

# Azure Cloud Data Engineering for BFSI: Scalable ETL and Real-Time Analytics Solution

## 1. Project Title

Azure Cloud Data Engineering for BFSI: Scalable ETL and Real-Time Analytics Solution

## 2. Student Information

- **Name:** SUDHIR YADAV
- **Project:** DATA ENGINEERING

## 3. Problem Definition and Objectives

**Problem Statement:** A fictional financial institution processes millions of transactions daily and needs to analyze this data for fraud detection, risk assessment, and regulatory compliance. The primary challenge is to efficiently aggregate, process, and store both historical and real-time data streams to derive actionable insights and optimize operational efficiency.

### Objectives:

- Ingest historical data from Azure Blob Storage.
- Stream real-time data from Azure Event Hub for real-time analytics.
- Transform data using Azure Databricks for cleaning, aggregation, and optimization.
- Optimize data storage by implementing cost-efficient Medallion Architecture.
- Generate visual reports using Power BI to provide business insights.

## 4. Data Collection, EDA, and Preprocessing

**Data Collection:** The dataset consists of historical CSV files and real-time JSON files:

- **Transactional Data:** Historical and real-time transaction data.
- **Customer Data:** Customer demographic details.
- **Merchant Data:** Merchant-related analytics.
- **Fraud Detection Data:** Identifying suspicious activity.
- **Regulatory Compliance Data:** Ensuring adherence to financial regulations.

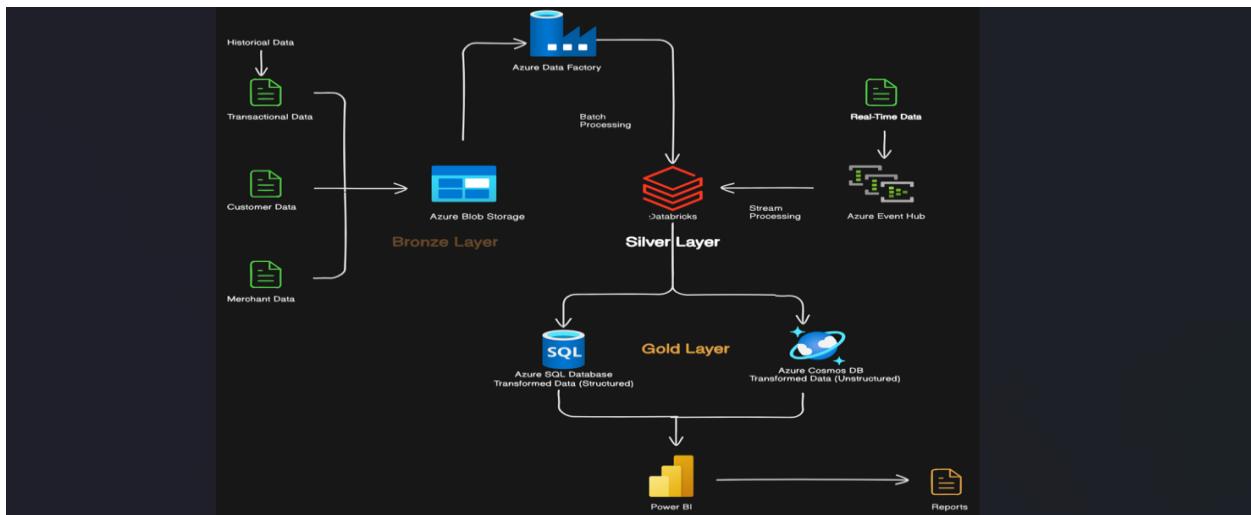
### Exploratory Data Analysis (EDA):

- **Identified missing values and handled them via imputation:** Missing values were detected in several columns. Imputation techniques such as mean, median, and mode were used to fill in these gaps, ensuring the dataset remained robust.
- **Checked data distribution for outliers and inconsistencies:** Data distributions were analyzed using histograms and box plots. Outliers were identified and addressed through techniques like capping and transformation to maintain data integrity.

#### Preprocessing Steps:

- **Handle Missing Data:** Imputation to clean the dataset: Missing data was handled using imputation methods to ensure no loss of valuable information.
- **Data Formatting:** Standardized date format for unified structure: Dates were standardized to a common format (e.g., YYYY-MM-DD) to ensure consistency across the dataset.
- **Feature Engineering:** Aggregated transactions for enhanced customer analytics: Transactions were aggregated to create new features that provide deeper insights into customer behavior.

#### Diagram: High-Level Overview:



## 5. Data Storage and Optimization

- **Storage Architecture: Medallion Approach**
  - **Bronze Layer:** Azure Blob Storage for raw historical and real-time data. This layer stores unprocessed data in its original format.

- o **Silver Layer:** Azure Databricks for cleaning and transforming data. This layer processes and refines the data, making it ready for analysis.
- o **Gold Layer:** Azure SQL Database and Azure Cosmos DB for aggregated data analysis. This layer stores highly curated and aggregated data for advanced analytics.
- **Optimization Techniques:**
  - o **Partitioning and indexing for fast querying in SQL:** Data was partitioned and indexed to improve query performance and reduce retrieval times.
  - o **Hierarchical namespace to show medallion architecture:** A hierarchical namespace was used to organize data within the medallion architecture, enhancing data management and accessibility.

## 6. Real-Time Data Processing and Streaming

- **Processing Pipeline:**
  - o **Azure Event Hub receives JSON data from real-time sources:** Real-time data is ingested through Azure Event Hub, which handles high-throughput data streams.
  - o **Azure Databricks performs transformation and cleaning:** Data is transformed and cleaned in real-time using Azure Databricks, ensuring it is ready for immediate analysis.
  - o **Azure SQL Database stores processed real-time data for reporting:** Processed data is stored in Azure SQL Database, making it available for real-time reporting and analytics.
- **Triggering Events:**
  - o **Data Arrival:** New JSON transaction added, process and store data: Each new transaction triggers the data processing pipeline, ensuring up-to-date data is always available.
  - o **Fraud Alert:** High-value anomaly detected, trigger alert: Anomalies in transaction data trigger fraud alerts, enabling immediate action to be taken.

## 7. Solution Design & Integration

- **Complete ETL Process:**
  - o **ETL Processing:** Transform data using Azure Databricks: Data is extracted, transformed, and loaded using Azure Databricks, ensuring it is clean and ready for analysis.
  - o **Data Storage:** Store structured data in Azure SQL DB: Structured data is stored in Azure SQL Database, providing a reliable and scalable storage solution.
  - o **Real-Time Stream:** Ingest live transactions using Azure Event Hub: Live transactions are ingested in real-time, ensuring the data pipeline remains current.

## **8. Implementation and Results**

**Workflow Process:** Data moves through the pipeline in a structured manner, from ingestion to processing to storage. A flowchart was created to illustrate this process, highlighting key steps and interactions.

## **9. Conclusion and Future Work**

- **Key Findings:**
  - **Optimized Data Pipeline:** Faster query performance: The optimized data pipeline resulted in significantly faster query performance, enhancing overall efficiency.
  - **Accurate Fraud Detection:** Improved security measures: Enhanced fraud detection mechanisms improved security, reducing the risk of fraudulent activities.
- **Future Enhancements:**
  - **Integrate machine learning for predictive fraud detection:** Machine learning models will be integrated to predict and prevent fraud more effectively.
  - **Optimize costs by shifting cold data to lower-cost storage tiers:** Cost optimization strategies will be implemented by moving infrequently accessed data to lower-cost storage options.
  - **Enhance security by implementing role-based access control (RBAC):** Security will be further enhanced by implementing RBAC, ensuring only authorized users have access to sensitive data.

## **10. Final Presentation**

### **Power BI Dashboard Insights:**

- **Customer Trends:** Shows top spending users: Visualizations highlight the top spending customers, providing insights into customer behavior and preferences.
- **Service Downtime:** Identifies critical failures: Dashboards identify periods of service downtime, helping to pinpoint and address critical failures

