, and customer behavior analysis are performed using Spark Structured Streaming in Azure Databricks.

Delta Lake for Streaming Data Storage – Streamed data is written into Delta Lake tables in ADLS (Bronze Layer), ensuring ACID compliance and real-time updates.

Power BI & Alerts for Instant Insights – Real-time dashboards in Power BI display live transaction trends, and Azure Functions trigger alerts for suspicious activities like potential fraud.

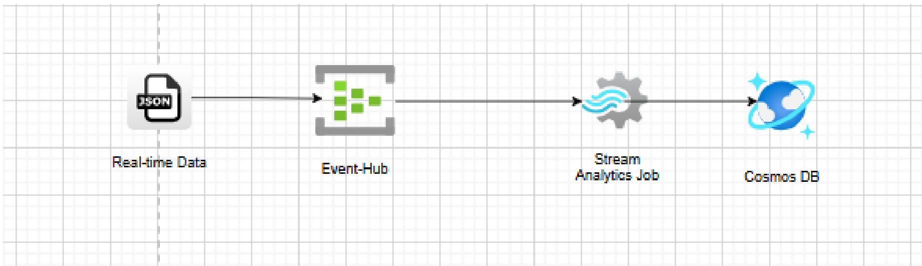


Fig. Real time data entering Cosmos DB

Triggering Events :

- **Fraud Detection Alerts** : If a transaction amount exceeds a predefined threshold or an unusual pattern is detected, an Azure Function triggers an alert.
- **Service Availability Monitoring** : If a bank service is down or slow, an Event Grid notification is sent to the IT support team for immediate action.
- **Real-Time Risk Assessment** : High-risk transactions (e.g., large withdrawals, rapid transactions in
  different locations) trigger real-time risk assessment models to flag suspicious activity.

7. Solution Design & Integration

- **Data Extraction from Multiple Sources** : Historical data is pulled from blobstorage via Azure Data Factory (ADF), and real-time data is ingested using Azure Event Hubs from banking transactions.

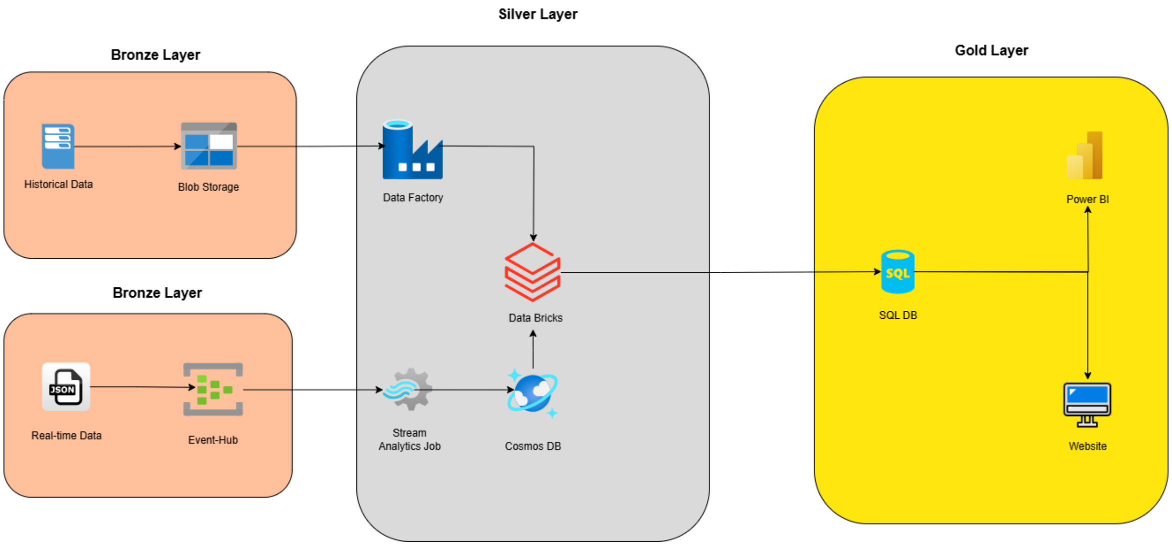
- **Transformation & Cleansing in Databricks :** Raw data is processed in Azure Databricks using Apache Spark, including data validation, missing value handling, and feature engineering.

- **Loading into Medallion Architecture :** Processed data is stored in Azure Data Lake Storage (ADLS) following the Bronze (raw), Silver (cleaned), and Gold (aggregated) layers.

- **Batch & Real-Time Processing Integration :** Historical transaction data is processed in batch mode while real-time streaming is handled with Stream Analytics.

- **Orchestration Using Azure Data Factory :** ADF Pipelines automate data movement between layers, trigger transformations, and integrate batch + real-time processing for seamless operations.

System Architecture



Automated Workflows:

- **ADF Pipeline for ETL Automation** : ADF triggers data extraction, transformation, and storage using scheduled pipelines, reducing manual intervention.
- **Event-Driven Processing with Azure Functions :** Fraud detection triggers real-time alerts based on transaction anomalies, ensuring quick action.
- **CI/CD Deployment with Azure DevOps :** Automated deployments using GitHub Actions and Azure DevOps Pipelines, ensuring seamless updates to ETL workflows.

8. Implementation and Results

Architecture:

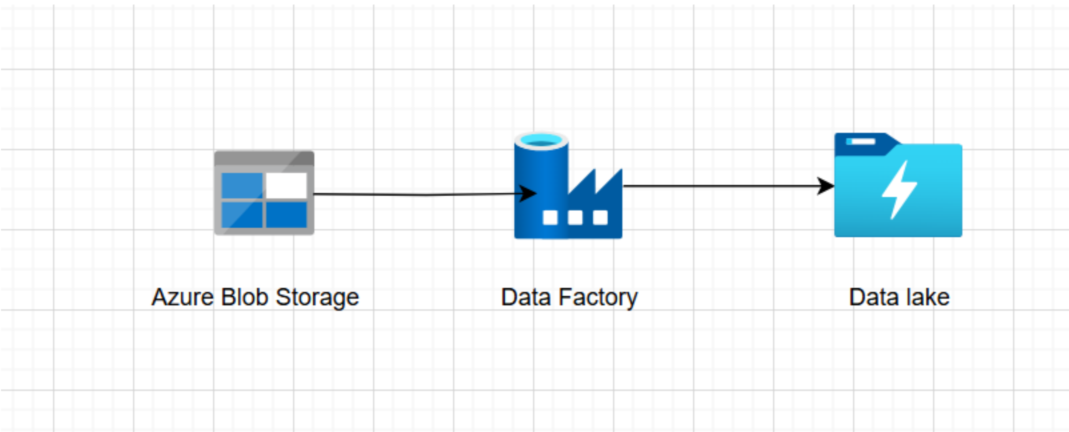


Fig. Historical data flow

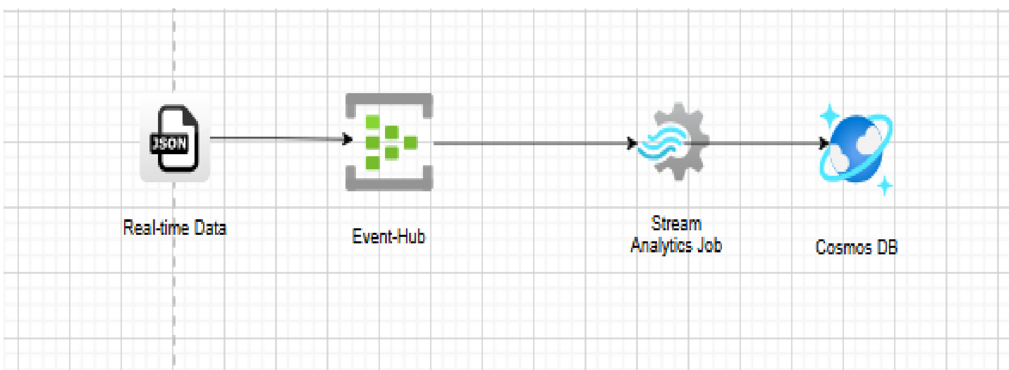


Fig. Real time data flow

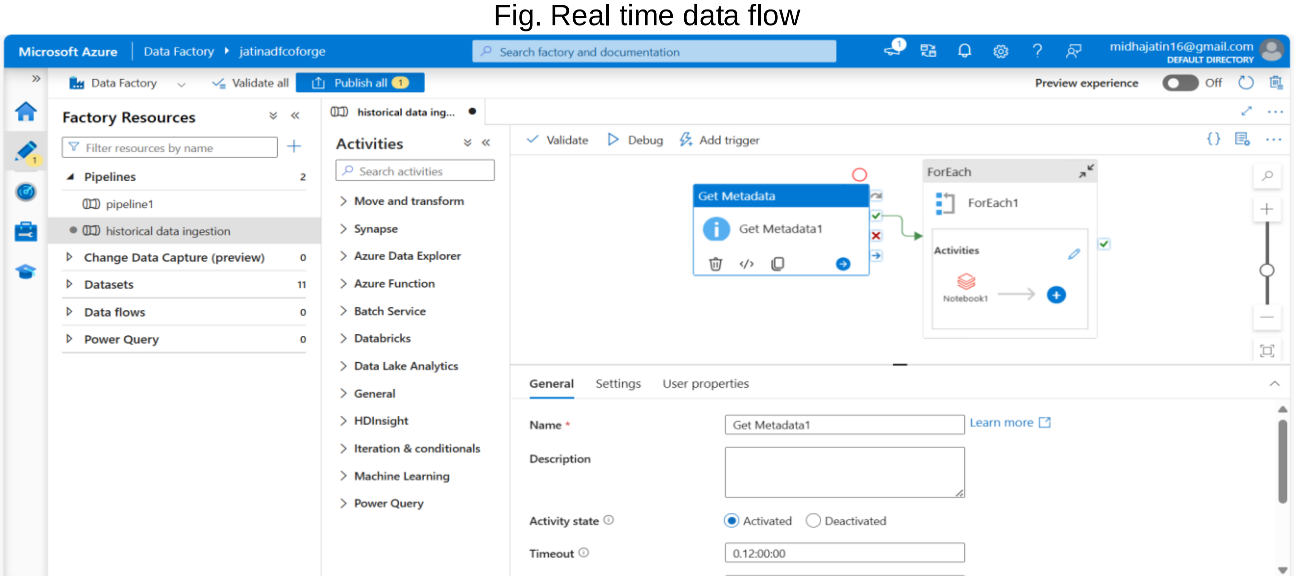


Fig. Ingesting historical data from blob to data bricks towards delta lake

```
# Connection details as variables
cosmos_endpoint = "https://jatincosmosdb.documents.azure.com:443/"
cosmos_key = "IOvGwpuVgubfdwdSgHknWaUfPFU8BJYKoX9TMOkac5i7Wbv24G5MSvOy8iVt9sE2TXl10rGiI5sBACDbb69zrw=="
database_name = "jatincosmosdb"
container_name = "livedatastorage"
```