## **11. Appendices**

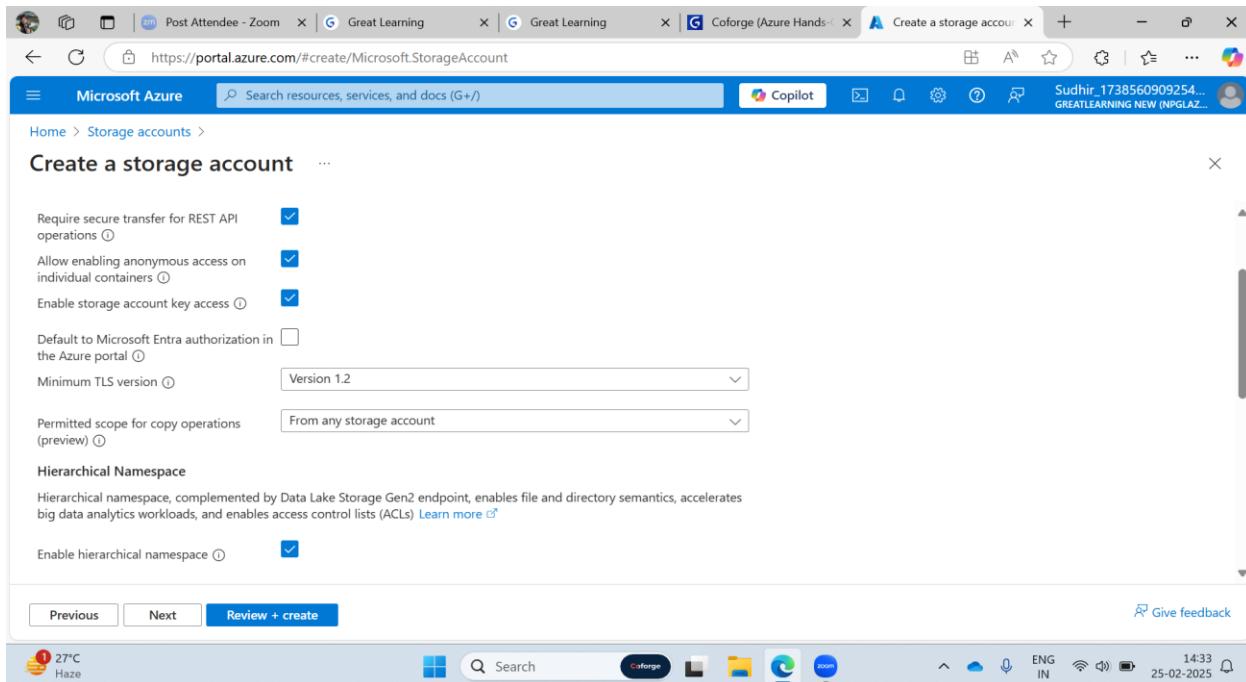
**Appendix A:** Code Snippets (Include Python/PySpark scripts used for transformation.) **Appendix B:** References (List relevant sources, documentation, or research materials.)

This report presents a comprehensive end-to-end data pipeline using Azure Data Factory, Databricks, Event Hub, SQL Database, and Power BI to process both historical and real-time BFSI data.

---

## MID-REPORT -> Screenshots of Implementation:

### STORAGE ACCOUNT



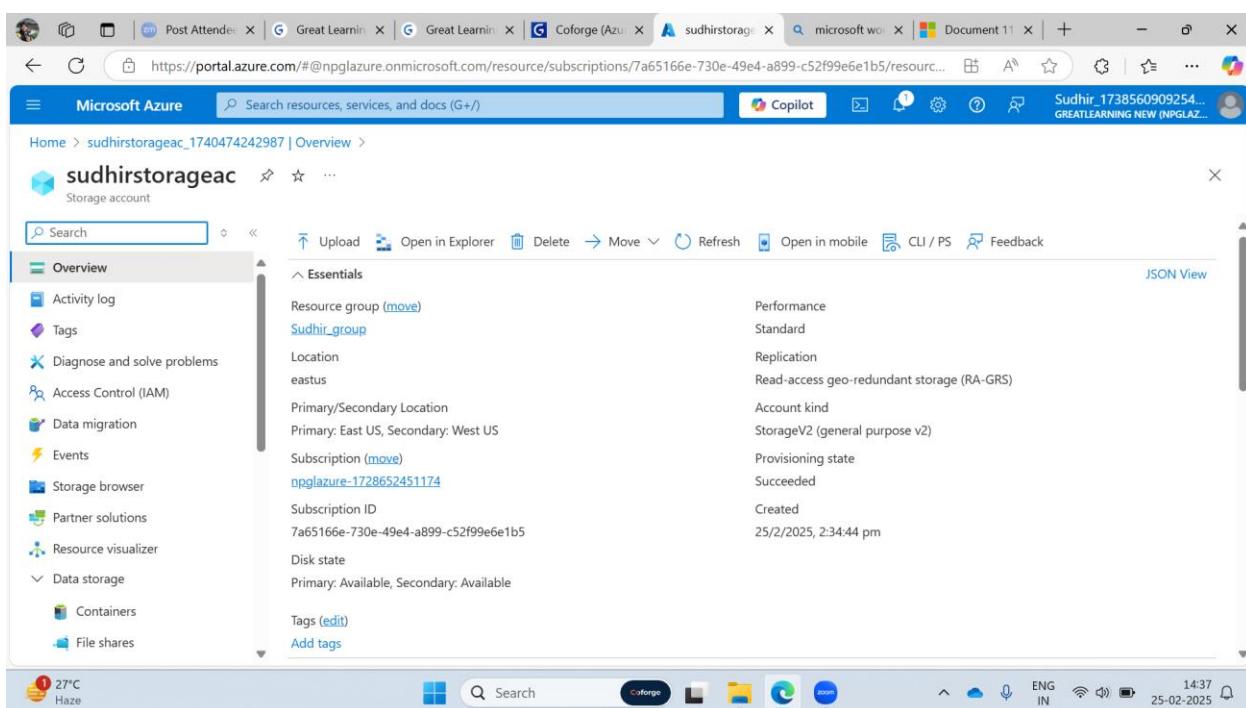
The screenshot shows the 'Create a storage account' wizard in the Microsoft Azure portal. The page title is 'Create a storage account'. There are several configuration options with checkboxes:

- Require secure transfer for REST API operations (checked)
- Allow enabling anonymous access on individual containers (checked)
- Enable storage account key access (checked)

Below these, there is a checkbox for 'Default to Microsoft Entra authorization in the Azure portal' which is unchecked. Under 'Minimum TLS version', the dropdown is set to 'Version 1.2'. Under 'Permitted scope for copy operations (preview)', the dropdown is set to 'From any storage account'.

In the 'Hierarchical Namespace' section, it is mentioned that it is complemented by Data Lake Storage Gen2 endpoint, enables file and directory semantics, accelerates big data analytics workloads, and enables access control lists (ACLs). A link to 'Learn more' is provided. The checkbox for 'Enable hierarchical namespace' is checked.

At the bottom, there are 'Previous' and 'Next' buttons, and a prominent 'Review + create' button.



The screenshot shows the 'Overview' page for the storage account 'sudhirstorageac'. The page title is 'sudhirstorageac | Overview'. The left sidebar has a 'Overview' section with links to Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Partner solutions, Resource visualizer, and Data storage. Under 'Data storage', there are links to Containers and File shares.

The main content area displays the following details:

| Essentials                               |  |
|--|--|
| Resource group (move)                    | Performance                                |
| Sudhir_group                             | Standard                                   |
| Location                                 | Replication                                |
| eastus                                   | Read-access geo-redundant storage (RA-GRS) |
| Primary/Secondary Location               | Account kind                               |
| Primary: East US, Secondary: West US     | StorageV2 (general purpose v2)             |
| Subscription (move)                      | Provisioning state                         |
| npglazure-1728652451174                  | Succeeded                                  |
| Subscription ID                          | Created                                    |
| 7a65166e-730e-49e4-a899-c52f99e6e1b5     | 25/2/2025, 2:34:44 pm                      |
| Disk state                               |  |
| Primary: Available, Secondary: Available |  |
| Tags (edit)                              |  |
| Add tags                                 |  |

## CONTAINER CREATION:

The screenshot shows the Microsoft Azure Storage account overview for 'sudhirstorageac'. On the left, a sidebar menu includes 'Containers' (selected), 'File shares', 'Queues', 'Tables', 'Networking', and 'Access keys'. The main area displays a table of existing containers with columns 'Name' (e.g., '\$logs'), 'Last modified' (e.g., 2/25/2025, 2:35:08 PM), and 'Anonymous access level' (e.g., Container (anonymous read access for containers and blobs)). A 'New container' dialog box is open on the right, prompting for a name ('sudhir-container') and anonymous access level ('Container (anonymous read access for containers and blobs)'). A warning message states: 'All container and blob data can be read by anonymous request. Clients can enumerate blobs within the container by anonymous request, but cannot enumerate containers within the storage account. Anonymous access bypasses Access Control List (ACL) settings.' A 'Create' button is at the bottom of the dialog.

The screenshot shows the Microsoft Azure Storage account overview for 'sudhirstorageac'. The 'Containers' section now lists two containers: '\$logs' and 'sudhir-container'. The 'sudhir-container' row includes columns for 'Name' (sudhir-container), 'Last modified' (2/25/2025, 2:38:11 PM), 'Anonymous access level' (Container), and 'Lease state' (Available). The rest of the interface is identical to the previous screenshot, including the sidebar and system tray.

## UPLOADING DATA IN CONTAINER:

The screenshot shows the Microsoft Azure Storage Container Overview page for the 'sudhir-container' blob container. The container has five blobs listed:

| Name                     | Modified              | Access tier    | Archive status | Blob type  | Size |
|--------------------------|-----------------------|----------------|----------------|------------|------|
| compliance_data.csv      | 2/25/2025, 2:39:18 PM | Hot (Inferred) |                | Block blob | 4.56 |
| customer_data.csv        | 2/25/2025, 2:39:18 PM | Hot (Inferred) |                | Block blob | 7.21 |
| fraud_detection_data.csv | 2/25/2025, 2:39:18 PM | Hot (Inferred) |                | Block blob | 1.25 |
| merchant_data.csv        | 2/25/2025, 2:39:18 PM | Hot (Inferred) |                | Block blob | 4.05 |
| transactional_data.csv   | 2/25/2025, 2:39:18 PM | Hot (Inferred) |                | Block blob | 5.65 |

## ADF CREATION

The screenshot shows the Microsoft Data Factory Overview page for the 'Microsoft.DataFactory-20250225144019' deployment. The deployment is marked as complete with a green checkmark icon.