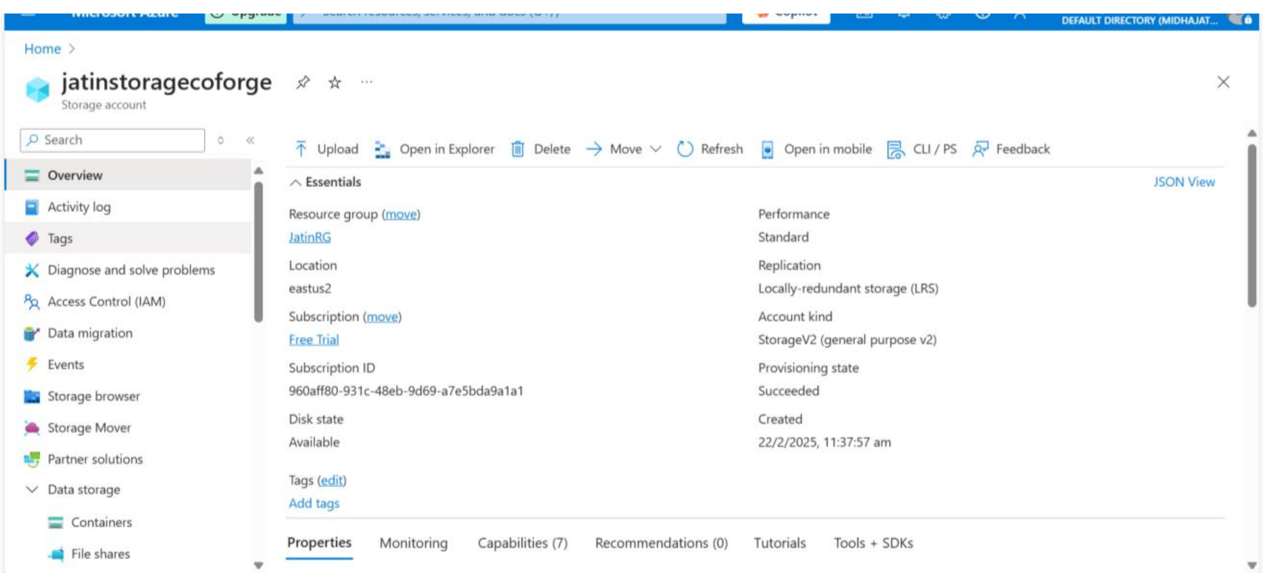
```
# Read from Cosmos DB
cosmos_df = spark.read \
    .format("cosmos.oltp") \
    .option("spark.cosmos.accountEndpoint", cosmos_endpoint) \
    .option("spark.cosmos.accountKey", cosmos_key) \
    .option("spark.cosmos.database", database_name) \
    .option("spark.cosmos.container", container_name) \
    .load()

# Select only required columns (remove Cosmos DB metadata)
cosmos_realtime_df = cosmos_df.select(
    col("transaction_id"),
    col("customer_id"),
    col("transaction_date"),
    col("product_id"),
    col("quantity"),
    col("payment_type"),
    col("transaction_amount")
)
```

Connecting Cosmos to databricks for transformations

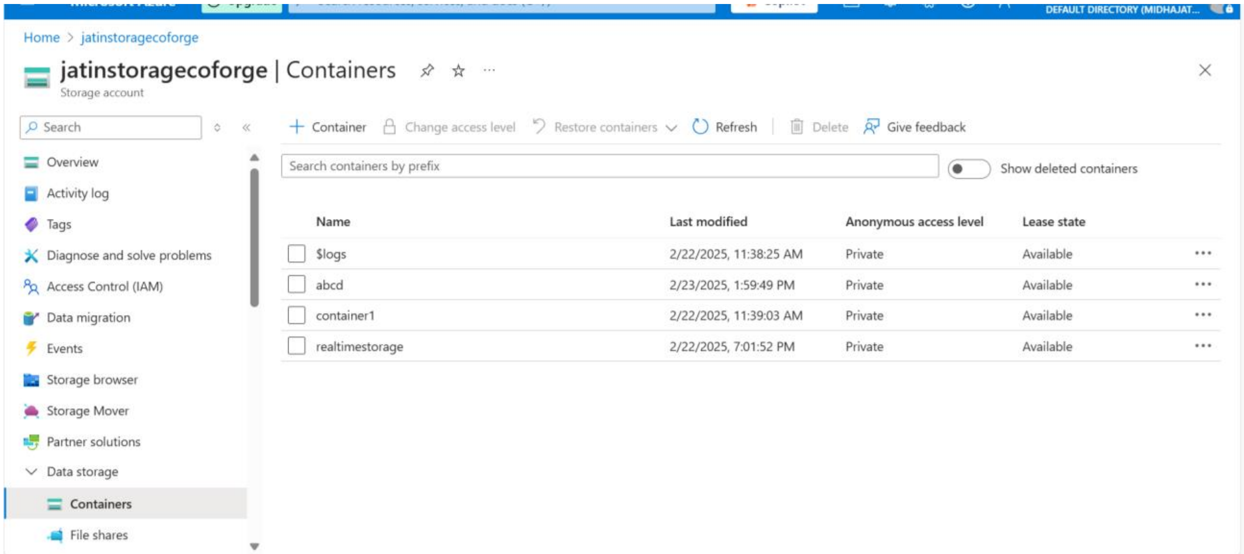
WORKING SCREENSHOTS

For historical data:

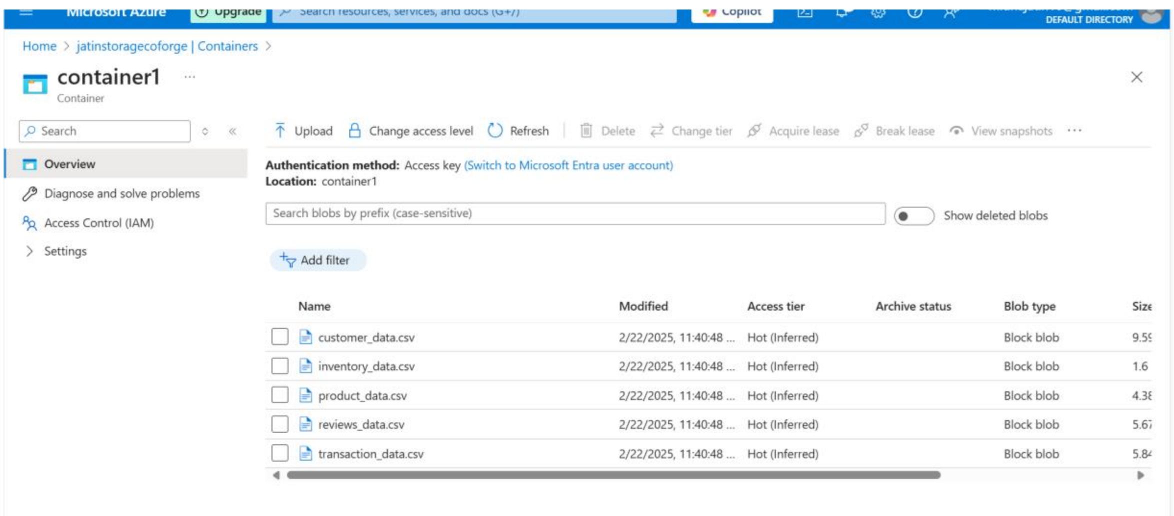
Creating Storage Account



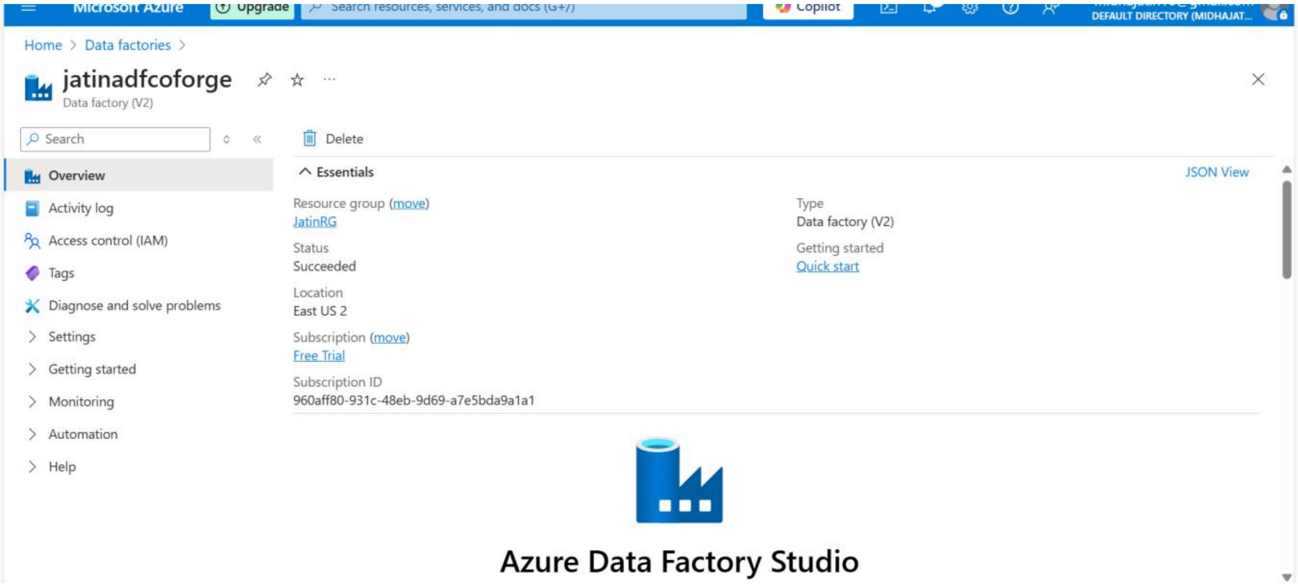
Creating Containers in the storage account