**Deployment details:**

- Deployment name: Microsoft.DataFactory-202502...
- Start time: 2/25/2025, 2:41:12 PM
- Subscription: npglazure-1728652451174
- Correlation ID: 0f431550-b2f1-4e56-bd56-048...
- Resource group: Sudhir\_group

**Next steps:**

- Go to resource

**Feedback:**

Tell us about your experience with deployment

**Right sidebar:**

- Deployment succeeded:** Deployment 'Microsoft.DataFactory-20250225144019' to resource group 'Sudhir\_group' was successful.
- Cost management:** Get notified to stay within your budget and prevent unexpected charges on your bill. Set up cost alerts >
- Microsoft Defender for Cloud:** Secure your apps and infrastructure. Go to Microsoft Defender for Cloud >
- Free Microsoft tutorials:** Start learning today >
- Work with an expert:**

## DATABRICKS CREATION: -

The screenshot shows the Microsoft Azure portal with a deployment overview for the resource group 'Sudhir\_group'. A prominent message at the top right indicates that the deployment has succeeded. The deployment details show it was named 'Sudhir\_group\_Sudhir-DataBricks', initiated from a subscription starting at 2/25/2025, 2:44:12 PM, and was successful. The portal interface includes sections for Overview, Inputs, Outputs, and Template, along with links for Deployment details and Next steps. On the right side, there are promotional cards for Cost management, Microsoft Defender for Cloud, and Free Microsoft tutorials.

## CLUSTER CREATION

The screenshot shows the Databricks Compute cluster creation interface. The left sidebar lists various workspace sections like Workspace, Recents, Catalog, Workflows, and Compute. The Compute section is currently selected. The main panel is titled 'Sudhir's Cluster' and shows configuration options for a single-node cluster. It includes fields for Access mode (set to Single user access), Performance (Databricks runtime version set to 15.4 LTS (Scala 2.12, Spark 3.5.0), and Node type (Standard\_D3\_v2). There are also checkboxes for Photon Acceleration and terminating the cluster after 120 minutes of inactivity. At the bottom, there are 'Create compute' and 'Cancel' buttons. The status bar at the bottom shows system information like weather (27°C Haze), date (25-02-2025), and time (14:46).

The screenshot shows the Databricks Compute UI for 'Sudhir's Cluster'. The left sidebar has a 'Compute' tab selected. The main area shows the cluster configuration with a 'Standard\_D3\_v2' node type (14 GB Memory, 4 Cores) and a termination setting of 'Terminate after 120 minutes of inactivity'. There are sections for 'Tags' (no custom tags) and 'Summary' (1 Driver, 14 GB Memory, 4 Cores; Runtime 15.4.x-scala2.12; Photon Standard\_D3\_v2 1.5 DBU/h). A 'Advanced options' link is at the bottom. The browser address bar shows the URL for the Azure Databricks compute cluster.

Microsoft Azure | **databricks**

Search data, notebooks, recents, and more... CTRL + P

Sudhir-DataBricks

+ New

Compute

**Sudhir's Cluster**

Configuration Notebooks (0) Libraries Event log Spark UI Driver logs Metrics Apps Spark compute UI - Master

Standard\_D3\_v2 14 GB Memory, 4 Cores

Terminate after 120 minutes of inactivity

Tags

No custom tags

Automatically added tags

**Summary**

1 Driver 14 GB Memory, 4 Cores

Runtime 15.4.x-scala2.12

Photon Standard\_D3\_v2 1.5 DBU/h

Advanced options

27°C Haze

Search

ENG IN 25-02-2025 14:55

## UPLOADING HISTORICAL DATA INTO ADF:

The screenshot shows the Microsoft Azure Data Factory pipeline editor. A pipeline named "pipeline1" is selected. The pipeline consists of three main activities: "Get Metadata1" (under "Move and transform" category), "ForEach1" (under "Activities" category), and "Notebook1". The "ForEach1" activity is expanded to show its sub-activities: "ForEach1" and "Notebook1". The pipeline status table at the bottom indicates all activities have succeeded.

| Activity      | Status    | Type         | Timestamp             | Duration | Runtime                                 |
|---------------|-----------|--------------|-----------------------|----------|---|
| Notebook1     | Succeeded | Notebook     | 2/25/2025, 3:14:51 PM | 50s      | AutoResolveIntegrationRuntime (East US) |
| Notebook1     | Succeeded | Notebook     | 2/25/2025, 3:14:51 PM | 45s      | AutoResolveIntegrationRuntime (East US) |
| ForEach1      | Succeeded | ForEach      | 2/25/2025, 3:14:50 PM | 54s      | AutoResolveIntegrationRuntime (East US) |
| Get Metadata1 | Succeeded | Get Metadata | 2/25/2025, 3:14:33 PM | 17s      | AutoResolveIntegrationRuntime (East US) |

The screenshot shows a Microsoft Azure Databricks notebook titled "Historical-Data" in Python mode. The notebook interface includes a sidebar with workspace navigation (Recents, Catalog, Workflows, Compute) and data engineering features (Machine Learning, Playground, Experiments, Features, Models, Serving). The main workspace contains a single code cell with the placeholder text "Start typing or generate with AI (Ctrl + I)...". The status bar at the bottom indicates the cluster is "Sudhir's Cluster".

## PERFORMING TRANSFORMATION ON HISTORICAL DATASET:

Microsoft Azure | databricks

Historical-Data-Nootbook Python

```
# Filter rows where Column1 is not null
transactional_df = transactional_df.filter(transactional_df['transaction_id'].isNotNull())

# Display the transformed DataFrame
display(transactional_df)
```

(1) Spark Jobs

transactional\_df: pyspark.sql.dataframe.DataFrame = [transaction\_id: string, account\_id: string ... 4 more fields]

Table

| transaction_id | account_id | transaction_date | transaction_type     | transaction_amount | merch   |
|----------------|------------|------------------|----------------------|--------------------|---------|
| TXN10000       | ACC1006    | 2023-10-08       | Subscription Payment | 25000              | MER2030 |
| TXN10001       | ACC1002    | 2023-03-07       | Withdrawal           | 30000              | MER2007 |
| TXN10002       | ACC1088    | 2022-04-11       | ATM Withdrawal       | 1000               | MER2004 |
| TXN10003       | ACC1090    | 2022-07-21       | Subscription Payment | 10000              | MER2014 |
| TXN10004       | ACC1047    | 2023-03-15       | Withdrawal           | 70000              | MER2035 |
| TXN10005       | ACC1091    | 2022-12-22       | Withdrawal           | 5000               | MER2073 |
| TXN10006       | ACC1089    | 2023-12-27       | Subscription Payment | 5000               | MER2061 |

Microsoft Azure | databricks

Historical-Data-Nootbook Python

```
#Total number of transaction per day
from pyspark.sql.functions import count

daily_txn_vol = transactional_df.groupBy("transaction_date").agg(count("*").alias("txn_count"))
print(daily_txn_vol.count())
```

(3) Spark Jobs

daily\_txn\_vol: pyspark.sql.dataframe.DataFrame = [transaction\_date: date, txn\_count: long]

86

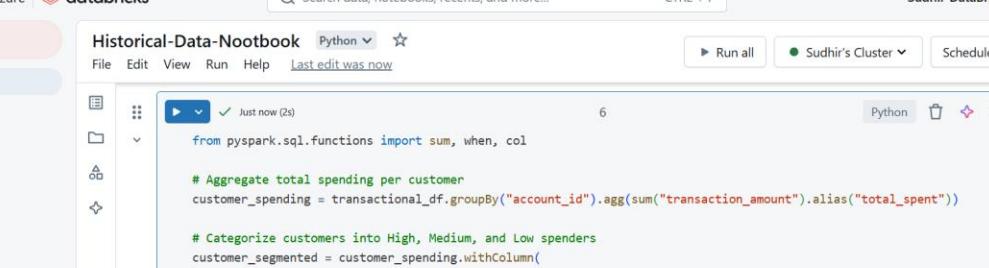
Just now (1s)

```
# Number of transaction of each type
txn_type_distribution = transactional_df.groupBy("transaction_type").agg(count("*").alias("txn_count"))
print(txn_type_distribution.count())
```

(3) Spark Jobs

txn\_type\_distribution: pyspark.sql.dataframe.DataFrame = [transaction\_type: string, txn\_count: long]

9



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

# Aggregate total spending per customer
customer_spending = transactional_df.groupBy("account_id").agg(sum("transaction_amount").alias("total_spent"))