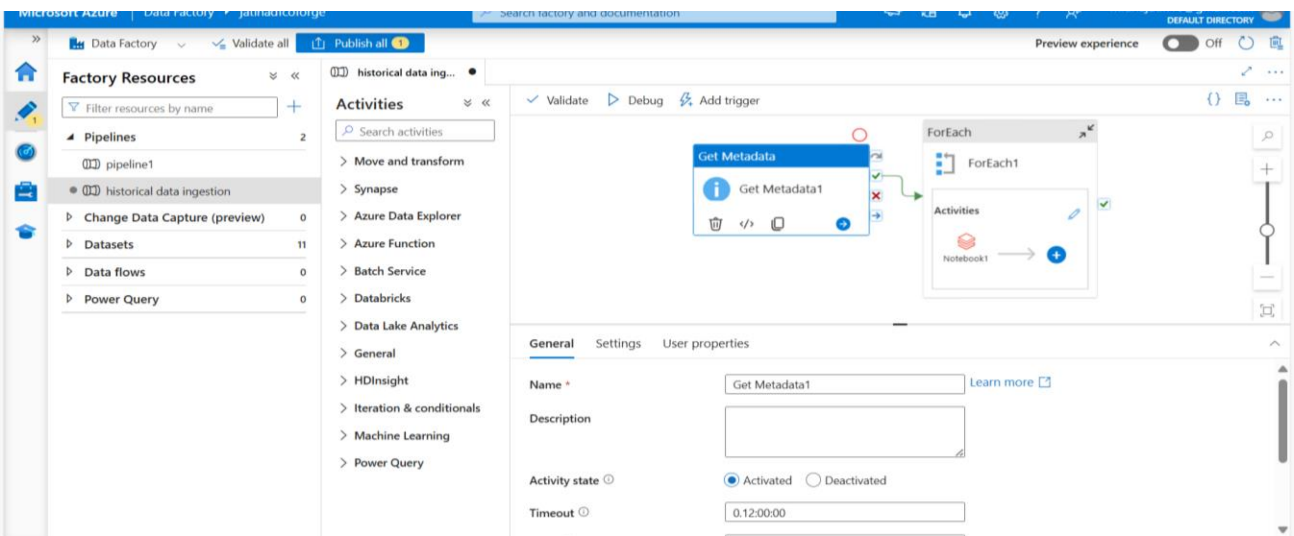
Blob storage where csv's were stored



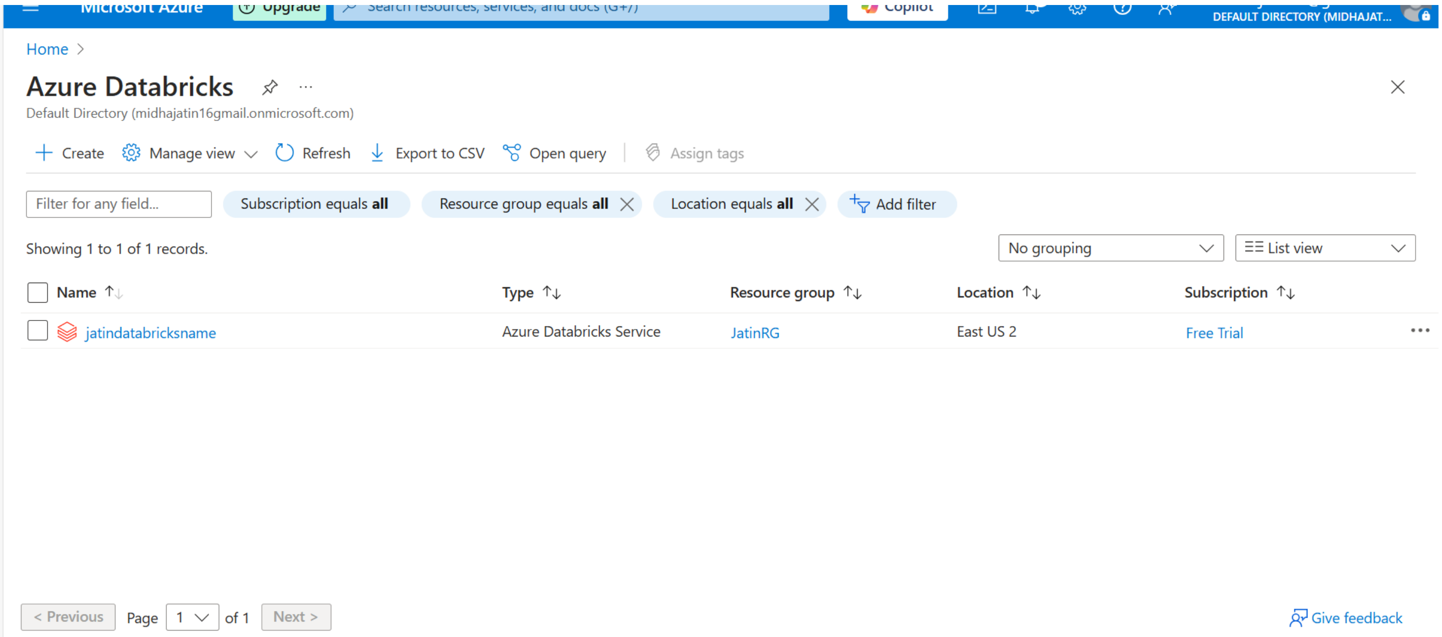
Initializing Data Factory



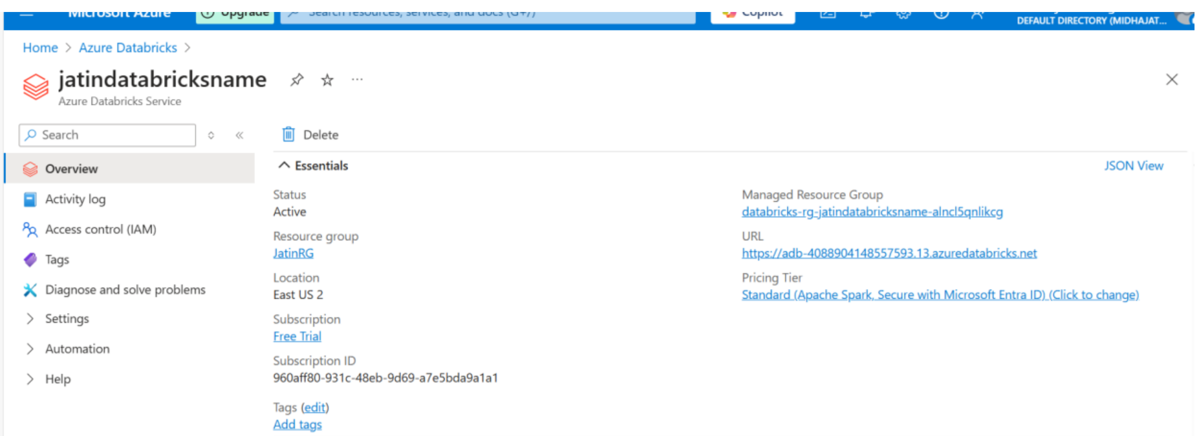
Ingesting data from Blob Storage through Data Factory

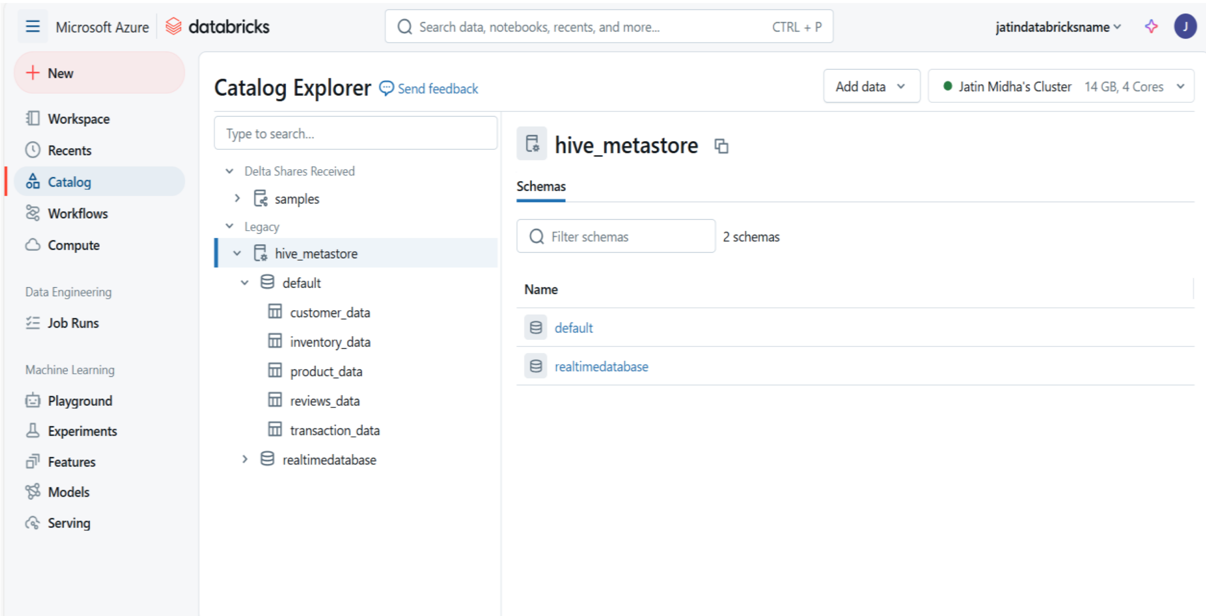


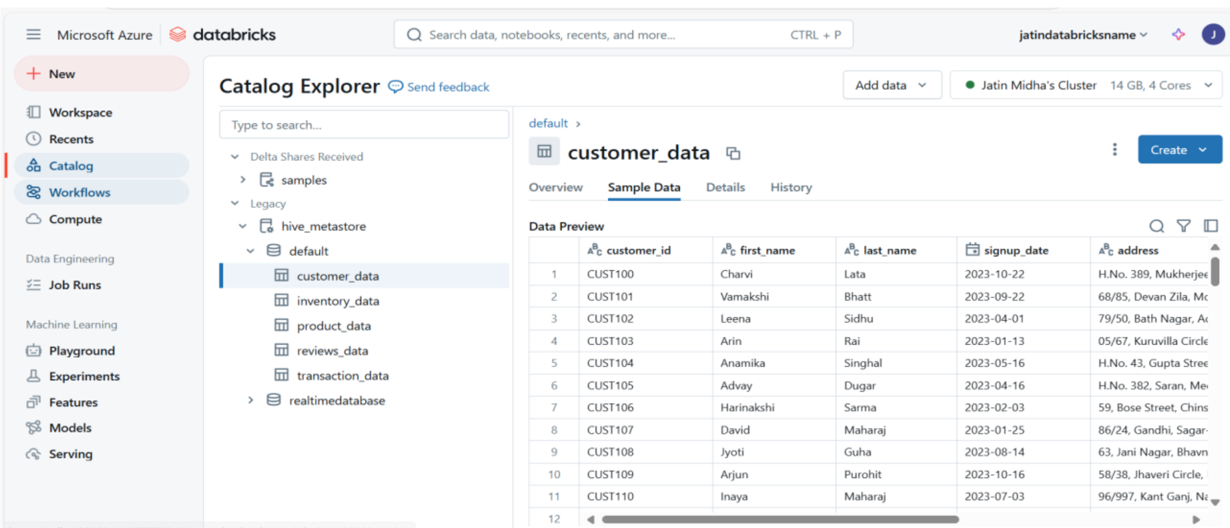
Opening databricks

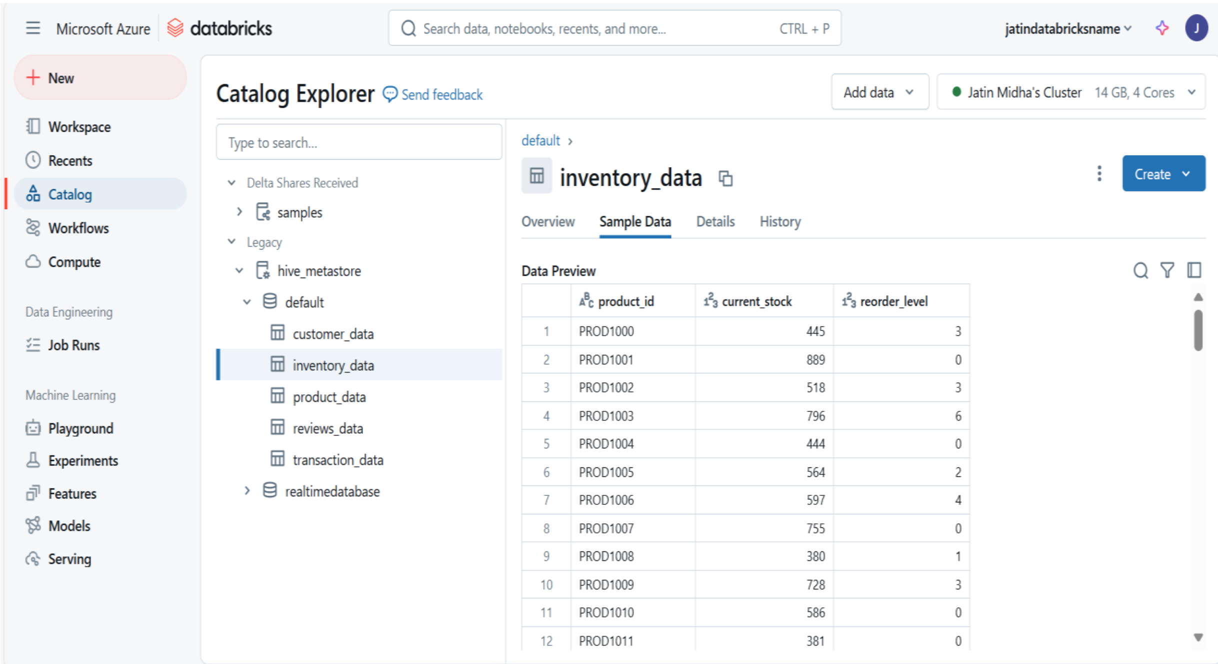


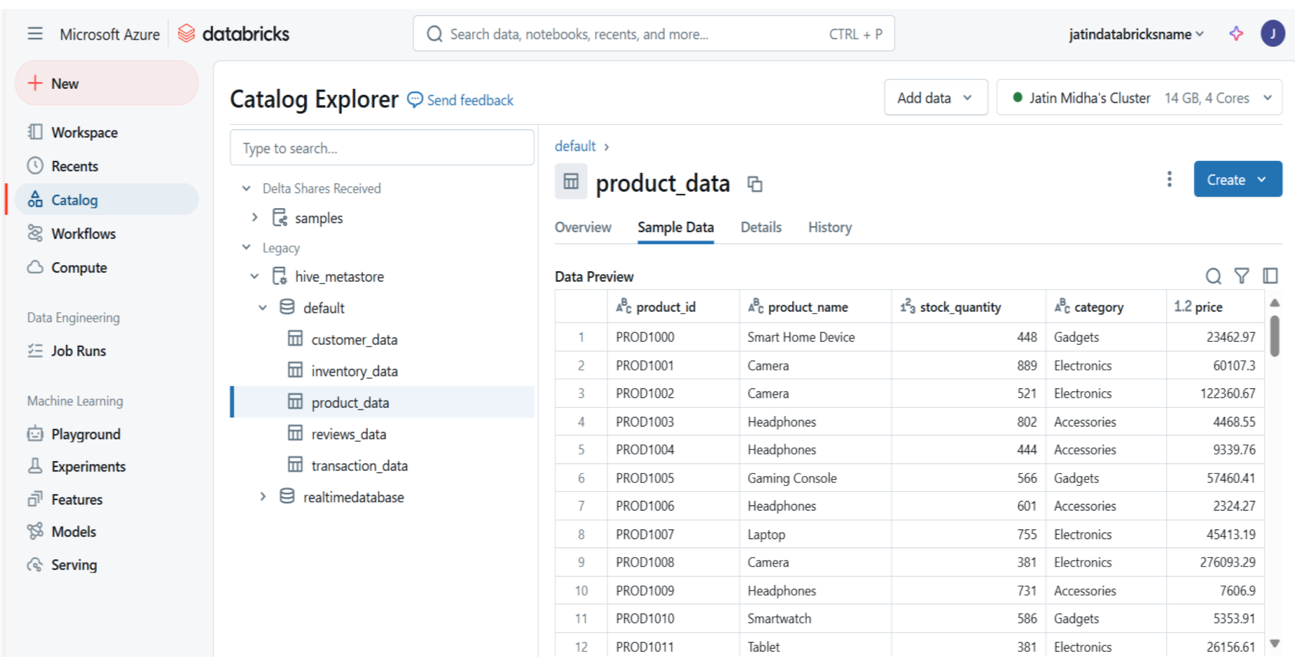
Databricks Details

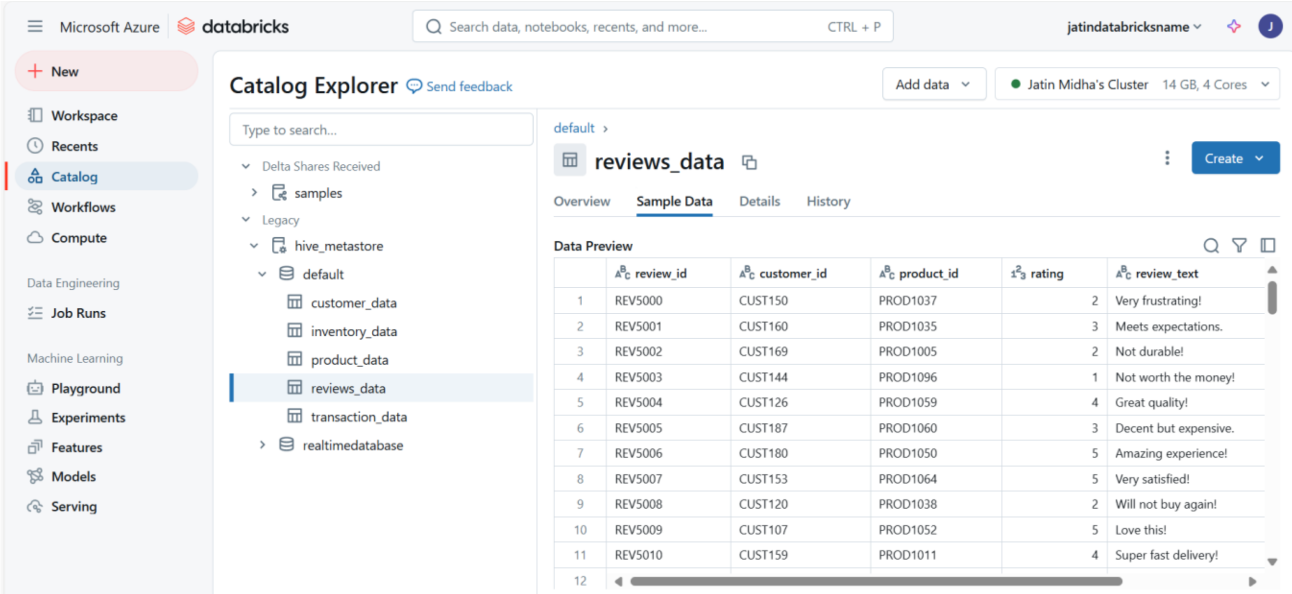


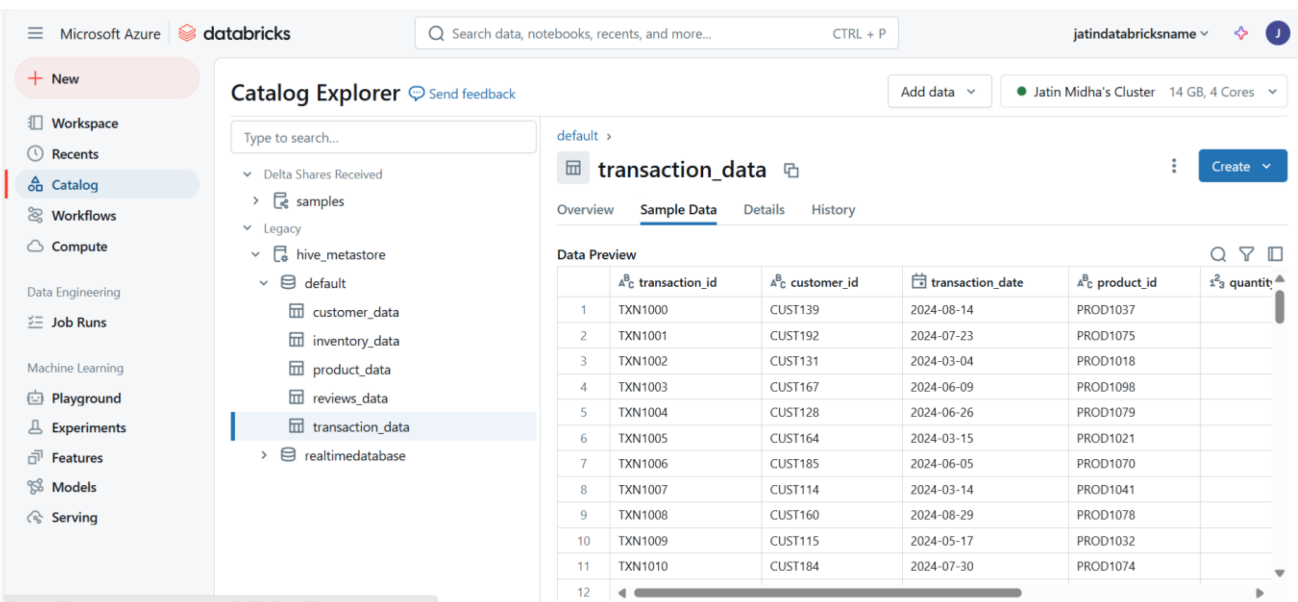

Microsoft Azure
databricks
Search data, notebooks, recents, and more...
CTRL + P
jatindatabricksname
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Catalog Explorer
Send feedback
Add data
Jatin Midha's Cluster 14 GB, 4 Cores
Type to search...
Delta Shares Received
samples
Legacy
hive_metastore
default
customer_data
inventory_data
product_data
reviews_data
transaction_data
realtimedatabase
hive_metastore
Schemas
Filter schemas
2 schemas
Name
default
realtimedatabase


Microsoft Azure
databricks
Search data, notebooks, recents, and more...
CTRL + P
jatindatabricksname
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Catalog Explorer
Send feedback
Add data
Jatin Midha's Cluster 14 GB, 4 Cores
Type to search...
Delta Shares Received
samples
Legacy
hive_metastore
default
customer_data
inventory_data
product_data
reviews_data
transaction_data
realtimedatabase
default
customer_data
Create
Overview
Sample Data
Details
History
Data Preview
customer_id first_name last_name signup_date address
1 CUST100 Charvi Lata 2023-10-22
2 CUST101 Vamakshi Bhatt 2023-09-22
3 CUST102 Leena Sidhu 2023-04-01
4 CUST103 Arin Rai 2023-01-13
5 CUST104 Anamika Singhal 2023-05-16
6 CUST105 Advay Dugar 2023-04-16
7 CUST106 Harinakshi Sarma 2023-02-03
8 CUST107 David Maharaj 2023-01-25
9 CUST108 Jyoti Guha 2023-08-14
10 CUST109 Arjun Purohit 2023-10-16
11 CUST110 Inaya Maharaj 2023-07-03
12


Microsoft Azure
databricks
Search data, notebooks, recents, and more...
CTRL + P
jatindatabricksname
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Catalog Explorer
Send feedback
Add data
Jatin Midha's Cluster 14 GB, 4 Cores
Type to search...
Delta Shares Received
samples
Legacy
hive_metastore
default
customer_data
inventory_data
product_data
reviews_data
transaction_data
realtimedatabase
default
inventory_data
Create
Overview
Sample Data
Details
History
Data Preview
product_id current_stock reorder_level
1 PROD1000 445 3
2 PROD1001 889 0
3 PROD1002 518 3
4 PROD1003 796 6
5 PROD1004 444 0
6 PROD1005 564 2
7 PROD1006 597 4
8 PROD1007 755 0
9 PROD1008 380 1
10 PROD1009 728 3
11 PROD1010 586 0
12 PROD1011 381 0

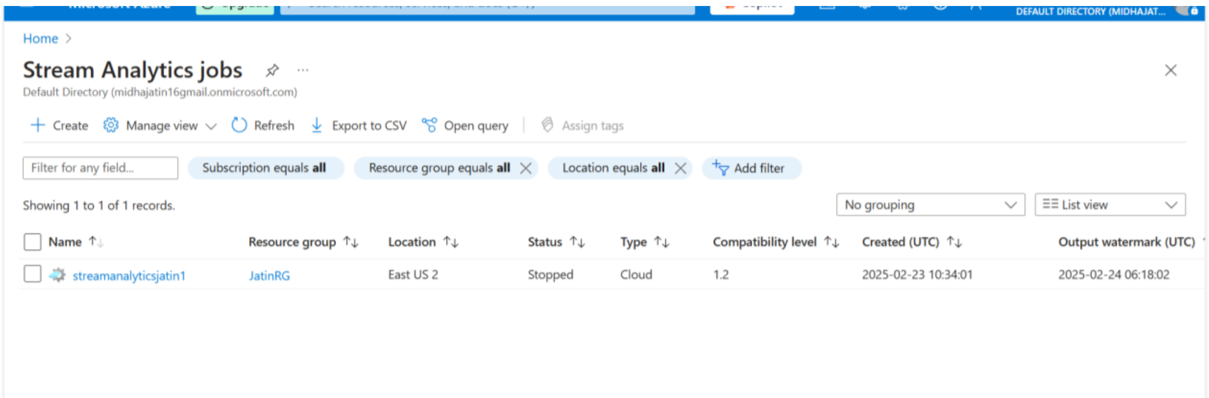

Microsoft Azure
databricks
Search data, notebooks, recents, and more...
CTRL + P
jatindatabricksname
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Catalog Explorer
Send feedback
Add data
Jatin Midha's Cluster 14 GB, 4 Cores
Type to search...
Delta Shares Received
samples
Legacy
hive_metastore
default
customer_data
inventory_data
product_data
reviews_data
transaction_data
realtimedatabase
default
product_data
Create
Overview
Sample Data
Details
History
Data Preview
product_id product_name stock_quantity category price
1 PROD1000 Smart Home Device 448 Gadgets 23462.97
2 PROD1001 Camera 889 Electronics 60107.3
3 PROD1002 Camera 521 Electronics 122360.67
4 PROD1003 Headphones 802 Accessories 4468.55
5 PROD1004 Headphones 444 Accessories 9339.76
6 PROD1005 Gaming Console 566 Gadgets 57460.41
7 PROD1006 Headphones 601 Accessories 2324.27
8 PROD1007 Laptop 755 Electronics 45413.19
9 PROD1008 Camera 381 Electronics 276093.29
10 PROD1009 Headphones 731 Accessories 7606.9
11 PROD1010 Smartwatch 586 Gadgets 5353.91
12 PROD1011 Tablet 381 Electronics 26156.61