# Categorize customers into High, Medium, and Low spenders
customer_segmented = customer_spending.withColumn(
    "customer_segment",
    when(col("total_spent") >= 100000, "High")
    .when(col("total_spent").between(50000, 99999), "Medium")
    .otherwise("Low")
)

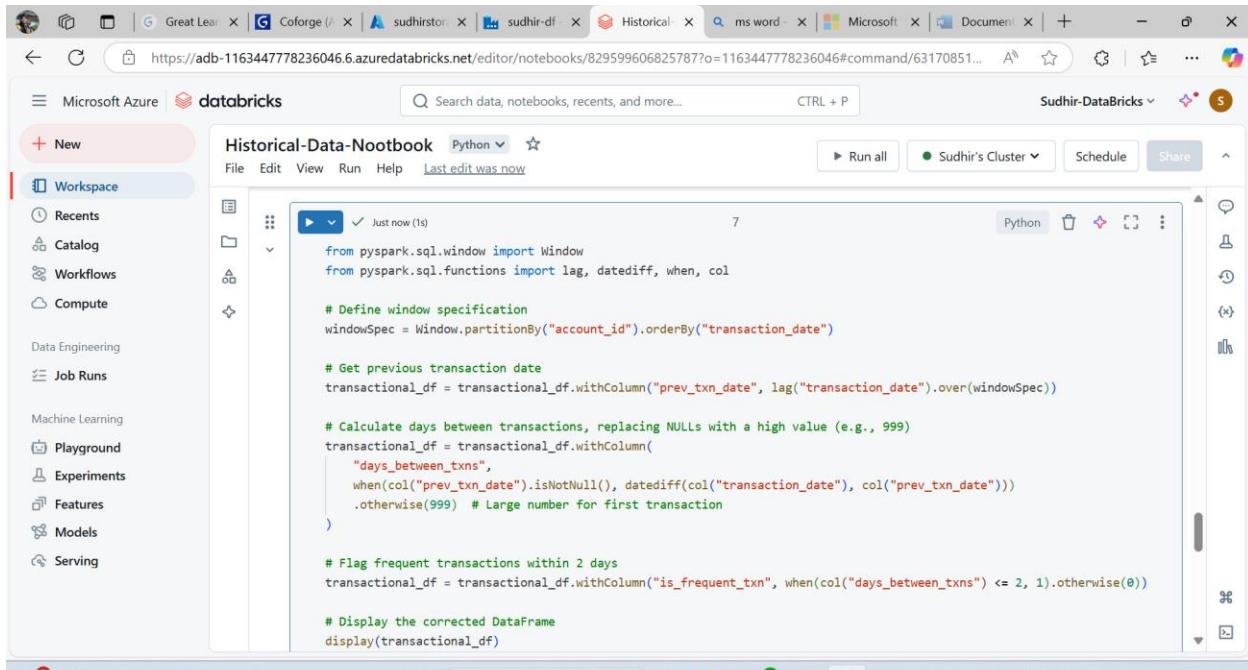
# Join the segmented data back to the original transaction DataFrame
transactional_df_enriched = transactional_df.join(customer_segmented, on="account_id", how="left")

transactional_df_enriched.display()

# (3) Spark Jobs
customer_segmented: pyspark.sql.dataframe.DataFrame = [account_id: string, total_spent: long ... 1 more field]
customer_spending: pyspark.sql.dataframe.DataFrame = [account_id: string, total_spent: long]
```

The screenshot shows the Microsoft Azure Databricks workspace. On the left, a sidebar lists various notebooks and workspace categories like Workspace, Recents, Catalog, Workflows, Compute, Data Engineering, Job Runs, Machine Learning, Playground, Experiments, Features, Models, and Serving. The main area displays a notebook titled "Historical-Data-Nootbook" in Python. The code in the notebook attempts to join three dataframes: customer\_segmented, customer\_spending, and transactional\_df\_enriched. A preview of the transactional DataFrame is shown below, containing columns such as account\_id, transaction\_date, transaction\_type, transaction\_amount, and merchant\_id.

| account_id | transaction_date | transaction_type     | transaction_amount | merchant_id |
|------------|------------------|----------------------|--------------------|-------------|
| ACC1006    | 2023-10-08       | Subscription Payment | 25000              | MER2030     |
| ACC1002    | 2023-03-07       | Withdrawal           | 30000              | MER2007     |
| ACC1088    | 2022-04-11       | ATM Withdrawal       | 1000               | MER2004     |
| ACC1090    | 2022-07-21       | Subscription Payment | 10000              | MER2014     |
| ACC1047    | 2023-03-15       | Withdrawal           | 70000              | MER2035     |



```
from pyspark.sql.window import Window
from pyspark.sql.functions import lag, datediff, when, col

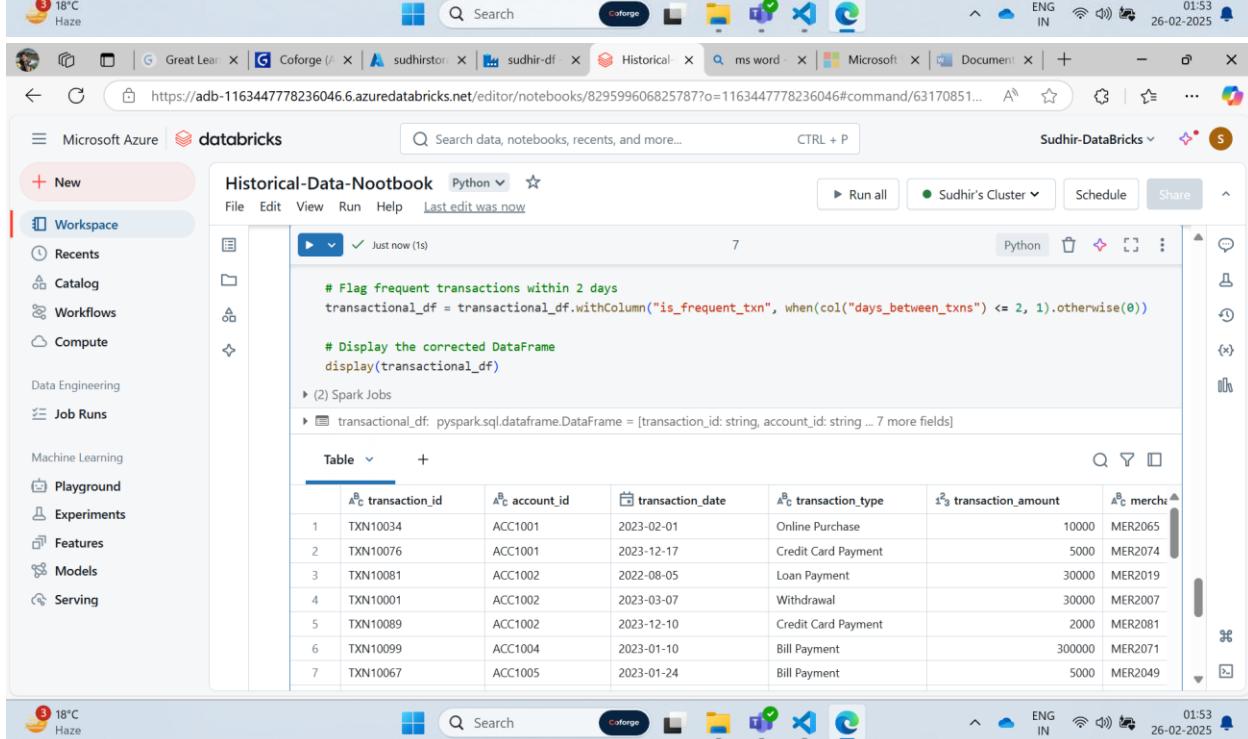
# Define window specification
windowSpec = Window.partitionBy("account_id").orderBy("transaction_date")

# Get previous transaction date
transactional_df = transactional_df.withColumn("prev_txn_date", lag("transaction_date").over(windowSpec))

# Calculate days between transactions, replacing NULLs with a high value (e.g., 999)
transactional_df = transactional_df.withColumn(
    "days_between_txns",
    when(col("prev_txn_date").isNotNull(), datediff(col("transaction_date"), col("prev_txn_date")))
    .otherwise(999) # Large number for first transaction
)

# Flag frequent transactions within 2 days
transactional_df = transactional_df.withColumn("is_frequent_txn", when(col("days_between_txns") <= 2, 1).otherwise(0))

# Display the corrected DataFrame
display(transactional_df)
```



|   | transaction_id | account_id | transaction_date | transaction_type    | transaction_amount | merchant_id |
|---|----------------|------------|------------------|---------------------|--------------------|-------------|
| 1 | TXN10034       | ACC1001    | 2023-02-01       | Online Purchase     | 10000              | MER2065     |
| 2 | TXN10076       | ACC1001    | 2023-12-17       | Credit Card Payment | 5000               | MER2074     |
| 3 | TXN10081       | ACC1002    | 2022-08-05       | Loan Payment        | 30000              | MER2019     |
| 4 | TXN10001       | ACC1002    | 2023-03-07       | Withdrawal          | 30000              | MER2007     |
| 5 | TXN10089       | ACC1002    | 2023-12-10       | Credit Card Payment | 2000               | MER2081     |
| 6 | TXN10099       | ACC1004    | 2023-01-10       | Bill Payment        | 300000             | MER2071     |
| 7 | TXN10067       | ACC1005    | 2023-01-24       | Bill Payment        | 5000               | MER2049     |

Microsoft Azure | **Historical-Databricks** Python ⚡

File Edit View Run Help Last edit was now

Just now (1s)

```
column_names = ["account_id", "first_name", "last_name", "signup_date", "account_type", "email"]
customer_df = spark.read.csv("./dbfs/tmp/customer_data.csv", header=False, inferSchema=True).toDF(*column_names)
display(customer_df)
```

(3) Spark Jobs

customer\_df: pyspark.sql.dataframe.DataFrame = [account\_id: string, first\_name: string ... 4 more fields]

Table +

|   | account_id | first_name  | last_name  | signup_date | account_type      | email                         |
|---|------------|-------------|------------|-------------|-------------------|-------------------------------|
| 1 | ACC1000    | Nimrat      | Patil      | 2022-01-01  | Current Account   | nimrat.patil@gmail.com        |
| 2 | ACC1001    | Ekaraj      | Gara       | 2022-05-31  | Savings Account   | ekaraj.gara@yahoo.com         |
| 3 | ACC1002    | Ekiya       | Contractor | 2022-01-01  | Loan Account      | ekiya.contractor@hotmail.com  |
| 4 | ACC1003    | Christopher | Patil      | 2022-03-02  | Loan Account      | christopher.patil@outlook.com |
| 5 | ACC1004    | Jyoti       | Issac      | 2022-08-29  | Recurring Deposit | jyoti.issac@hotmail.com       |
| 6 | ACC1005    | Devansh     | Soman      | 2022-04-01  | Recurring Deposit | devansh.soman@outlook.com     |
| 7 | ACC1006    | Krishna     | Gera       | 2022-03-02  | Savings Account   | krishna.gera@hotmail.com      |
| 8 | ACC1007    | Zilmil      | Sagar      | 2022-05-01  | Fixed Deposit     | zilmil.sagar@outlook.com      |
| 9 | ACC1008    | Vritti      | Swamy      | 2022-05-01  | Recurring Deposit | vritti.swamy@outlook.com      |

18°C Haze

Search CoForge

Run all Sudhir's Cluster Schedule Share

Python

01:54 26-02-2025

Microsoft Azure | **Historical-Databricks** Python ⚡

File Edit View Run Help Last edit was now

Just now (<1s)

```
# Filter rows where Column1 is not null
customer_df = customer_df.filter(customer_df.account_id.isNotNull())

# Display the transformed DataFrame
display(customer_df)
```

(1) Spark Jobs

customer\_df: pyspark.sql.dataframe.DataFrame = [account\_id: string, first\_name: string ... 4 more fields]

Table +

|   | account_id | first_name  | last_name  | signup_date | account_type      | email                         |
|---|------------|-------------|------------|-------------|-------------------|-------------------------------|
| 1 | ACC1000    | Nimrat      | Patil      | 2022-01-01  | Current Account   | nimrat.patil@gmail.com        |
| 2 | ACC1001    | Ekaraj      | Gara       | 2022-05-31  | Savings Account   | ekaraj.gara@yahoo.com         |
| 3 | ACC1002    | Ekiya       | Contractor | 2022-01-01  | Loan Account      | ekiya.contractor@hotmail.com  |
| 4 | ACC1003    | Christopher | Patil      | 2022-03-02  | Loan Account      | christopher.patil@outlook.com |
| 5 | ACC1004    | Jyoti       | Issac      | 2022-08-29  | Recurring Deposit | jyoti.issac@hotmail.com       |
| 6 | ACC1005    | Devansh     | Soman      | 2022-04-01  | Recurring Deposit | devansh.soman@outlook.com     |
| 7 | ACC1006    | Krishna     | Gera       | 2022-03-02  | Savings Account   | krishna.gera@hotmail.com      |

18°C Haze

Search CoForge

Run all Sudhir's Cluster Schedule Share

Python

01:55 26-02-2025

Microsoft Azure | **Historical-Databricks** Python Search data, notebooks, recents, and more... CTRL + P Sudhir-DataBrics + - X

File Edit View Run Help Last edit was now

Just now (1s) 10 Python

```
column_names = ["merchant_id", "merchant_name", "category", "risk_rating"]
merchant_df = spark.read.csv("/dbfs/tmp/merchant_data.csv", header=False, inferSchema=True).toDF(*column_names)
display(merchant_df)
```

(3) Spark Jobs

merchant\_df: pyspark.sql.dataframe.DataFrame = [merchant\_id: string, merchant\_name: string ... 2 more fields]

Table +

|   | A <sub>c</sub> merchant_id | A <sub>c</sub> merchant_name | A <sub>c</sub> category | 1.2 risk_rating |
|---|----------------------------|------------------------------|-------------------------|-----------------|
| 1 | MER2000                    | Sandal PLC                   | E-commerce              | 0.82            |
| 2 | MER2001                    | Kanda, Chaudry and Mammen    | Entertainment           | 0.85            |
| 3 | MER2002                    | Gupta, Grover and Bora       | Pharmacy                | 0.81            |
| 4 | MER2003                    | Barman, Lad and Palan        | E-commerce              | 0.75            |
| 5 | MER2004                    | Sagar, Basu and Kanda        | E-commerce              | 0.36            |
| 6 | MER2005                    | Kar Ltd                      | Automotive              | 0.96            |
| 7 | MER2006                    | Prasad, Buch and Natt        | Retail                  | 0.99            |
| 8 | MER2007                    | Seshadri-Pai                 | Transport               | 0.88            |
| 9 | MER2008                    | Hora Ltd                     | Healthcare              | 0.26            |

18°C Haze 01:55 26-02-2025

Microsoft Azure | **Historical-Databricks** Python Search data, notebooks, recents, and more... CTRL + P Sudhir-DataBrics + - X

File Edit View Run Help Last edit was now

Just now (<1s) 11 Python

```
# Filter rows where Column1 is not null
merchant_df = merchant_df.filter(merchant_df['merchant_id'].isNotNull())

# Display the transformed DataFrame
display(merchant_df)
```

(1) Spark Jobs

merchant\_df: pyspark.sql.dataframe.DataFrame = [merchant\_id: string, merchant\_name: string ... 2 more fields]

Table +

|   | A <sub>c</sub> merchant_id | A <sub>c</sub> merchant_name | A <sub>c</sub> category | 1.2 risk_rating |
|---|----------------------------|------------------------------|-------------------------|-----------------|
| 1 | MER2000                    | Sandal PLC                   | E-commerce              | 0.82            |
| 2 | MER2001                    | Kanda, Chaudry and Mammen    | Entertainment           | 0.85            |
| 3 | MER2002                    | Gupta, Grover and Bora       | Pharmacy                | 0.81            |
| 4 | MER2003                    | Barman, Lad and Palan        | E-commerce              | 0.75            |
| 5 | MER2004                    | Sagar, Basu and Kanda        | E-commerce              | 0.36            |
| 6 | MER2005                    | Kar Ltd                      | Automotive              | 0.96            |

18°C Haze 01:55 26-02-2025

Microsoft Azure | databricks

Historical-Data-Nootbook Python

File Edit View Run Help Last edit was now

Run all Sudhir's Cluster Schedule Share

14 MER2013 Agate and Sons Retail 0.69  
15 MER2014 Sarma-Sandhu Transport 0.52  
100 rows | 0.33s runtime Refreshed now

Just now (1s) 12

```
from pyspark.sql.functions import col  
  
# Drop rows if merchant_id, merchant_name, or category is NULL  
merchant_df = merchant_df.dropna(subset=["merchant_id", "merchant_name", "category"])  
  
# Fill missing risk ratings with the average risk rating  
avg_risk = merchant_df.selectExpr("avg(risk_rating)").collect()[0][0]  
merchant_df = merchant_df.fillna({"risk_rating": avg_risk})
```

(2) Spark Jobs

merchant\_df: pyspark.sql.dataframe.DataFrame

```
merchant_id: string  
merchant_name: string  
category: string  
risk_rating: double
```

Microsoft Azure | databricks

Historical-Data-Nootbook Python

File Edit View Run Help Last edit was now

Run all Sudhir's Cluster Schedule Share

Just now (<1s) 13