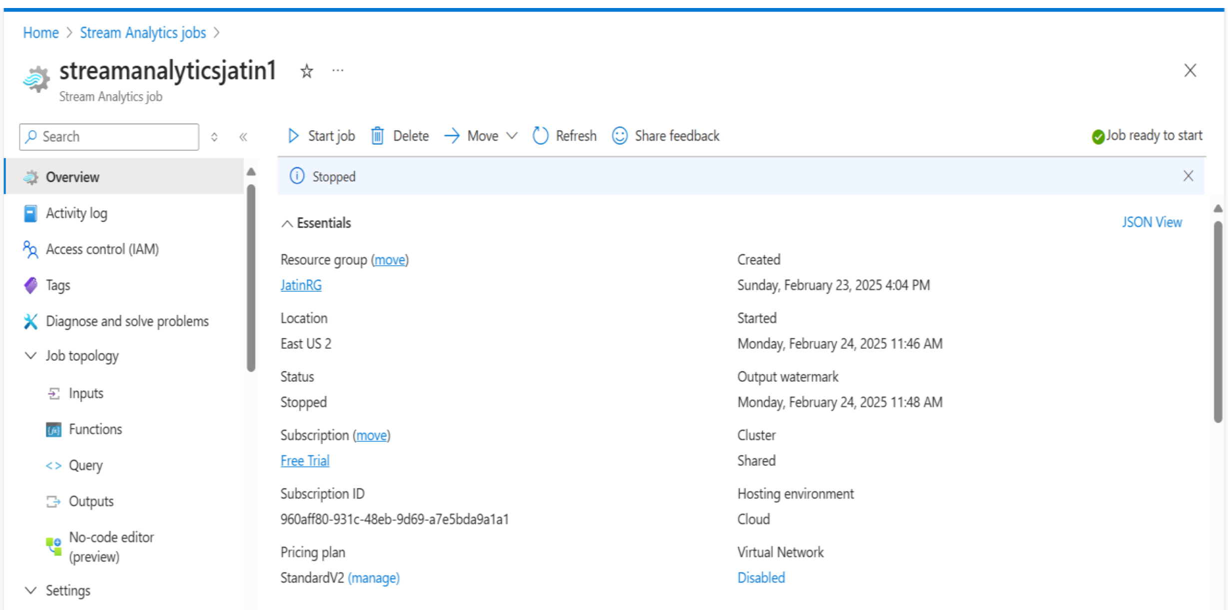




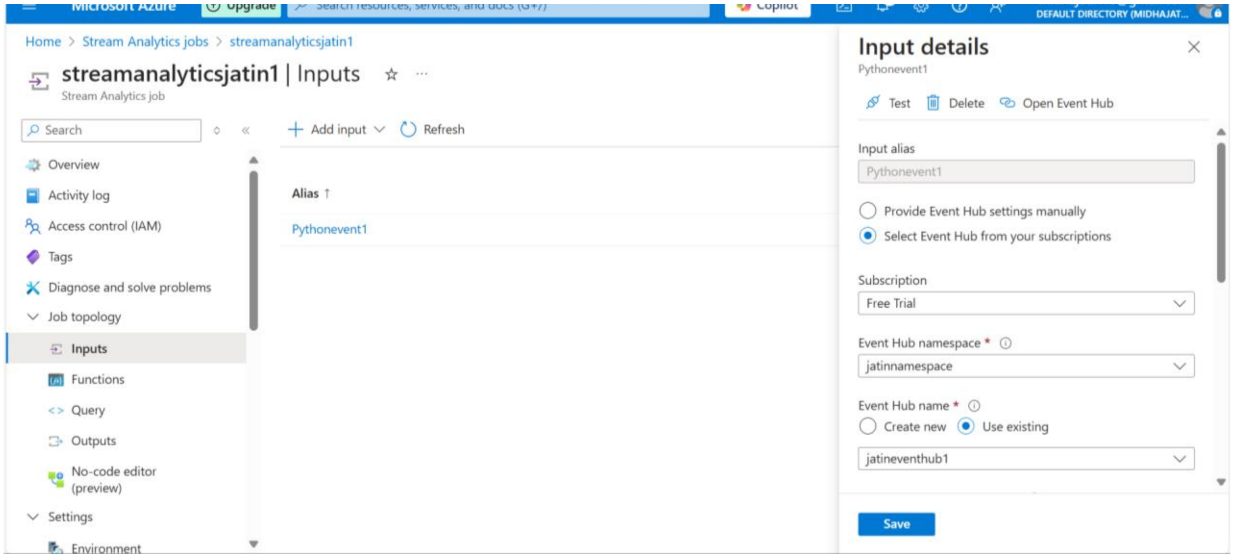
For Real-Time Data

Initializing Stream Analytics Job

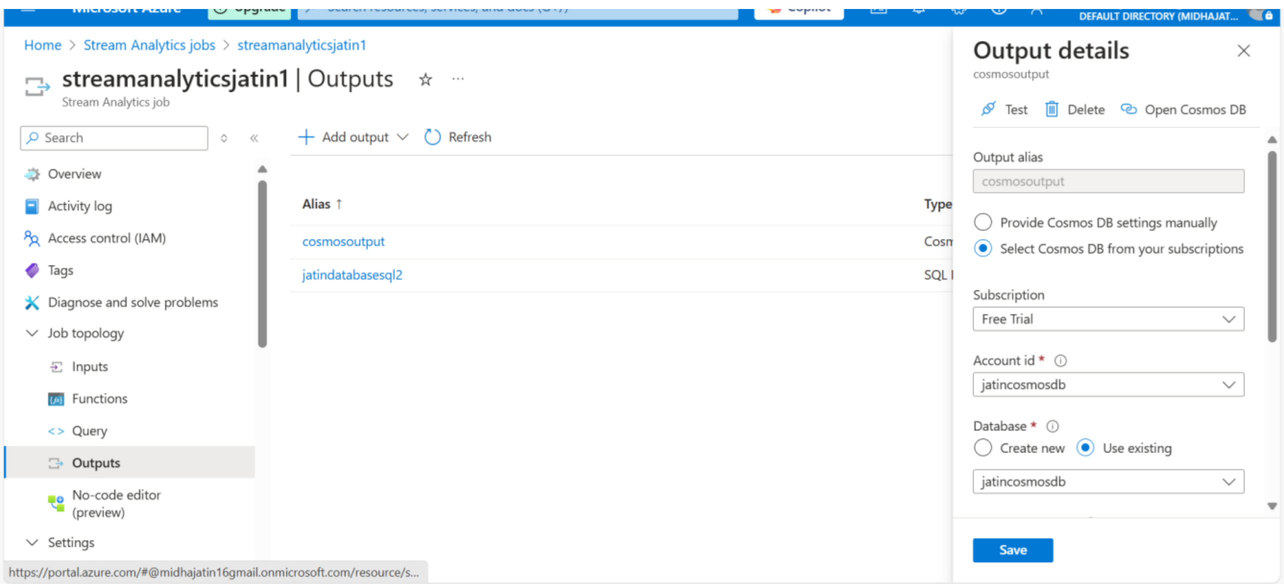




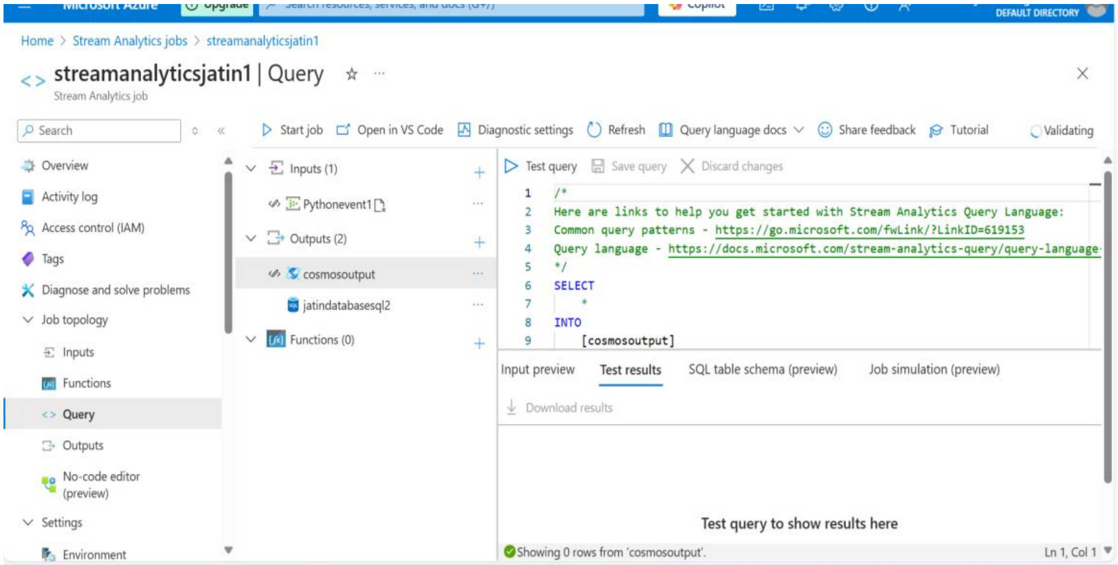
Setting up Input



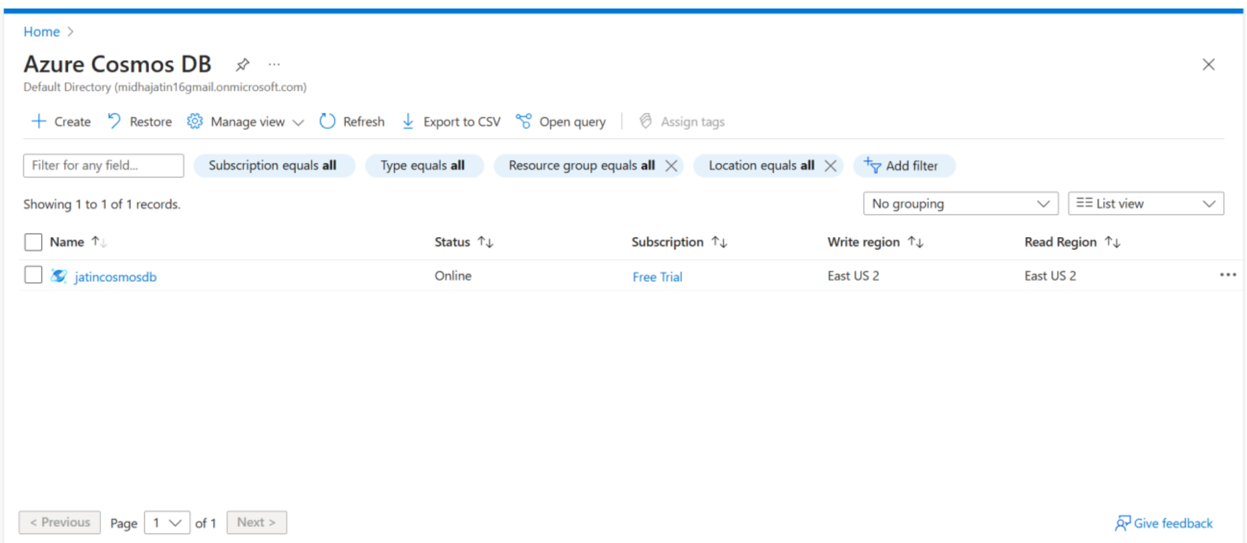
Setting up Output



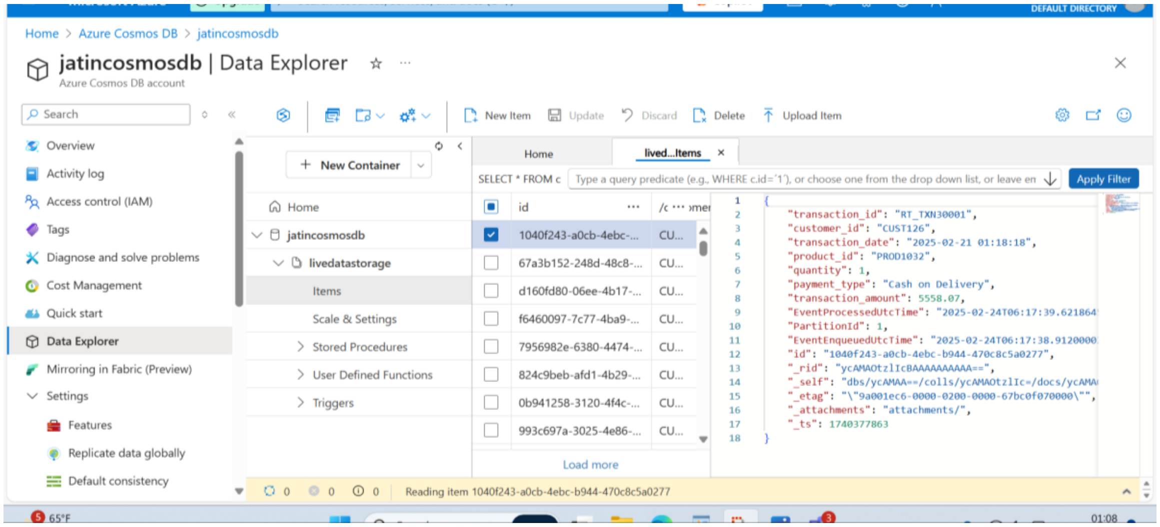
Using Stream Analytics Job to ingest real time data



Setting up Cosmos DB



Data is sent to Cosmos DB



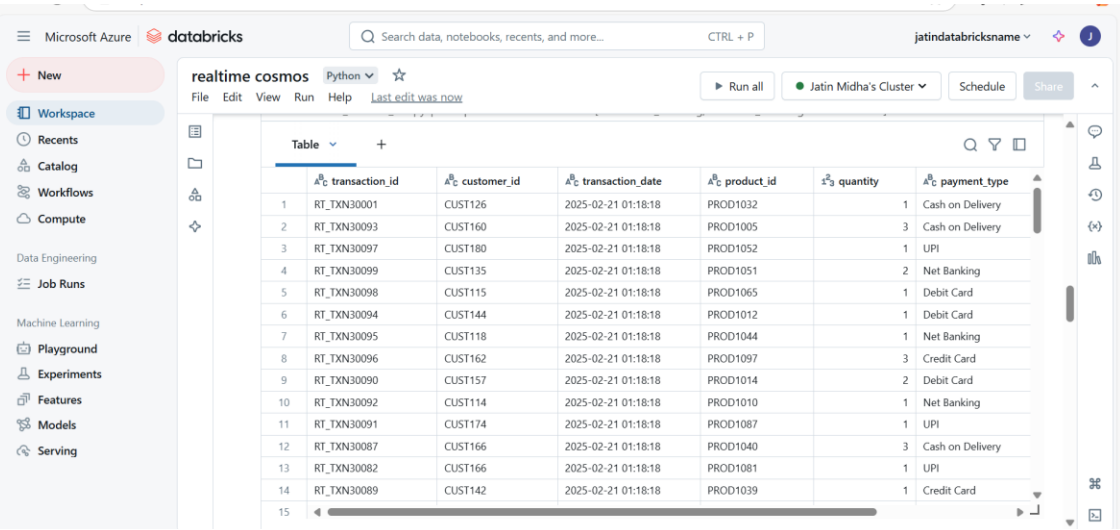

```
from pyspark.sql.functions import col

# Connection details as variables
cosmos_endpoint = "https://jatincosmosdb.documents.azure.com:443/"
cosmos_key = "IOvGwpuVgubfdwdSgHknWaUfPFU8BJYKoX9TMOkac5i7Wbv24G5MSvOy8iVt9sE2TXl10rGiI5sBACDbb69zrw=="
database_name = "jatincosmosdb"
container_name = "livedatastorage"

# Read from Cosmos DB
cosmos_df = spark.read \
    .format("cosmos.oltp") \
    .option("spark.cosmos.accountEndpoint", cosmos_endpoint) \
    .option("spark.cosmos.accountKey", cosmos_key) \
    .option("spark.cosmos.database", database_name) \
    .option("spark.cosmos.container", container_name) \
    .load()

# Select only required columns (remove Cosmos DB metadata)
cosmos_realtime_df = cosmos_df.select(
    col("transaction_id"),
    col("customer_id"),
    col("transaction_date"),
    col("product_id"),
    col("quantity"),
    col("payment_type"),
    col("transaction_amount")
)

# Show data
display(cosmos_realtime_df)
```



```
%sql
create database if not exists realtimedatabase;
```

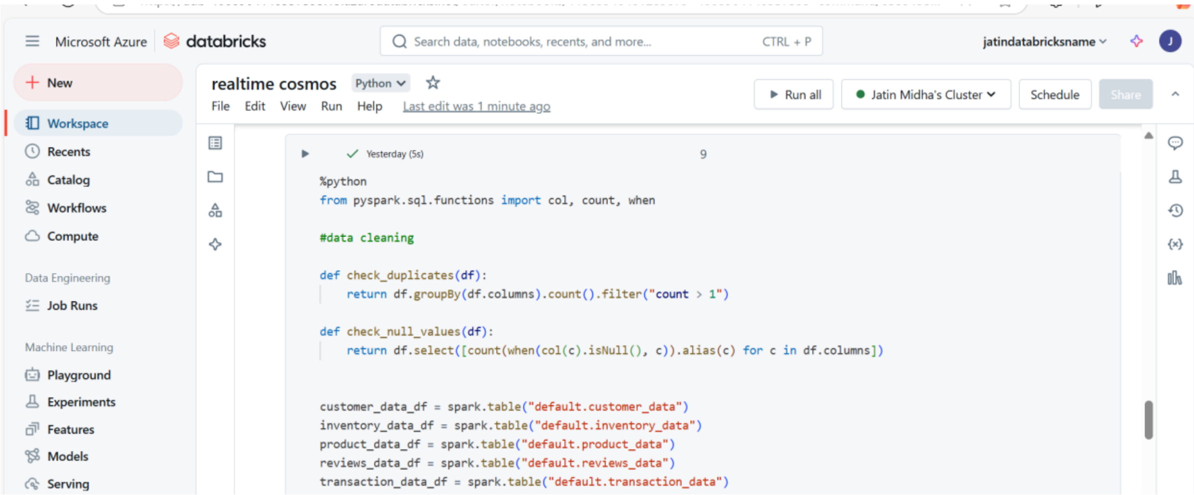
```python
# Save as a Delta table in Hive Metastore
database_name = "realtimedatabase"
table_name = "real_time_data_table"
path = f'dbfs:/user/hive/warehouse/realtimedatabase.db/{table_name}'
cosmos_realtime_df.write.format("delta").mode("overwrite").option("path",
path).saveAsTable(f'{database_name}.{table_name}')

# Print confirmation message
print(f" Data successfully saved as a Delta table: {table_name}")
```

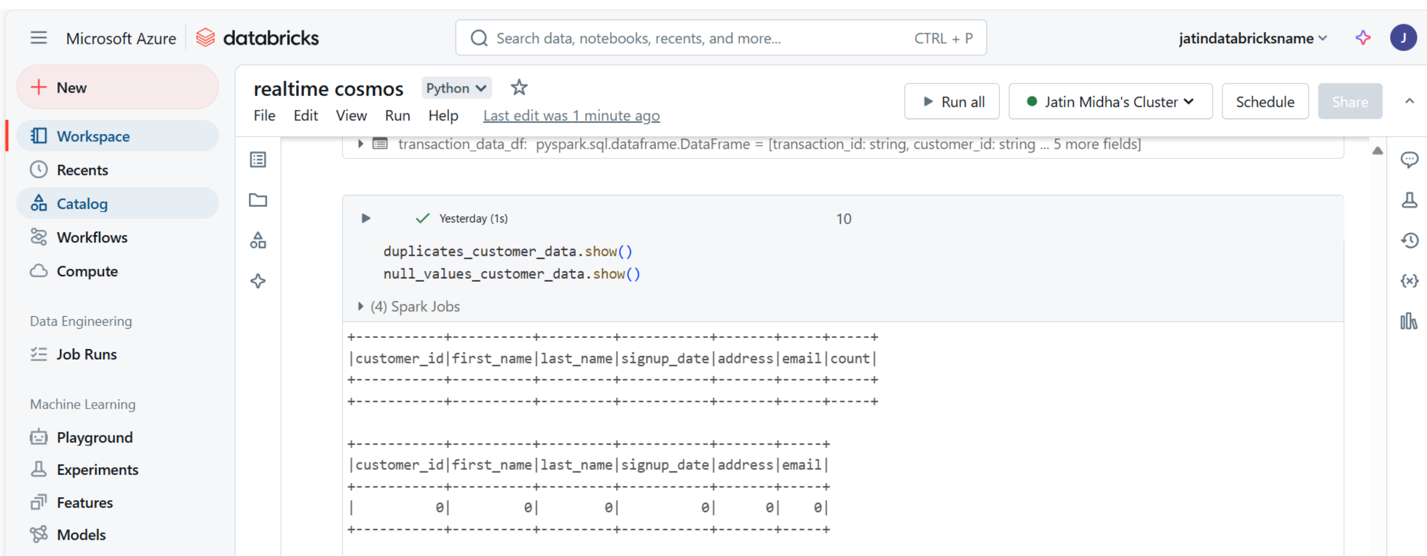
Data loaded from Cosmos to Databricks



	transaction_id	customer_id	transaction_date	product_id	quantity	payment_type
1	RT_TXN30001	CUST126	2025-02-21 01:18:18	PROD1032	1	Cash on Delivery
2	RT_TXN30093	CUST160	2025-02-21 01:18:18	PROD1005	3	Cash on Delivery
3	RT_TXN30097	CUST180	2025-02-21 01:18:18	PROD1052	1	UPI
4	RT_TXN30099	CUST135	2025-02-21 01:18:18	PROD1051	2	Net Banking
5	RT_TXN30098	CUST115	2025-02-21 01:18:18	PROD1065	1	Debit Card
6	RT_TXN30094	CUST144	2025-02-21 01:18:18	PROD1012	1	Debit Card
7	RT_TXN30095	CUST118	2025-02-21 01:18:18	PROD1044	1	Net Banking
8	RT_TXN30096	CUST162	2025-02-21 01:18:18	PROD1097	3	Credit Card
9	RT_TXN30090	CUST157	2025-02-21 01:18:18	PROD1014	2	Debit Card
10	RT_TXN30092	CUST114	2025-02-21 01:18:18	PROD1010	1	Net Banking
11	RT_TXN30091	CUST174	2025-02-21 01:18:18	PROD1087	1	UPI
12	RT_TXN30087	CUST166	2025-02-21 01:18:18	PROD1040	3	Cash on Delivery
13	RT_TXN30082	CUST166	2025-02-21 01:18:18	PROD1081	1	UPI
14	RT_TXN30089	CUST142	2025-02-21 01:18:18	PROD1039	1	Credit Card
15						

Data checked for cleaning



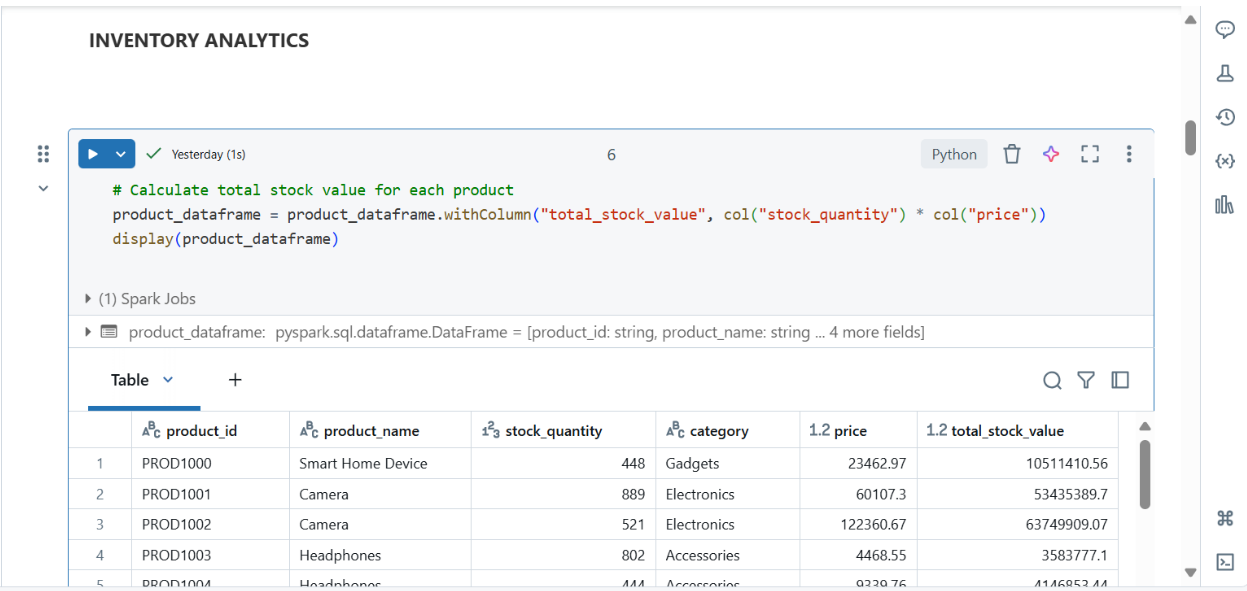


Pyspark Transformations

For Historical Data:

```
from pyspark.sql import SparkSession

spark=SparkSession.builder.appName('Historical Data Transformations').getOrCreate()
product_dataframe=spark.sql("Select * from default.product_data")
```

```python
#transformations

# Calculate average price per category
from pyspark.sql.functions import col, sum, avg
avg_price_per_category = product_dataframe.groupBy("category").agg(avg("price").alias("average_price"))
# Calculate total stock value for each product
product_dataframe = product_dataframe.withColumn("total_stock_value", col("stock_quantity") *
col("price"))
display(product_dataframe)
```



	product_id	product_name	stock_quantity	category	price	total_stock_value
1	PROD1000	Smart Home Device	448	Gadgets	23462.97	10511410.56
2	PROD1001	Camera	889	Electronics	60107.3	53435389.7
3	PROD1002	Camera	521	Electronics	122360.67	63749909.07
4	PROD1003	Headphones	802	Accessories	4468.55	3583777.1
5	PROD1004	Headphones	444	Accessories	9339.76	4146853.44

```python
from pyspark.sql.functions import sum as spark_sum

# Join the DataFrames on customer_id
joined_df = transactions_dataframe.join(customer_dataframe, on="customer_id", how="inner")

# Calculate total transaction amount for each customer
customer_total_amount =
joined_df.groupBy("customer_id").agg(spark_sum("transaction_amount").alias("total_transaction_amount"))

customer_spending_df = joined_df.groupBy("customer_id") \
                                .agg(avg("transaction_amount").alias("avg_transaction")
                                )

joined_df = joined_df.join(customer_total_amount, on="customer_id", how="inner")

joined_df= joined_df.join(customer_spending_df, on="customer_id", how="inner")

# Drop the second 'transaction_amount' column
#joined_df = joined_df.dropDuplicates()
```

```
display(joined_df)
```

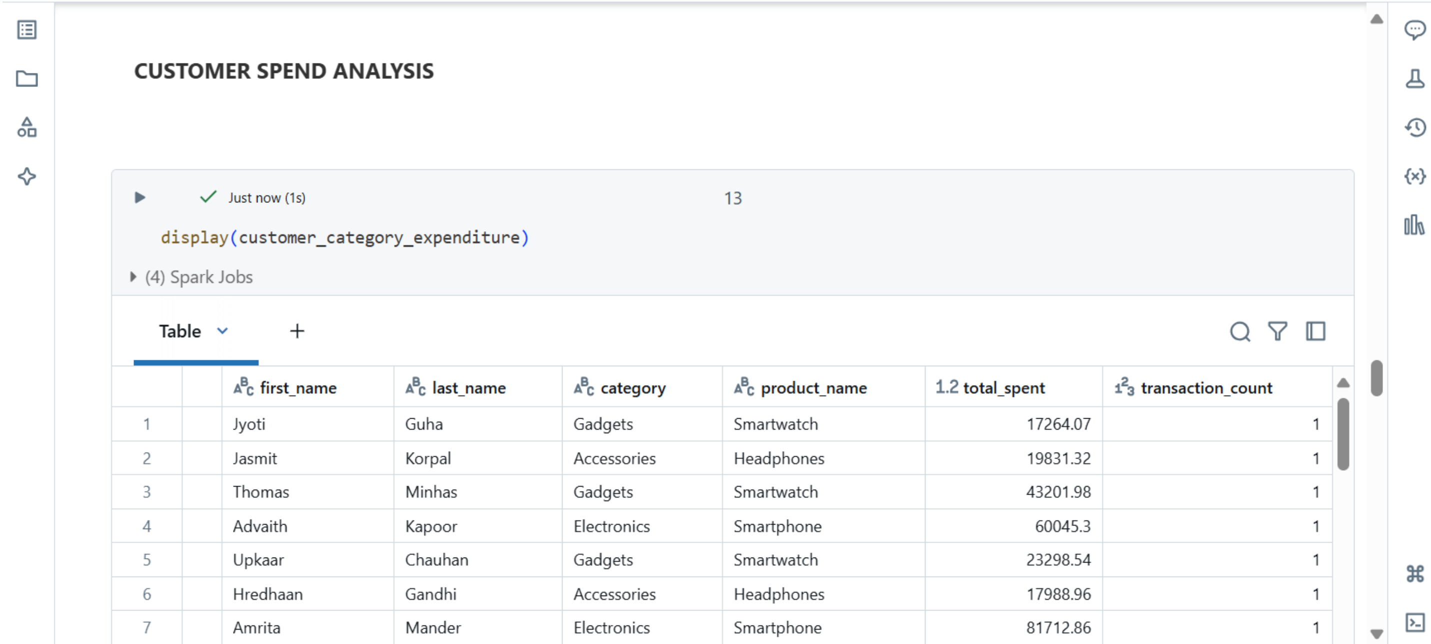
Table

		email	total_transaction_amount	avg_transaction
10	Ganj, Nagpur-632378	inaya.maharaj@yahoo.com	205487.94	205487.94
11	ıchar Chowk, Junagadh-341496	vamakshi.gade@gmail.com	30341.73	30341.73
12	anj, Bathinda-826246	amaira.dara@hotmail.com	21459.12	21459.12
13	ri, Secunderabad-966633	maanas.choudhary@outlook.com	201618.37999999998	67206.12666666666
14	ila, Nangloi Jat-154202	xiti.dhar@hotmail.com	36505.53	18252.765
15	Street, Cuttack-973667	chanakya.dewan@gmail.com	32064.63	32064.63
16	. Nangloi Jat-669019	upma.bhasin@gmail.com	887248.77	295749.59
17	lswa Jahangir Pur 540524	netra.chaudhuri@outlook.com	179008.23	179008.23
18	arula Marg, Gandhinagar-280962	advay.bhattacharyya@outlook.com	382650.83999999997	191325.41999999998
19	ında Road, Gudivada 063965	hredhaan.gandhi@gmail.com	17988.96	17988.96
20	urthy Street, Bhalswa Jahangir Pur-180587	dhriti.badami@hotmail.com	184271.32	92135.66
21	/ Zila, Sonipat 708909	tanish.oak@outlook.com	25059.47	25059.47
22	hwa Ganj, Ajmer 504395	anthony.dara@hotmail.com	4339.47	4339.47
23	ıdy Nagar, Raichur 972560	vritti.gara@yahoo.com	251944.88999999998	125972.44499999999
24	[illegible]	[illegible]	[illegible]	[illegible]

100 rows | 1.52s runtime

Refreshed yesterday

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, count, avg

# Join transaction and product data
joined_df2 = transactions_dataframe.join(product_dataframe, on="product_id")

# Join result with customer data
joined_df2 = joined_df2.join(customer_dataframe, on="customer_id")

customer_category_expenditure = joined_df2.groupBy("customer_id", "first_name", "last_name",
"category", "product_name") \
                                                 .agg(
                                                     sum("transaction_amount").alias("total_spent"),
                                                     count("transaction_id").alias("transaction_count")
                                                 )
```

	first_name	last_name	category	product_name	total_spent	transaction_count
1	Jyoti	Guha	Gadgets	Smartwatch	17264.07	1
2	Jasmit	Korpal	Accessories	Headphones	19831.32	1
3	Thomas	Minhas	Gadgets	Smartwatch	43201.98	1
4	Advaith	Kapoor	Electronics	Smartphone	60045.3	1
5	Upkaar	Chauhan	Gadgets	Smartwatch	23298.54	1
6	Hredhaan	Gandhi	Accessories	Headphones	17988.96	1
7	Amrita	Mander	Electronics	Smartphone	81712.86	1

```
# Install the necessary library
%pip install textblob

from textblob import TextBlob
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

reviews_dataframe = spark.sql("SELECT * FROM default.reviews_data")

# Define a function to analyze sentiment
def analyze_sentiment(review):
    analysis = TextBlob(review)
    if analysis.sentiment.polarity > 0:
        return 'Positive'
    elif analysis.sentiment.polarity == 0:
        return 'Neutral'
    else:
        return 'Negative'

# Register the UDF (User Defined Function)
sentiment_udf = udf(analyze_sentiment, StringType())

# Apply the sentiment analysis function to the review text
reviews_df2 = reviews_dataframe.withColumn("sentiment",
sentiment_udf(reviews_dataframe["review_text"]))

# Display the results
```

```
display(reviews_df2)
```

Table

	customer_id	product_id	rating	review_text	review_date	sentiment
1	CUST150	PROD1037	2	Very frustrating!	2024-01-23	Negative
2	CUST160	PROD1035	3	Meets expectations.	2024-01-22	Neutral
3	CUST169	PROD1005	2	Not durable!	2024-01-18	Neutral
4	CUST144	PROD1096	1	Not worth the money!	2024-02-09	Negative
5	CUST126	PROD1059	4	Great quality!	2024-02-03	Positive
6	CUST187	PROD1060	3	Decent but expensive.	2024-01-27	Negative
7	CUST180	PROD1050	5	Amazing experience!	2024-01-21	Positive
8	CUST153	PROD1064	5	Very satisfied!	2024-02-07	Positive
9	CUST120	PROD1038	2	Will not buy again!	2024-01-29	Neutral
10	CUST107	PROD1052	5	Love this!	2024-01-30	Positive
11	CUST159	PROD1011	4	Super fast delivery!	2024-02-02	Positive
12	CUST128	PROD1076	3	Neutral opinion.	2024-01-11	Neutral
13	CUST177	PROD1072	1	Will not buy again!	2024-02-09	Neutral
14	CUST183	PROD1071	3	Fairly good.	2024-01-26	Positive
15						

100 rows | 15.05s runtime

Refreshed yesterday

```
jdbc_url = "jdbc:sqlserver://jatinsql1.database.windows.net:1433;databaseName=jatindatabasesql2 "
db_properties = {
    "user": "jatin1",
    "password": "Qwertyuiop12",
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}

joined_df.write \
    .format("jdbc") \
    .option("url", jdbc_url) \
    .option("dbtable", "dbo.total_average_transactions") \
    .option("user", db_properties["user"]) \
    .option("password", db_properties["password"]) \
    .option("driver", db_properties["driver"]) \
    .mode("overwrite") \
    .save()
```

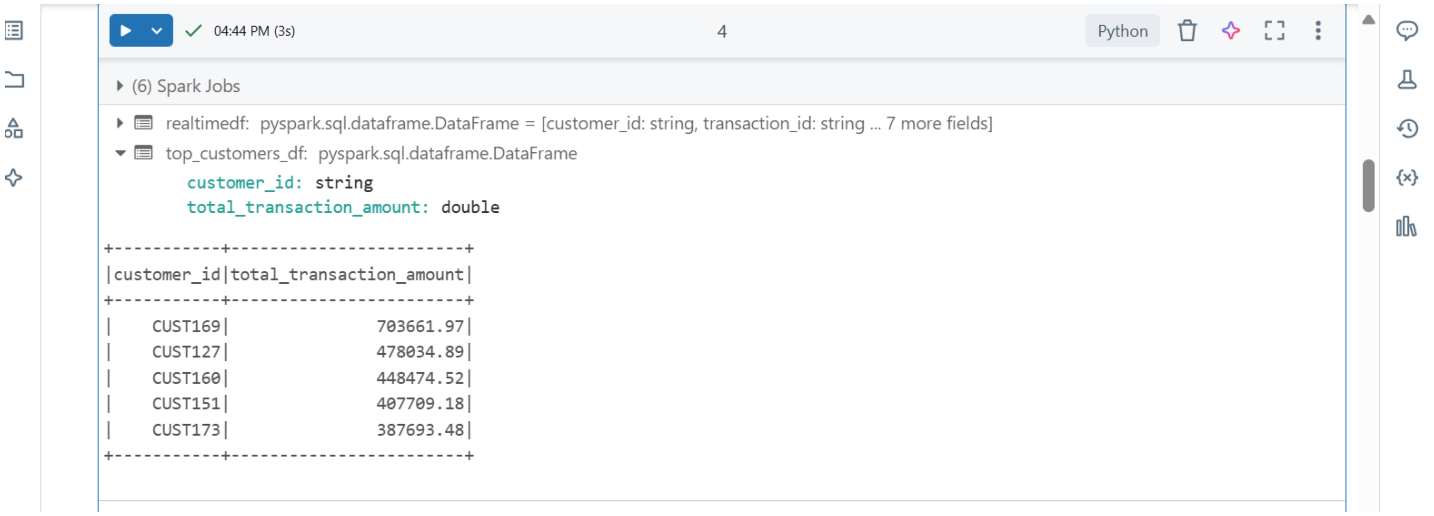
For Real Time Data

Showing top customers by transaction amount

```
from pyspark.sql import SparkSession
```



```
spark=SparkSession.builder.appName("Realtime_trans").getOrCreate()
realtimedf= spark.sql('select * from realtimedatabase.real_time_data_table')

from pyspark.sql.functions import sum
total_revenue_df = realtimedf.agg(sum("transaction_amount").alias("total_revenue"))
total_revenue_df.show()
#showing top customers by transactions amount
top_customers_df =
realtimedf.groupBy("customer_id").agg(sum("transaction_amount").alias("total_transaction_amount"))
top_customers_df = top_customers_df.orderBy("total_transaction_amount", ascending=False).limit(5)
top_customers_df.show()
realtimedf=realtimedf.join(top_customers_df, "customer_id", "left")

jdbc_url = "jdbc:sqlserver://jatinsql1.database.windows.net:1433;databaseName=jatindatabasesql2 "
db_properties = {
    "user": "jatin1",
    "password": "Qwertyuiop12",  # Avoid storing passwords in code
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}

top_customers_df.write \
    .format("jdbc") \
    .option("url", jdbc_url) \
    .option("dbtable", "dbo.top_customers_shorttable_realtime") \
    .option("user", db_properties["user"]) \
    .option("password", db_properties["password"]) \
    .option("driver", db_properties["driver"]) \
    .mode("overwrite") \
    .save()
#top 5 customers with high amount transaction in realtime data
display(top_customers_df)
```


```
jdbc_url = "jdbc:sqlserver://jatinsql1.database.windows.net:1433;databaseName=jatindatabasesql2 "
db_properties = {
    "user": "jatin1",
    "password": "Qwertyuiop12",  # Avoid storing passwords in code
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}

realtimedf.write \
    .format("jdbc") \
    .option("url", jdbc_url) \
    .option("dbtable", "dbo.top_customers_realtime") \
    .option("user", db_properties["user"]) \
    .option("password", db_properties["password"]) \
    .option("driver", db_properties["driver"]) \
    .mode("overwrite") \
    .save()
```

```
from pyspark.sql.functions import count
payment_type_count_df =
realtimedf.groupBy("payment_type").agg(count("transaction_id").alias("transaction_count"))
payment_type_count_df.show()
# Drop the 'payment_type' column
realtimedf = realtimedf.drop("total_quantity_sold")
```

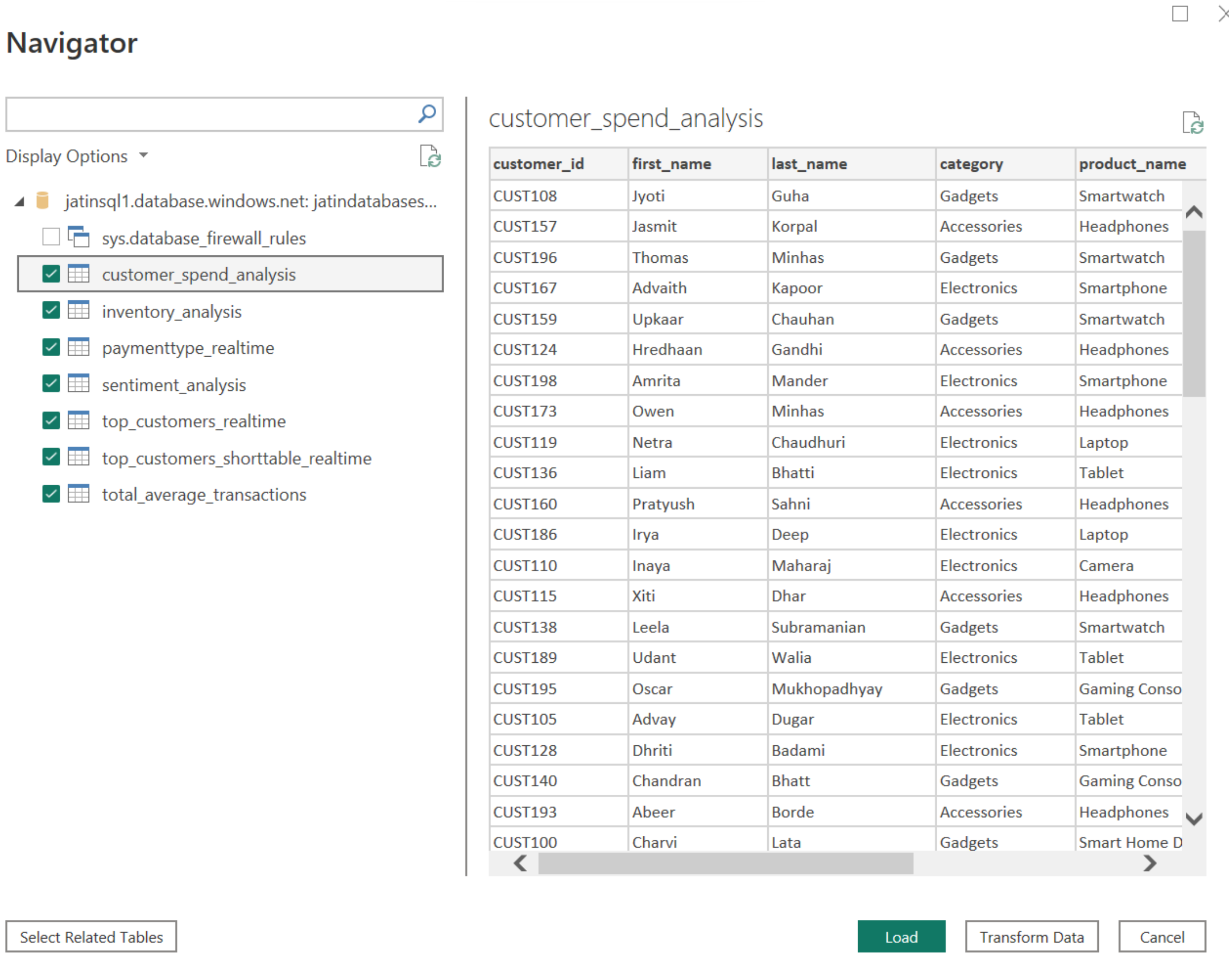


```
display(realtimedf)
```

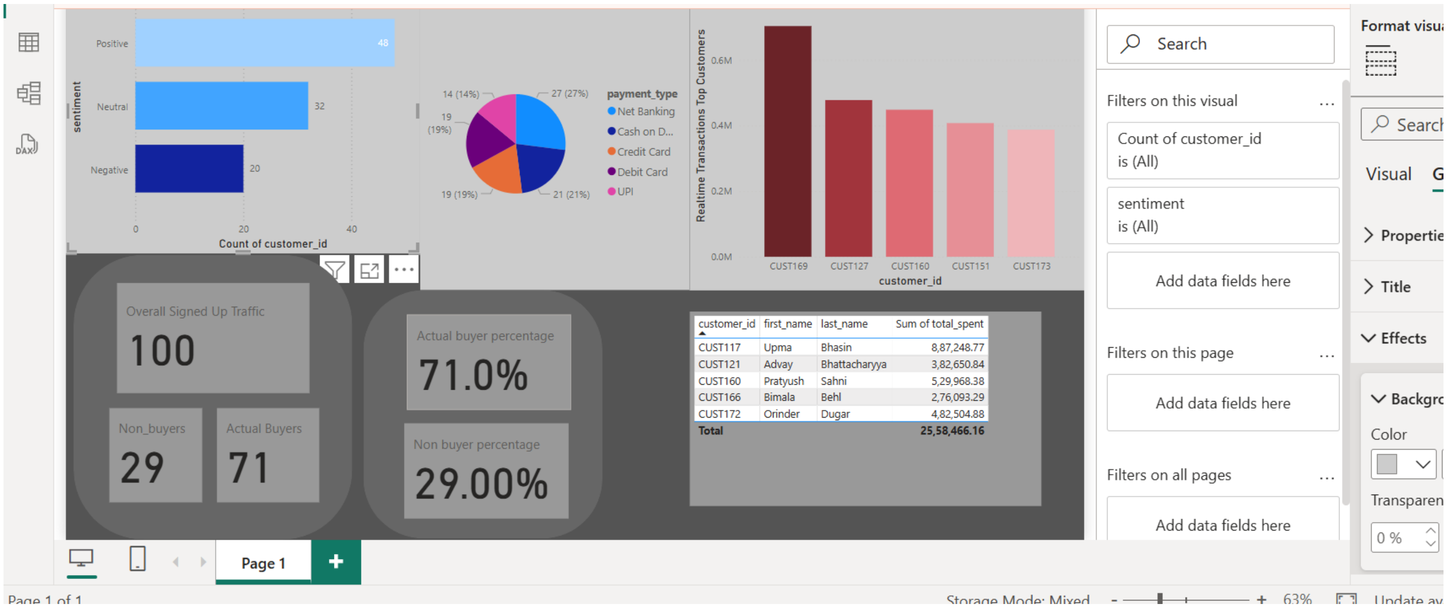
Transaction amount by payment type

```
▸ (5) Spark Jobs
▸ payment_type_count_df:  pyspark.sql.dataframe.DataFrame = [payment_type: string, transaction_count: long]
▸ realtimedf:  pyspark.sql.dataframe.DataFrame = [customer_id: string, transaction_id: string ... 6 more fields]
+----------------+-----------------+
|    payment_type|transaction_count|
+----------------+-----------------+
|     Credit Card|               19|
|     Net Banking|               27|
|      Debit Card|               19|
|Cash on Delivery|               21|
|             UPI|               14|
+----------------+-----------------+
```

Connecting Power Bi to Azure Sql Database and importing tables



customer_id	first_name	last_name	category	product_name
CUST108	Jyoti	Guha	Gadgets	Smartwatch
CUST157	Jasmit	Korpal	Accessories	Headphones
CUST196	Thomas	Minhas	Gadgets	Smartwatch
CUST167	Advaith	Kapoor	Electronics	Smartphone
CUST159	Upkaar	Chauhan	Gadgets	Smartwatch
CUST124	Hredhaan	Gandhi	Accessories	Headphones
CUST198	Amrita	Mander	Electronics	Smartphone
CUST173	Owen	Minhas	Accessories	Headphones
CUST119	Netra	Chaudhuri	Electronics	Laptop
CUST136	Liam	Bhatti	Electronics	Tablet
CUST160	Pratyush	Sahni	Accessories	Headphones
CUST186	Irya	Deep	Electronics	Laptop
CUST110	Inaya	Maharaj	Electronics	Camera
CUST115	Xiti	Dhar	Accessories	Headphones
CUST138	Leela	Subramanian	Gadgets	Smartwatch
CUST189	Udant	Walia	Electronics	Tablet
CUST195	Oscar	Mukhopadhyay	Gadgets	Gaming Conso
CUST105	Advay	Dugar	Electronics	Tablet
CUST128	Dhriti	Badami	Electronics	Smartphone
CUST140	Chandran	Bhatt	Gadgets	Gaming Conso
CUST193	Abeer	Borde	Accessories	Headphones
CUST100	Charvi	Lata	Gadgets	Smart Home D


Positive
Neutral
Negative
sentiment
48
32
20
0
20
40
Count of customer_id
14 (14%)
27 (27%)
19 (19%)
19 (19%)
21 (21%)
payment_type
Net Banking
Cash on D...
Credit Card
Debit Card
UPI
Realtime Transactions Top Customers
0.6M
0.4M
0.2M
0.0M
CUST169
CUST127
CUST160
CUST151
CUST173
customer_id
Overall Signed Up Traffic
100
Non_buyers
29
Actual Buyers
71
Actual buyer percentage
71.0%
Non buyer percentage
29.00%
customer_id first_name last_name Sum of total_spent
CUST117 Upma Bhasin 8,87,248.77
CUST121 Advay Bhattacharyya 3,82,650.84
CUST160 Pratyush Sahni 5,29,968.38
CUST166 Bimala Behl 2,76,093.29
CUST172 Orinder Dugar 4,82,504.88
Total 25,58,466.16
Search
Filters on this visual
Count of customer_id
is (All)
sentiment
is (All)
Add data fields here
Filters on this page
Add data fields here
Filters on all pages
Add data fields here
Format visu
Searc
Visual
Propertie
Title
Effects
Backgro
Color
Transparen
0 %
Page 1
Page 1 of 1
Storage Mode: Mixed
63%
Update av