```
from pyspark.sql.functions import when  
  
merchant_df = merchant_df.withColumn("risk_category", when(col("risk_rating") >= 0.8, "High").when(col("risk_rating") >= 0.5, "Medium").otherwise("Low"))  
merchant_df.display()
```

(1) Spark Jobs

merchant\_df: pyspark.sql.dataframe.DataFrame = [merchant\_id: string, merchant\_name: string ... 3 more fields]

Table +

|   | merchant_id | merchant_name             | category      | risk_rating | risk_category |
|---|-------------|---------------------------|---------------|-------------|---------------|
| 1 | MER2000     | Sandal PLC                | E-commerce    | 0.82        | High          |
| 2 | MER2001     | Kanda, Chaudry and Mammen | Entertainment | 0.85        | High          |
| 3 | MER2002     | Gupta, Grover and Bora    | Pharmacy      | 0.81        | High          |
| 4 | MER2003     | Barman, Lad and Palan     | E-commerce    | 0.75        | Medium        |
| 5 | MER2004     | Sagar, Basu and Kanda     | E-commerce    | 0.36        | Low           |
| 6 | MER2005     | Kar Ltd                   | Automotive    | 0.96        | High          |
| 7 | MER2006     | Prasad, Buch and Natt     | Retail        | 0.99        | High          |

Microsoft Azure | **Historical-Databricks** Python Search data, notebooks, recents, and more... CTRL + P Sudhir's Cluster Schedule Share

+ New Workspace Recents Catalog Workflows Compute Data Engineering Job Runs Machine Learning Playground Experiments Features Models Serving

Historical-Databricks Python Just now (1s) 14

```
column_names = ["compliance_id", "transaction_id", "compliance_check", "status"]
compliance_df = spark.read.csv("/dbfs/tmp/compliance_data.csv", header=False, inferSchema=True).toDF(*column_names)
display(compliance_df)
```

(3) Spark Jobs

compliance\_df: pyspark.sql.dataframe.DataFrame = [compliance\_id: string, transaction\_id: string ... 2 more fields]

Table +

| A <sup>B</sup> compliance_id | A <sup>B</sup> transaction_id | A <sup>B</sup> compliance_check | A <sup>B</sup> status |
|------------------------------|-------------------------------|---------------------------------|-----------------------|
| 1 COMP7000                   | TXN10031                      | No Issues                       | Failed                |
| 2 COMP7001                   | TXN10042                      | Sanction List Matched           | Failed                |
| 3 COMP7002                   | TXN10067                      | High-Risk Country Alert         | Failed                |
| 4 COMP7003                   | TXN10030                      | Mismatched Beneficiary Info     | Passed                |
| 5 COMP7004                   | TXN10029                      | Multiple Failed Logins          | Failed                |
| 6 COMP7005                   | TXN10025                      | Unusual Transaction Pattern     | Failed                |
| 7 COMP7006                   | TXN10093                      | Unusual Transaction Pattern     | Failed                |
| 8 COMP7007                   | TXN10033                      | Mismatched Beneficiary Info     | Failed                |
| 9 COMP7008                   | TXN10070                      | Unusual Transaction Pattern     | Passed                |

Microsoft Azure | **Historical-Databricks** Python Search data, notebooks, recents, and more... CTRL + P Sudhir's Cluster Schedule Share

+ New Workspace Recents Catalog Workflows Compute Data Engineering Job Runs Machine Learning Playground Experiments Features Models Serving

Historical-Databricks Python Just now (1s) 16

```
column_names = ["fraud_id", "transaction_id", "account_id", "fraud_score", "detection_date"]
fraud_detection_df = spark.read.csv("/dbfs/tmp/fraud_detection_data.csv", header=False, inferSchema=True).toDF(*column_names)
display(fraud_detection_df)
```

(3) Spark Jobs

fraud\_detection\_df: pyspark.sql.dataframe.DataFrame = [fraud\_id: string, transaction\_id: string ... 3 more fields]

Table +

| A <sup>B</sup> fraud_id | A <sup>B</sup> transaction_id | A <sup>B</sup> account_id | 1.2 fraud_score | detection_date |
|-------------------------|-------------------------------|---------------------------|-----------------|----------------|
| 1 FRD5000               | TXN10058                      | ACC1093                   | 0.99            | 2024-03-30     |
| 2 FRD5001               | TXN10063                      | ACC1031                   | 0.34            | 2024-02-13     |
| 3 FRD5002               | TXN10043                      | ACC1037                   | 0.66            | 2024-02-17     |
| 4 FRD5003               | TXN10077                      | ACC1043                   | 0.55            | 2024-02-04     |
| 5 FRD5004               | TXN10079                      | ACC1017                   | 0.32            | 2024-02-16     |
| 6 FRD5005               | TXN10008                      | ACC1053                   | 0.93            | 2024-02-08     |
| 7 FRD5006               | TXN10002                      | ACC1088                   | 0.32            | 2024-03-29     |
| 8 FRD5007               | TXN10051                      | ACC1060                   | 0.95            | 2024-02-02     |

Microsoft Azure | **Historical-Databricks** Python Search data, notebooks, recents, and more... CTRL + P Sudhir-Databricks + - X

File Edit View Run Help Last edit was now

Run all Sudhir's Cluster Schedule Share

+ New Workspace Recents Catalog Workflows Compute Data Engineering Job Runs Machine Learning Playground Experiments Features Models Serving

Historical-Databricks Notebook Python Just now (<1s)

```
from pyspark.sql.functions import lit
# Filter rows where Column1 is not null
compliance_df = compliance_df.filter(compliance_df.compliance_id.isNotNull())

# Add a new column with a constant value
compliance_df = compliance_df.withColumn("NewColumn", lit("ConstantValue"))

# Display the transformed DataFrame
display(compliance_df)
```

(1) Spark Jobs

compliance\_df: pyspark.sql.dataframe.DataFrame = [compliance\_id: string, transaction\_id: string ... 3 more fields]

Table +

|   | A <sup>B</sup> compliance_id | A <sup>B</sup> transaction_id | A <sup>B</sup> compliance_check | A <sup>B</sup> status | A <sup>B</sup> NewColumn |
|---|------------------------------|-------------------------------|---------------------------------|-----------------------|--------------------------|
| 1 | COMP7000                     | TXN10031                      | No Issues                       | Failed                | ConstantValue            |
| 2 | COMP7001                     | TXN10042                      | Sanction List Matched           | Failed                | ConstantValue            |
| 3 | COMP7002                     | TXN10067                      | High-Risk Country Alert         | Failed                | ConstantValue            |
| 4 | COMP7003                     | TXN10030                      | Mismatched Beneficiary Info     | Passed                | ConstantValue            |

Microsoft Azure | **Historical-Databricks** Python Search data, notebooks, recents, and more... CTRL + P Sudhir-Databricks + - X

File Edit View Run Help Last edit was now

Run all Sudhir's Cluster Schedule Share

+ New Workspace Recents Catalog Workflows Compute Data Engineering Job Runs Machine Learning Playground Experiments Features Models Serving

Historical-Databricks Notebook Python Just now (<1s)

```
# Filter rows where Column1 is not null
fraud_detection_df = fraud_detection_df.filter(fraud_detection_df.account_id.isNotNull())

# Display the transformed DataFrame
display(fraud_detection_df)
```

(1) Spark Jobs

fraud\_detection\_df: pyspark.sql.dataframe.DataFrame = [fraud\_id: string, transaction\_id: string ... 3 more fields]

Table +

|   | A <sup>B</sup> fraud_id | A <sup>B</sup> transaction_id | A <sup>B</sup> account_id | 1.2 fraud_score | detection_date |
|---|-------------------------|-------------------------------|---------------------------|-----------------|----------------|
| 1 | FRD5000                 | TXN10058                      | ACC1093                   | 0.99            | 2024-03-30     |
| 2 | FRD5001                 | TXN10063                      | ACC1031                   | 0.34            | 2024-02-13     |
| 3 | FRD5002                 | TXN10043                      | ACC1037                   | 0.66            | 2024-02-17     |
| 4 | FRD5003                 | TXN10077                      | ACC1043                   | 0.55            | 2024-02-04     |
| 5 | FRD5004                 | TXN10079                      | ACC1017                   | 0.32            | 2024-02-16     |
| 6 | FRD5005                 | TXN10008                      | ACC1053                   | 0.93            | 2024-02-08     |
| 7 | FRD5006                 | TXN10002                      | ACC1088                   | 0.32            | 2024-03-29     |

Microsoft Azure | databricks

Historical-Databricks Notebook Python

```
from pyspark.sql.functions import col, to_date, when
from pyspark.sql.types import FloatType

# Convert fraud_score to float
fraud_detection_df = fraud_detection_df.withColumn("fraud_score", col("fraud_score").cast(FloatType()))

# Categorize Fraud Risk Levels based on fraud_score
fraud_detection_df = fraud_detection_df.withColumn(
    "fraud_risk_level",
    when(col("fraud_score") < 0.3, "Low")
    .when((col("fraud_score") >= 0.3) & (col("fraud_score") < 0.7), "Medium")
    .otherwise("High")
)

# Filter out transactions with invalid or null fraud_score
fraud_detection_df = fraud_detection_df.filter(col("fraud_score").isNotNull())

# Drop duplicate fraud IDs (if any)
fraud_detection_df = fraud_detection_df.dropDuplicates(["fraud_id"])
```

Microsoft Azure | databricks

Historical-Databricks Notebook Python

```
fraud_detection_df = fraud_detection_df.filter(col("fraud_score").isNotNull())

# Drop duplicate fraud IDs (if any)
fraud_detection_df = fraud_detection_df.dropDuplicates(["fraud_id"])

# Display the transformed DataFrame
display(fraud_detection_df)
```

(2) Spark Jobs

fraud\_detection\_df: pyspark.sql.dataframe.DataFrame = [fraud\_id: string, transaction\_id: string ... 4 more fields]

| # | fraud_id | transaction_id | account_id | fraud_score        | detection_date | fraud_risk_level |
|---|----------|----------------|------------|--------------------|----------------|------------------|
| 1 | FRD5000  | TXN10058       | ACC1093    | 0.9900000095367432 | 2024-03-30     | High             |
| 2 | FRD5001  | TXN10063       | ACC1031    | 0.3400000035762787 | 2024-02-13     | Medium           |
| 3 | FRD5002  | TXN10043       | ACC1037    | 0.6600000262260437 | 2024-02-17     | Medium           |
| 4 | FRD5003  | TXN10077       | ACC1043    | 0.550000011920929  | 2024-02-04     | Medium           |
| 5 | FRD5004  | TXN10079       | ACC1017    | 0.3199999928474426 | 2024-02-16     | Medium           |
| 6 | FRD5005  | TXN10008       | ACC1053    | 0.9300000071525574 | 2024-02-08     | High             |

## DATA IN SQL DATABASE:

The screenshot shows the Microsoft Azure Query editor (preview) interface. On the left, there's a sidebar with various options like Overview, Activity log, Tags, Diagnose and solve problems, and Query editor (preview). The main area displays a table named 'transactional\_data' with columns: transaction\_id, account\_id, transaction\_date, transaction\_type, transaction\_amount, and merchant. The data shows three rows of transactions from different accounts on different dates.

| transaction_id | account_id | transaction_date | transaction_type     | transaction_amount | merchant |
|----------------|------------|------------------|----------------------|--------------------|----------|
| TXN10000       | ACC1006    | 2023-10-08       | Subscription Payment | 25000              |          |
| TXN10001       | ACC1002    | 2023-03-07       | Withdrawal           | 30000              |          |
| TXN10002       | ACC1088    | 2022-04-11       | ATM Withdrawal       | 1000               |          |
| TXN10002       | ACC1000    | 2022-07-21       | Subscription Payment | 10000              |          |

## POWER BI REPORT OF HISTORICAL DATASET:

## TRANSFORMATION IN REALTIME DATA

The screenshot shows a Databricks workspace with a notebook titled 'Real-Time-Data-Nootbook'. The notebook contains Python code for reading JSON data from a source and displaying it as a DataFrame. Below the code, a table is displayed showing three rows of transaction data with columns: transaction\_id, account\_id, transaction\_date, transaction\_type, transaction\_amount, and merchant.

| transaction_id | account_id | transaction_date    | transaction_type     | transaction_amount | merchant |
|----------------|------------|---------------------|----------------------|--------------------|----------|
| RT_TXN20000    | ACC1088    | 2025-02-01 09:19:36 | Subscription Payment | 15000              | MER2010  |
| RT_TXN20001    | ACC1074    | 2025-02-04 12:47:36 | ATM Withdrawal       | 10000              | MER2093  |
| RT_TXN20002    | ACC1080    | 2025-02-18 09:23:36 | Online Purchase      | 2000               | MER2052  |

Microsoft Azure | databricks

Real-Time-Data-Nootbook Python

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, year, month, dayofmonth
# Convert transaction_date to datetime format and extract year, month, and day
df = df.withColumn("transaction_date", col("transaction_date").cast("date"))
df = df.withColumn("year", year(col("transaction_date")))
df = df.withColumn("month", month(col("transaction_date")))
df = df.withColumn("day", dayofmonth(col("transaction_date")))

# Show the transformed DataFrame
df.display()
```

(1) Spark Jobs

display\_query\_2 (id: ce6c201d-7ae9-4bda-a7c5-74f042ff5e92) Last updated: 5 seconds ago

| transaction_id | account_id | transaction_date | transaction_type     | transaction_amount | merchant_id |
|----------------|------------|------------------|----------------------|--------------------|-------------|
| RT_TXN20000    | ACC1088    | 2025-02-01       | Subscription Payment | 15000              | MER2010     |

Microsoft Azure | databricks

Real-Time-Data-Nootbook Python

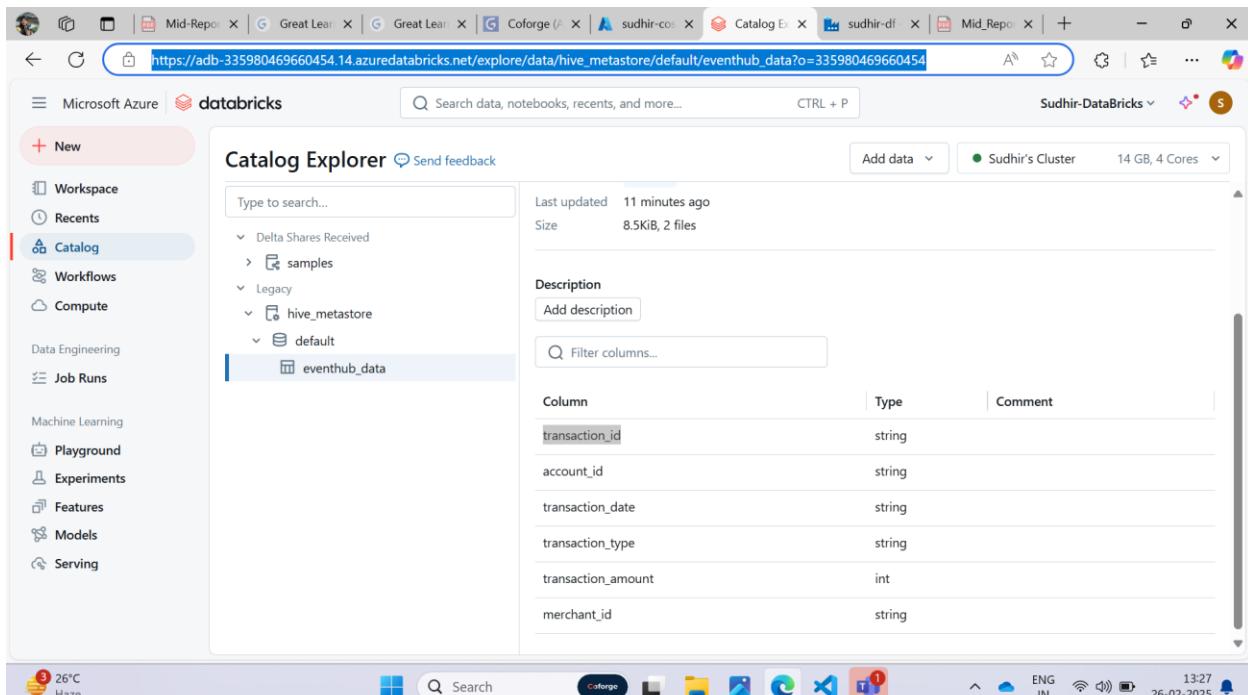
```
from pyspark.sql.functions import col
# Create a new column transaction_amount_usd by converting the transaction_amount from CAD to USD
df = df.withColumn("transaction_amount_usd", col("transaction_amount") * 0.75)
df.display()
```

(1) Spark Jobs

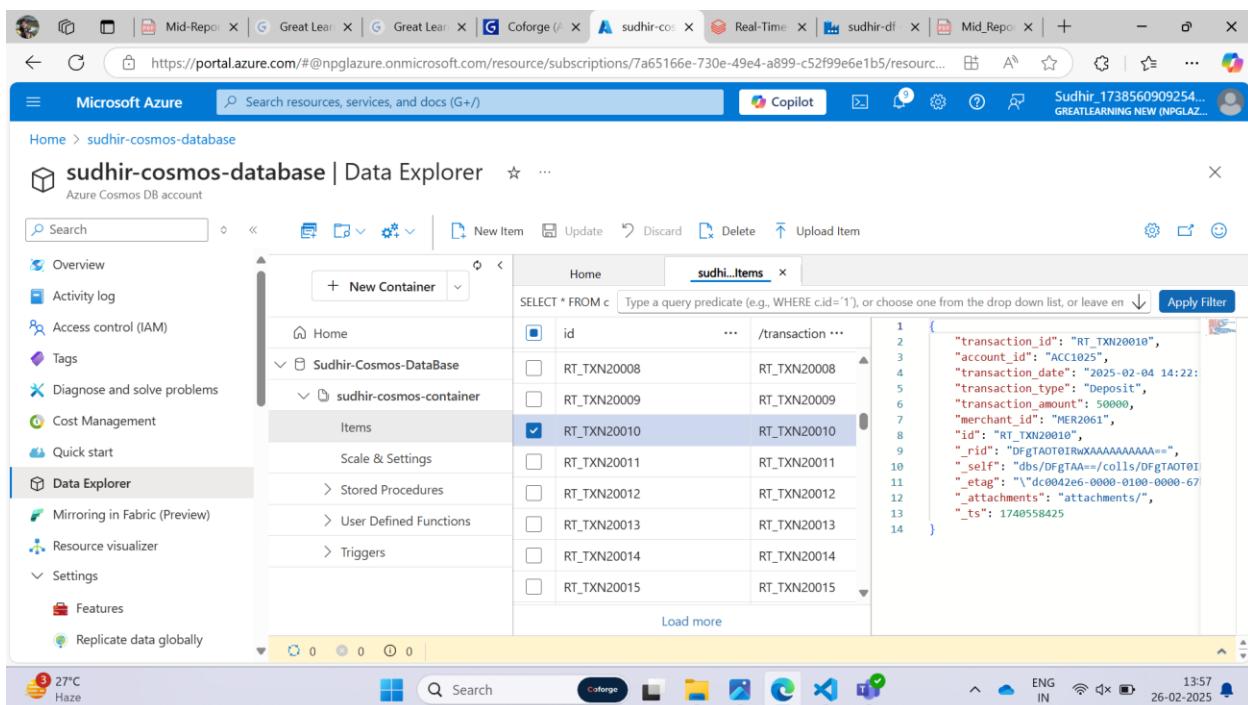
display\_query\_3 (id: e455d3cd-2ea7-4957-8eb6-f6ec82c23ce) Last updated: 5 seconds ago

| transaction_amount | merchant_id | year | month | day | transaction_amount_usd |
|--------------------|-------------|------|-------|-----|------------------------|
| 15000              | MER2010     | 2025 | 2     | 1   | 11250                  |
| 10000              | MER2093     | 2025 | 2     | 4   | 7500                   |
| 2000               | MER2052     | 2025 | 2     | 18  | 1500                   |
| 20000              | MER2036     | 2025 | 1     | 30  | 15000                  |
| 200000             | MER2060     | 2025 | 1     | 30  | 150000                 |
| 15000              | MER2029     | 2025 | 2     | 1   | 11250                  |
| 10000              | MER2001     | 2025 | 2     | 9   | 7500                   |

## STORING REALTIME DATA IN COSMOS DB:

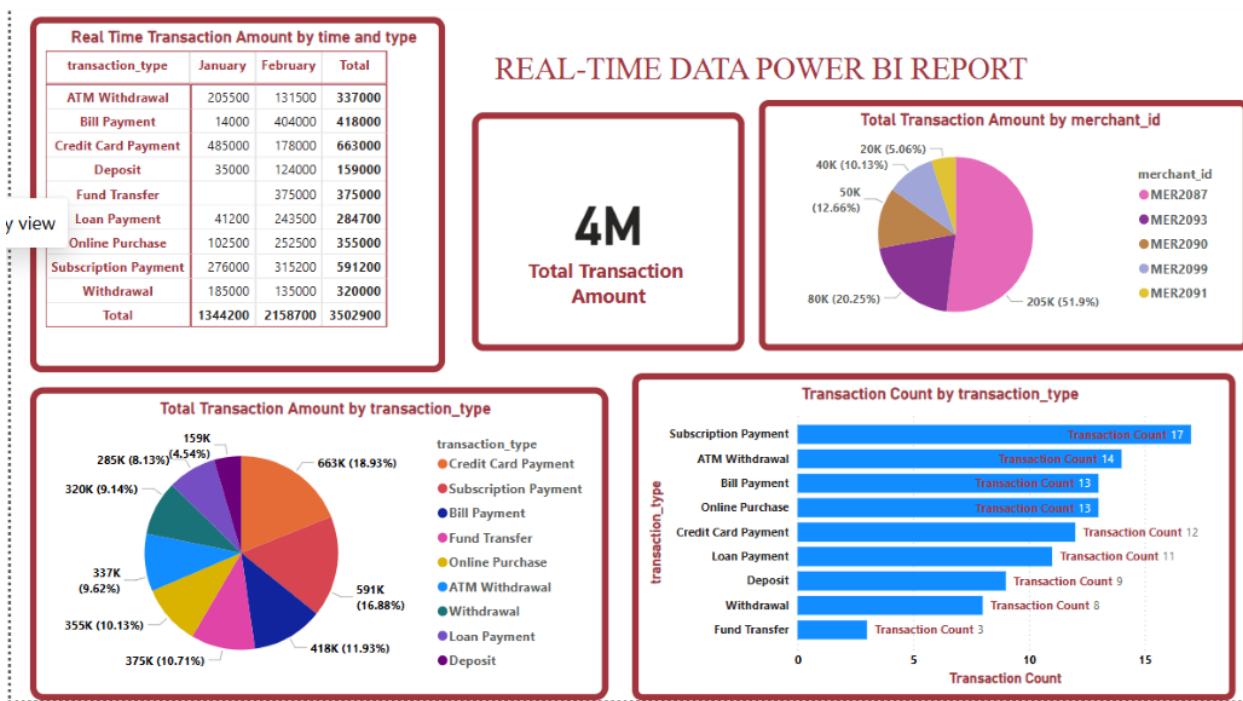
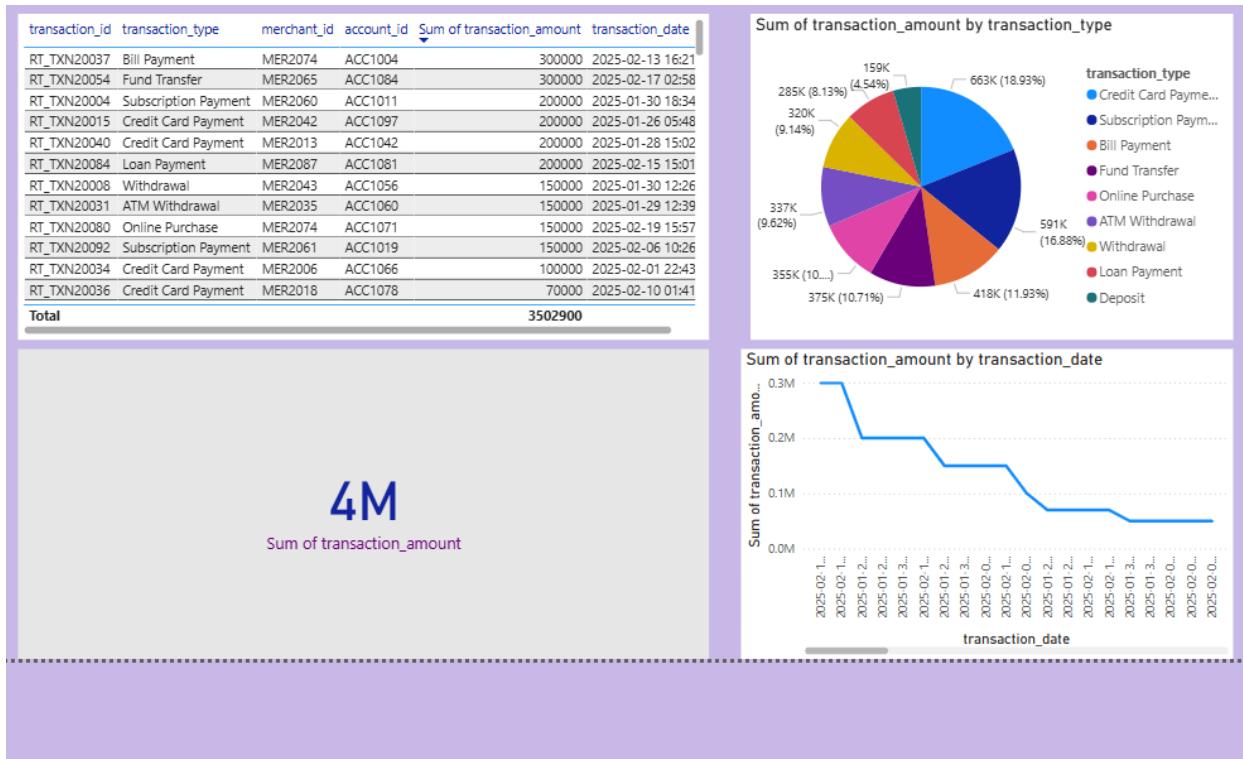


The screenshot shows the Databricks Catalog Explorer interface. On the left sidebar, under the 'Catalog' section, there is a tree view of databases and tables. Under 'default', the 'eventhub\_data' table is selected. To the right of the tree view, there is a detailed view of the table's schema, including columns like transaction\_id, account\_id, transaction\_date, transaction\_type, transaction\_amount, and merchant\_id, each with its corresponding type (string or int).

The screenshot shows the Azure Data Explorer interface for the 'sudhir-cosmos-database'. The left sidebar has a 'Data Explorer' section selected. In the main area, a table named 'sudhi\_items' is displayed with columns id, ... /transaction ... (with a dropdown menu), and RT\_TXN\*. The table contains 15 rows of transaction data. A preview pane on the right shows the JSON structure of one of the transactions, including fields like transaction\_id, account\_id, transaction\_date, transaction\_type, transaction\_amount, merchant\_id, id, \_rid, \_self, \_etag, attachments, and \_ts.

## POWER BI REPORT OF REALTIME DATA:



## POWER BI REPORT OF HISTORICAL DATA:

