# 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.

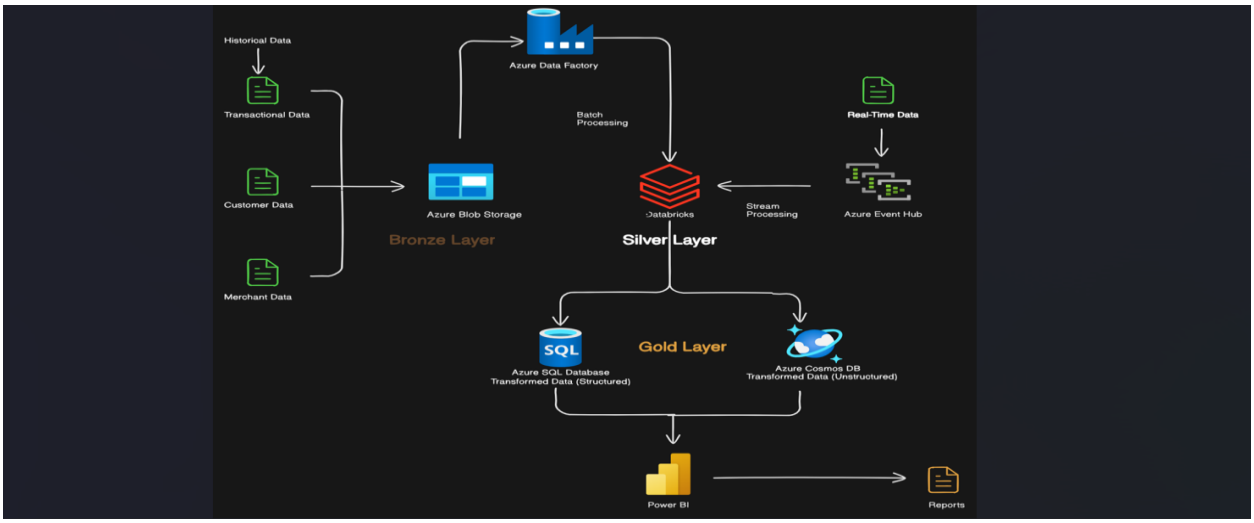
**Exploratory Data Analysis (EDA)**:

- **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.

    - **Silver Layer**: Azure Databricks for cleaning and transforming data. This layer processes and refines the data, making it ready for analysis.
    - **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**:
    - **Partitioning and indexing for fast querying in SQL**: Data was partitioned and indexed to improve query performance and reduce retrieval times.
    - **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**:
    - **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.
    - **Azure Databricks performs transformation and cleaning**: Data is transformed and cleaned in real-time using Azure Databricks, ensuring it is ready for immediate analysis.
    - **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**:
    - **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.
    - **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**:
    - **ETL Processing**: Transform data using Azure Databricks: Data is extracted, transformed, and loaded using Azure Databricks, ensuring it is clean and ready for analysis.
    - **Data Storage**: Store structured data in Azure SQL DB: Structured data is stored in Azure SQL Database, providing a reliable and scalable storage solution.
    - **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

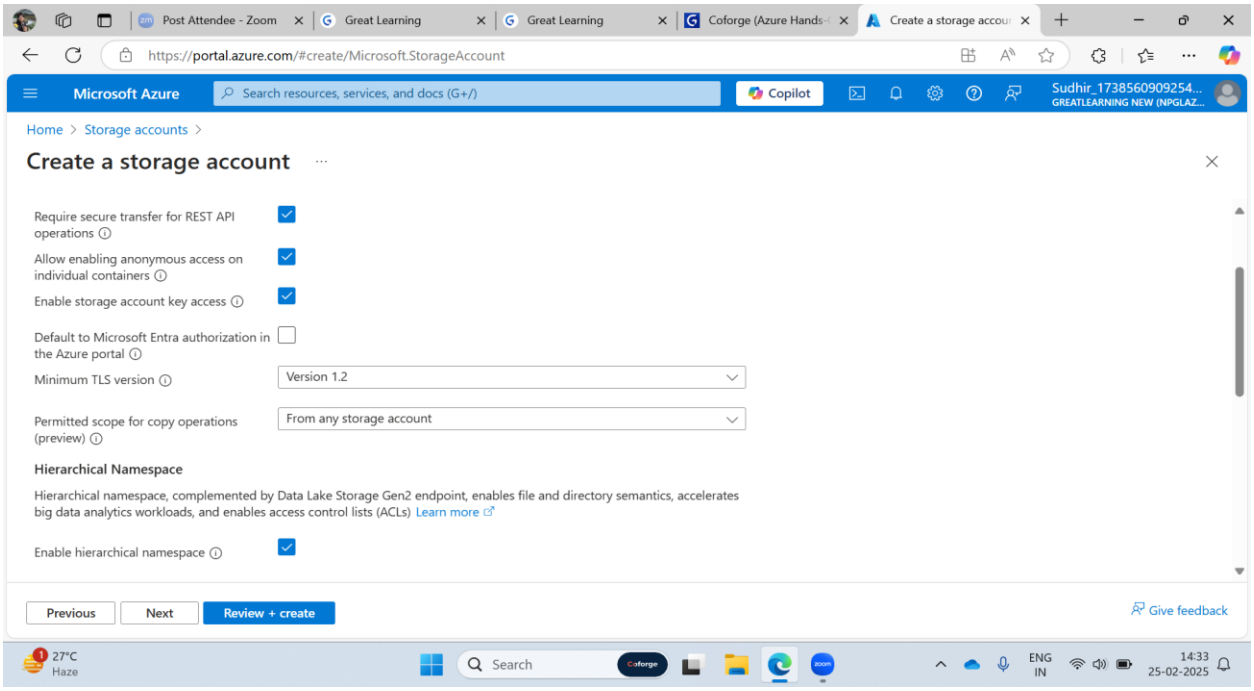
## *11. Appendices*

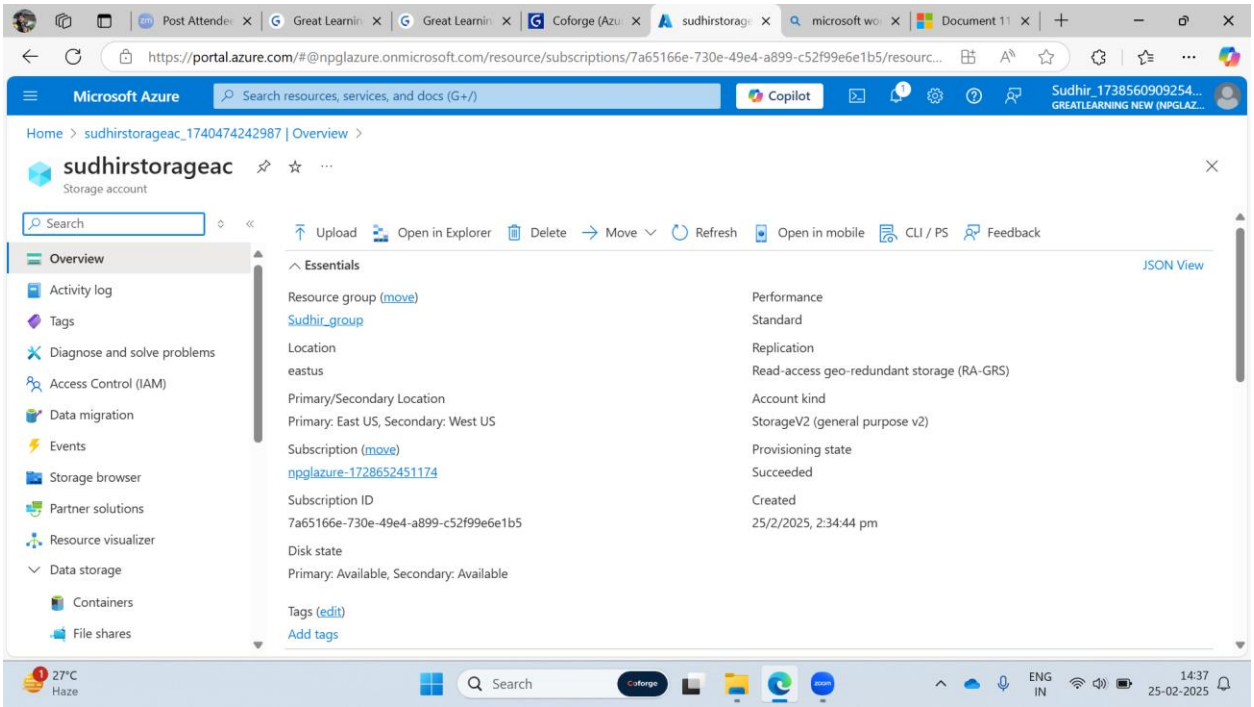
**Appendix A**: Code Snippets (Include Python/PySpark scripts used for transformation.) **Appendix B**: References (List relevant sources, documentation, or research materials.)

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.

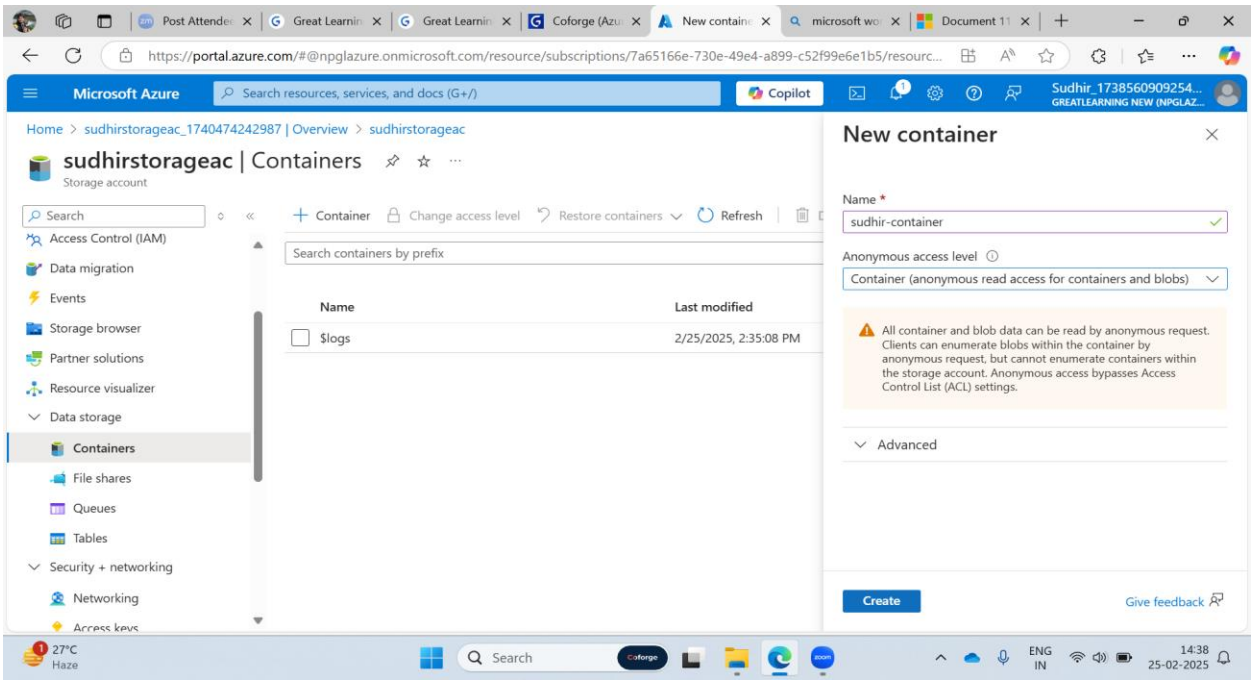
MID-REPORT -> Screenshots of Implementation:

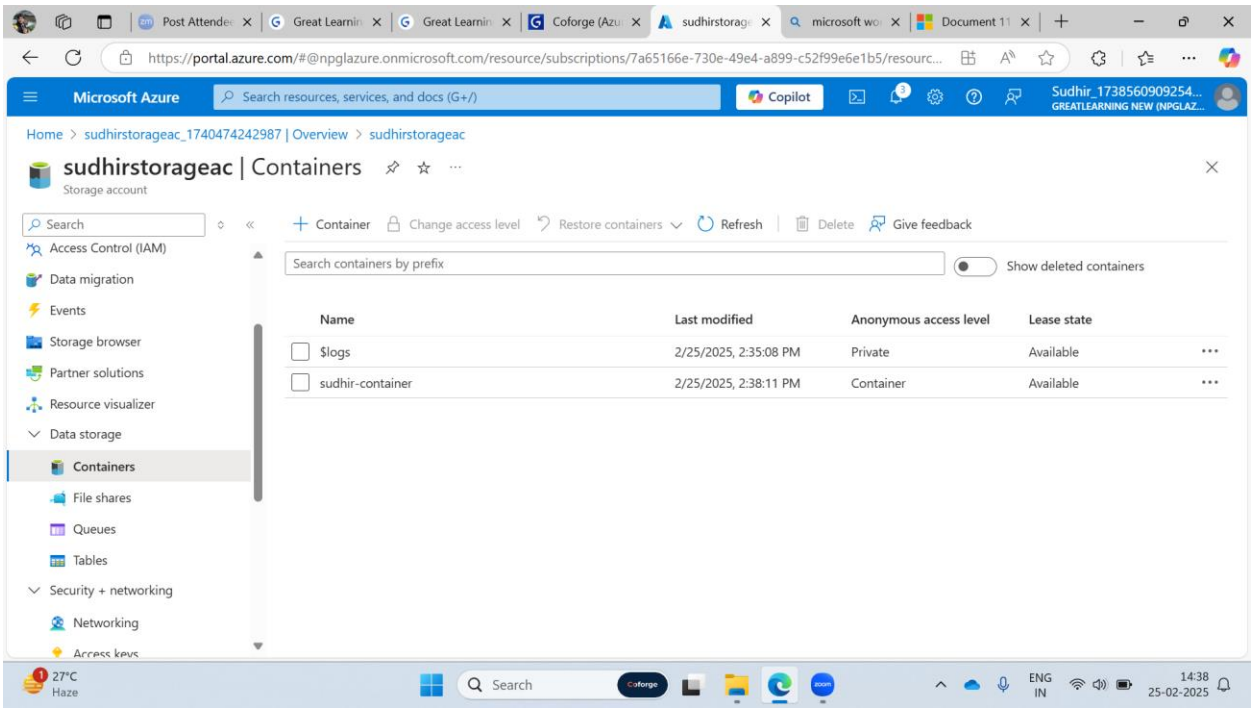
STORAGE ACCOUNT



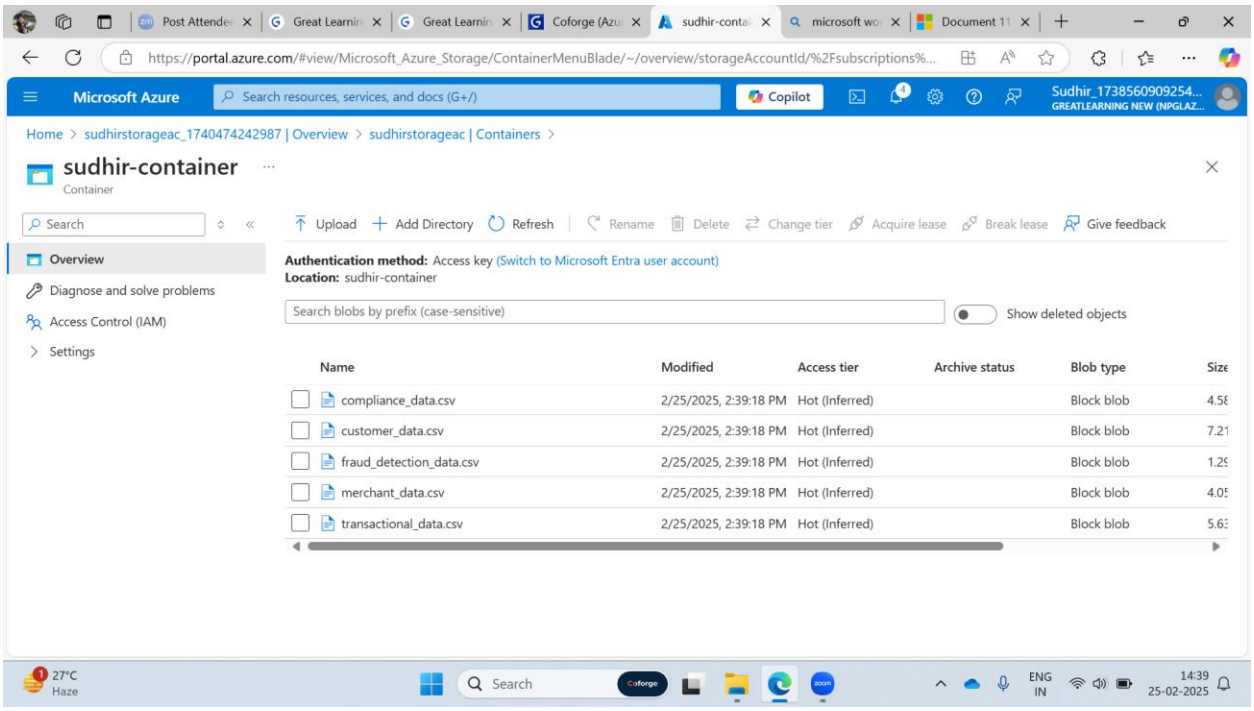



CONTAINER CREATION:

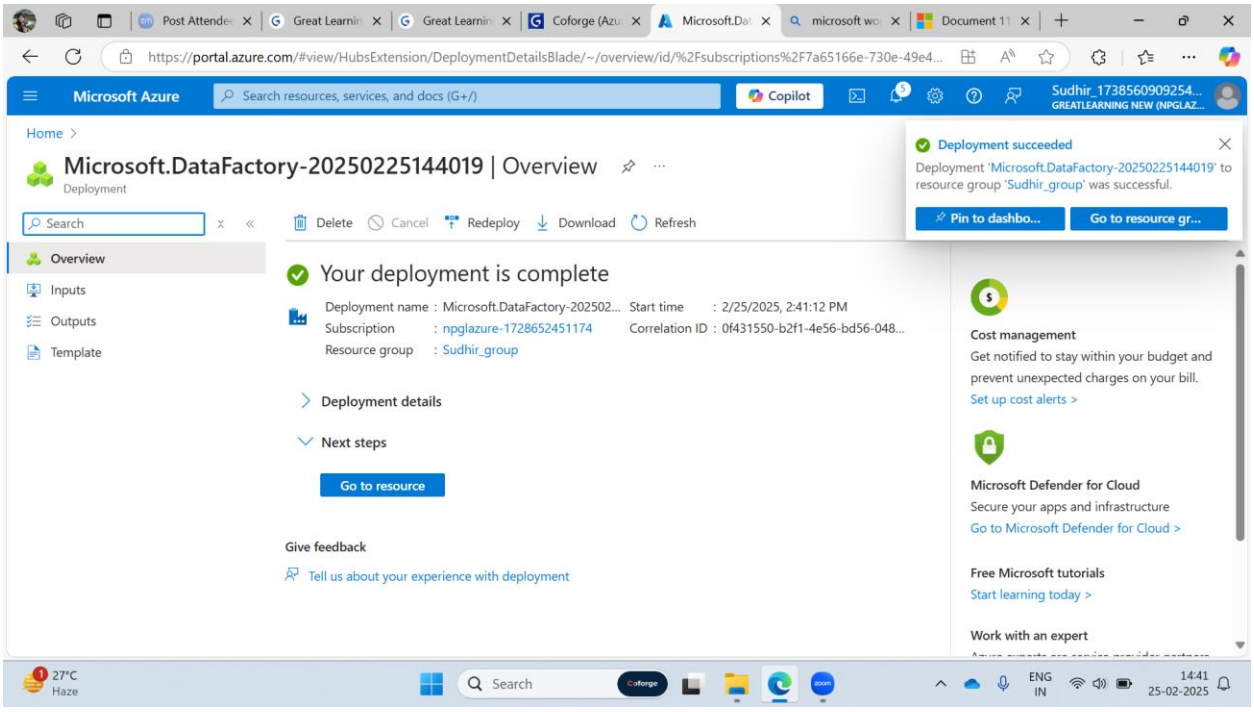


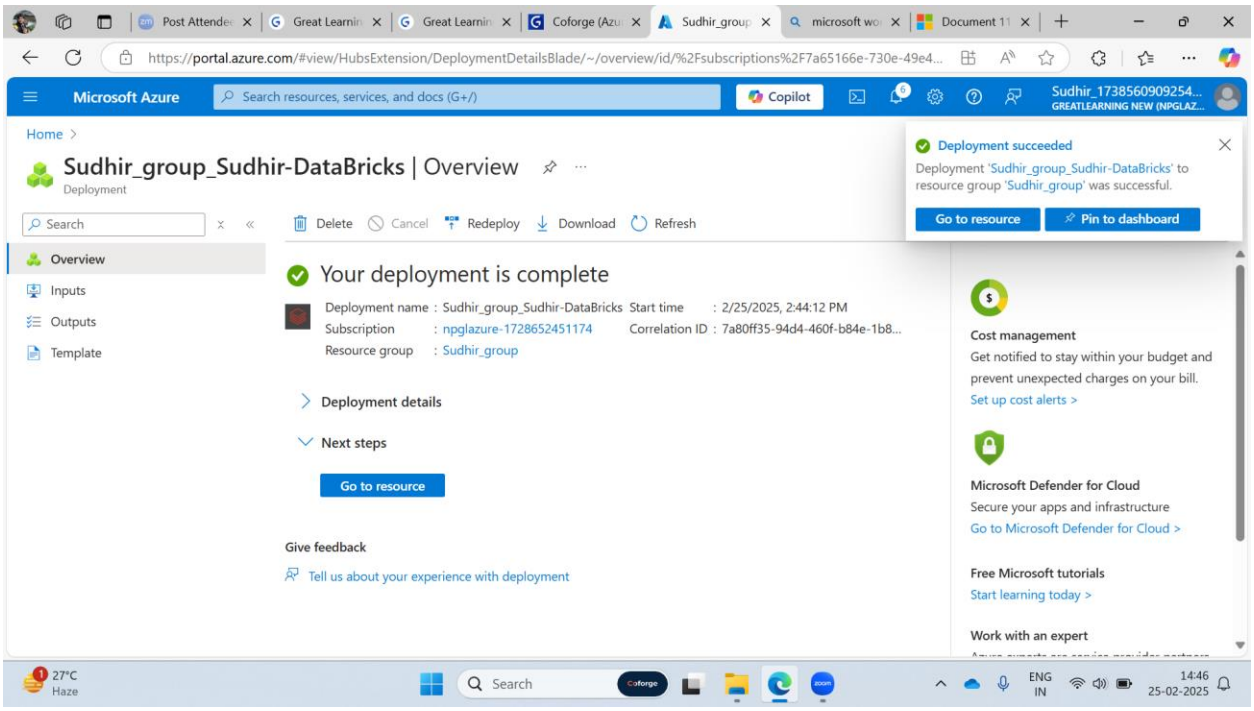

UPLOADING DATA IN CONTAINER:



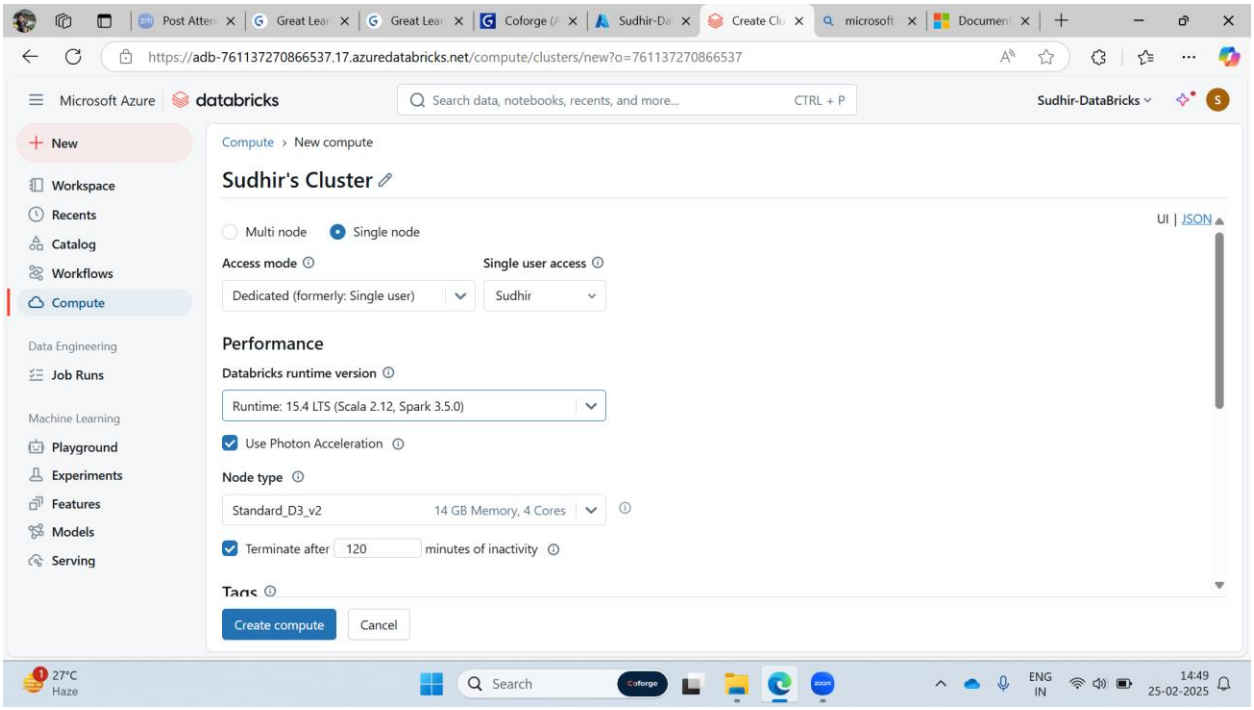

ADF CREATION

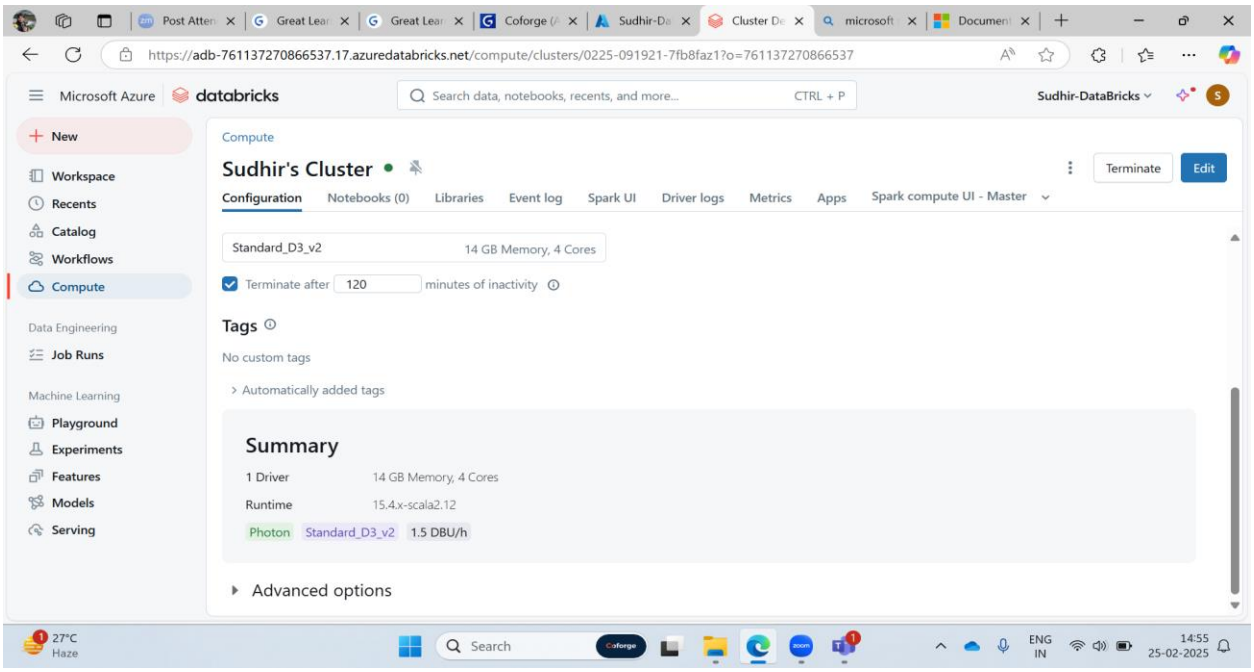



DATABRICKS CREATION: -

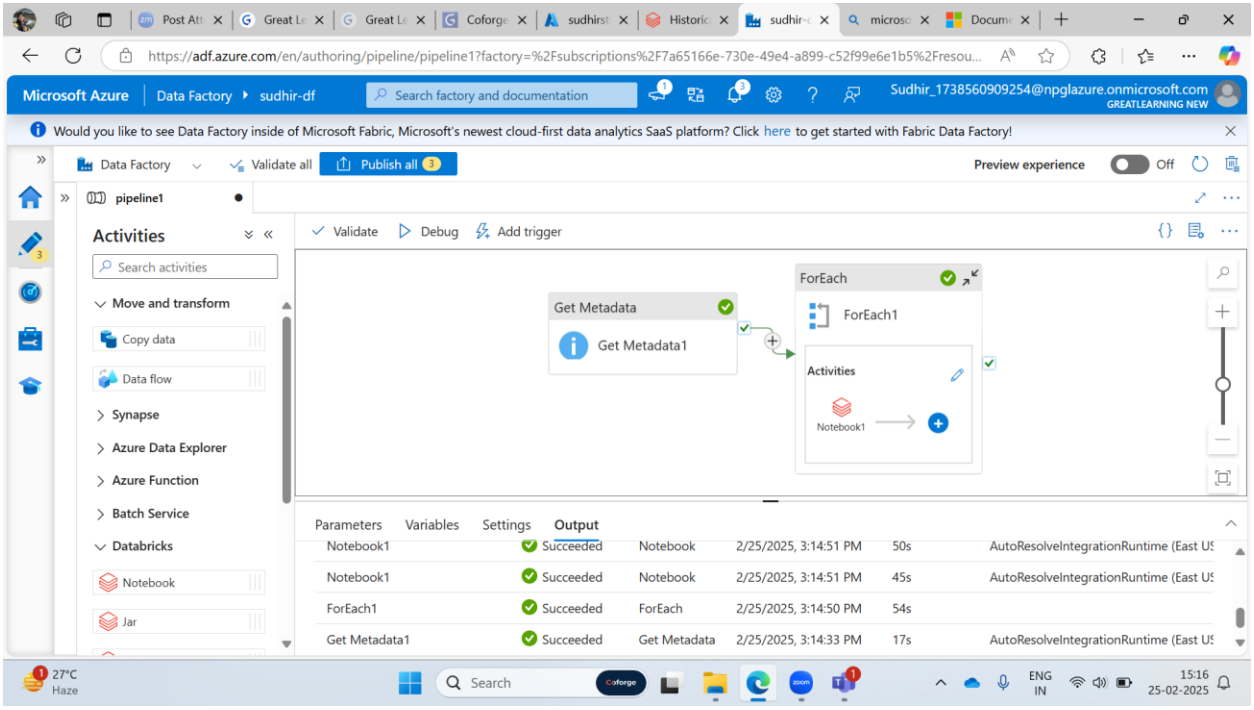


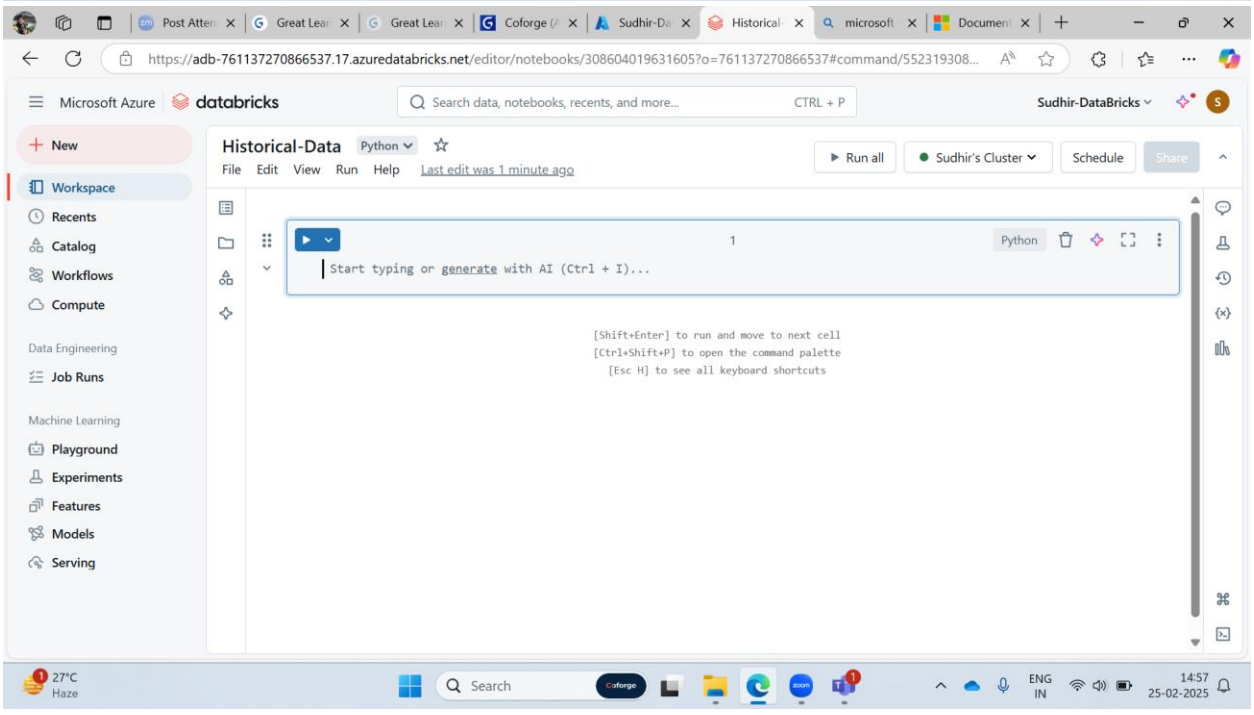

## CLUSTER CREATION



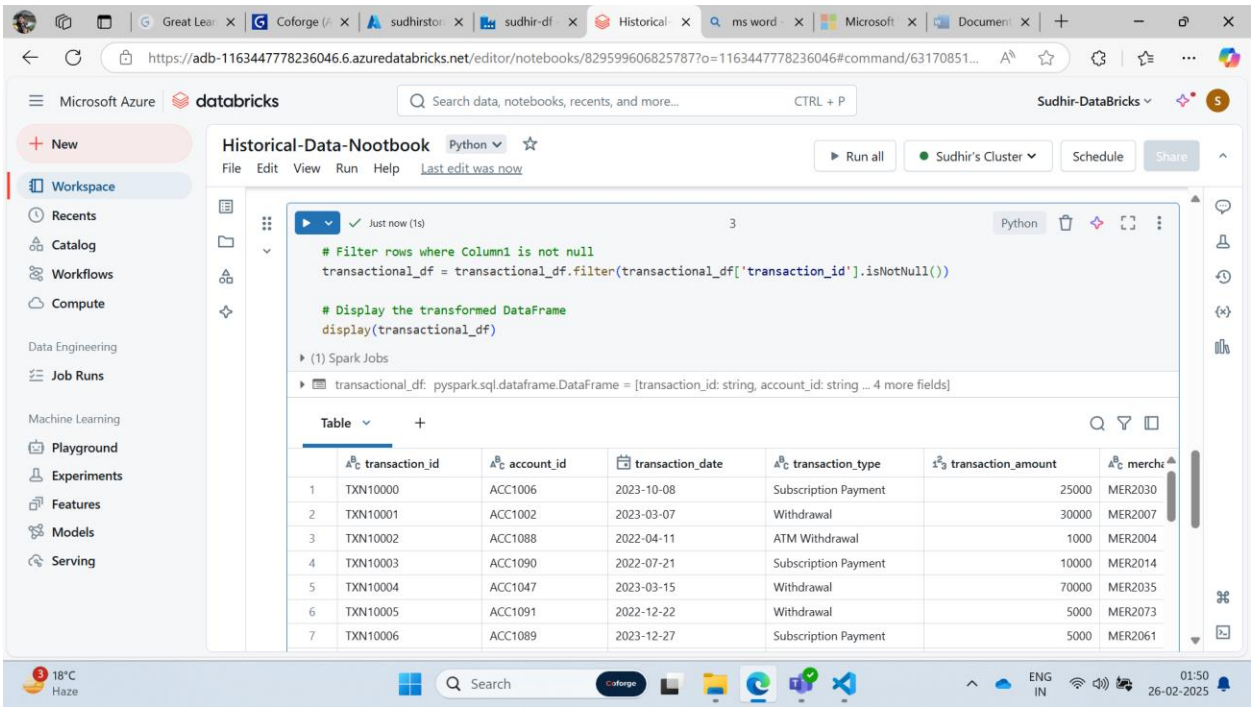

https://adb-761137270866537.17.azuredatabricks.net/compute/clusters/0225-091921-7fb8faz1?o=761137270866537
Microsoft Azure
databricks
Search data, notebooks, recents, and more...
CTRL + P
Sudhir-DataBricks
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Compute
Sudhir's Cluster
Terminate
Edit
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
14:55
25-02-2025

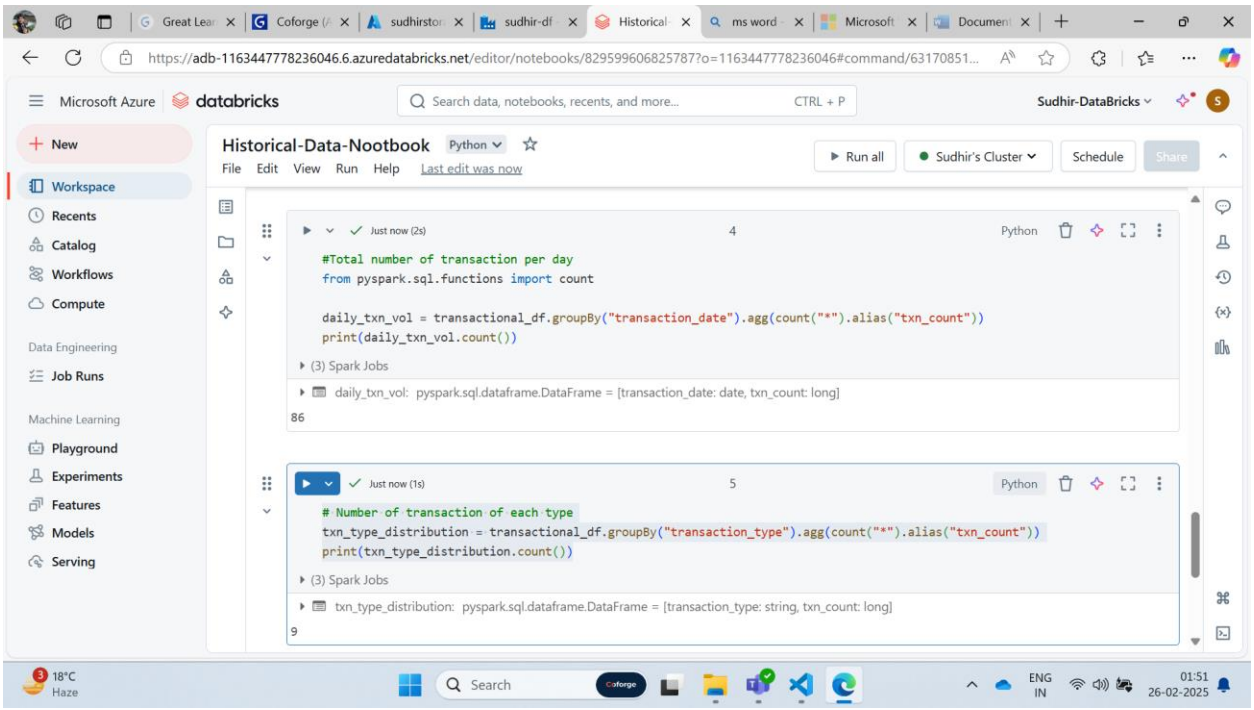
UPLOADING HISTORICAL DATA INTO ADF: h

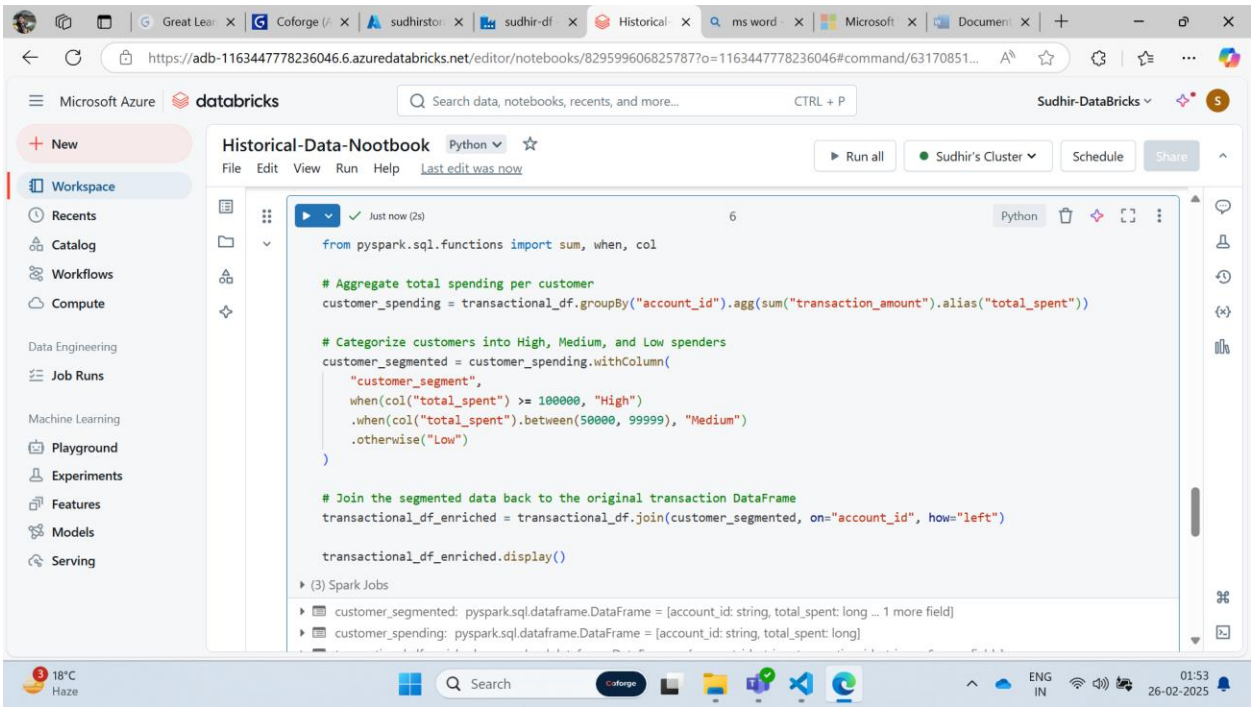


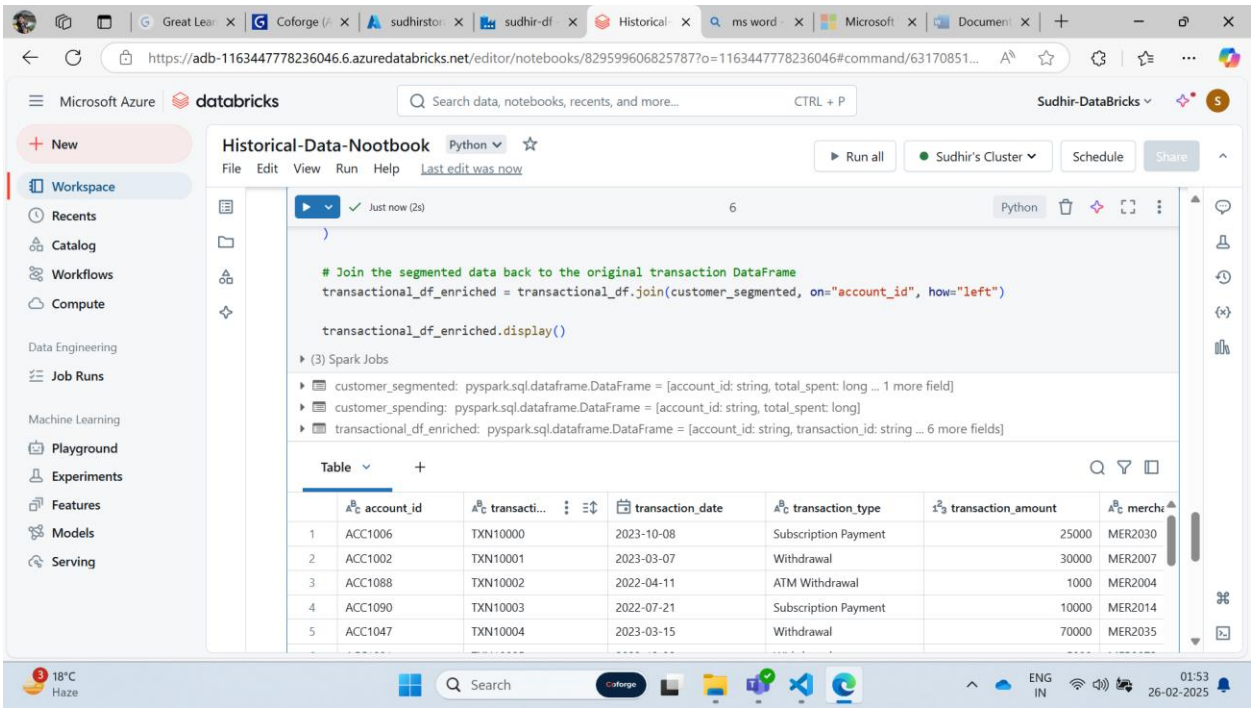



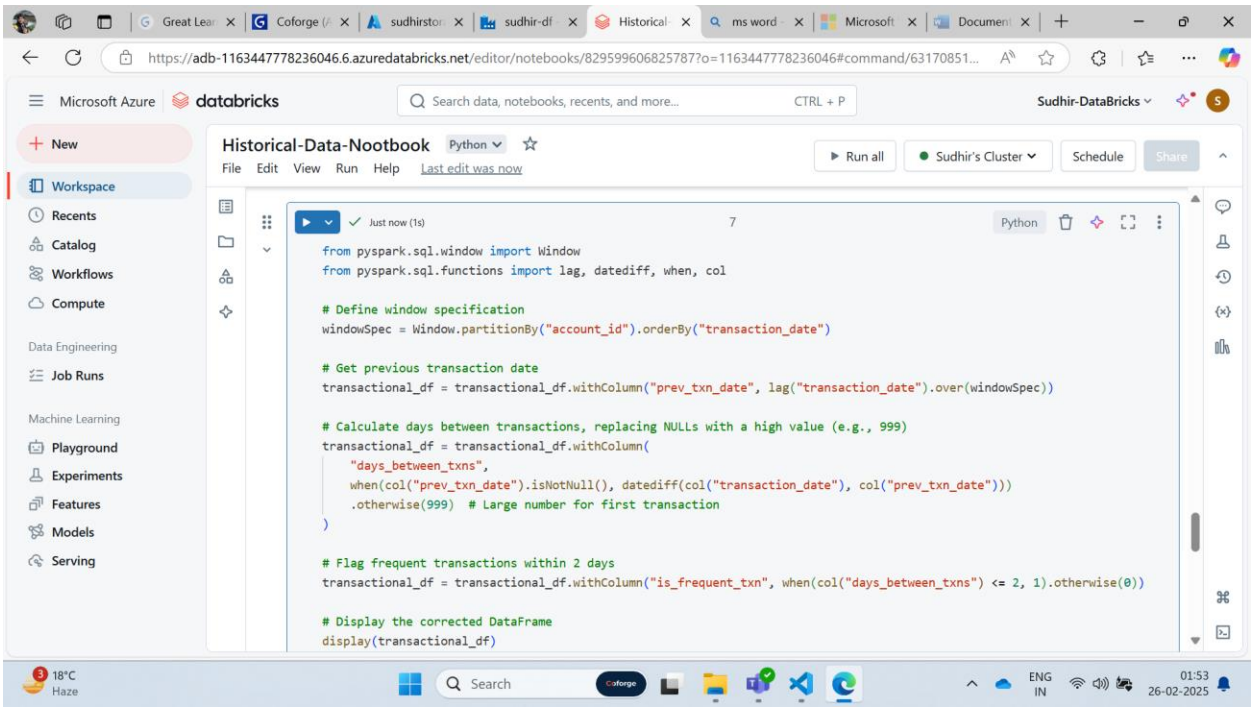

PERFORMING TRANSFORMATION ON HISTORICAL DATASET:

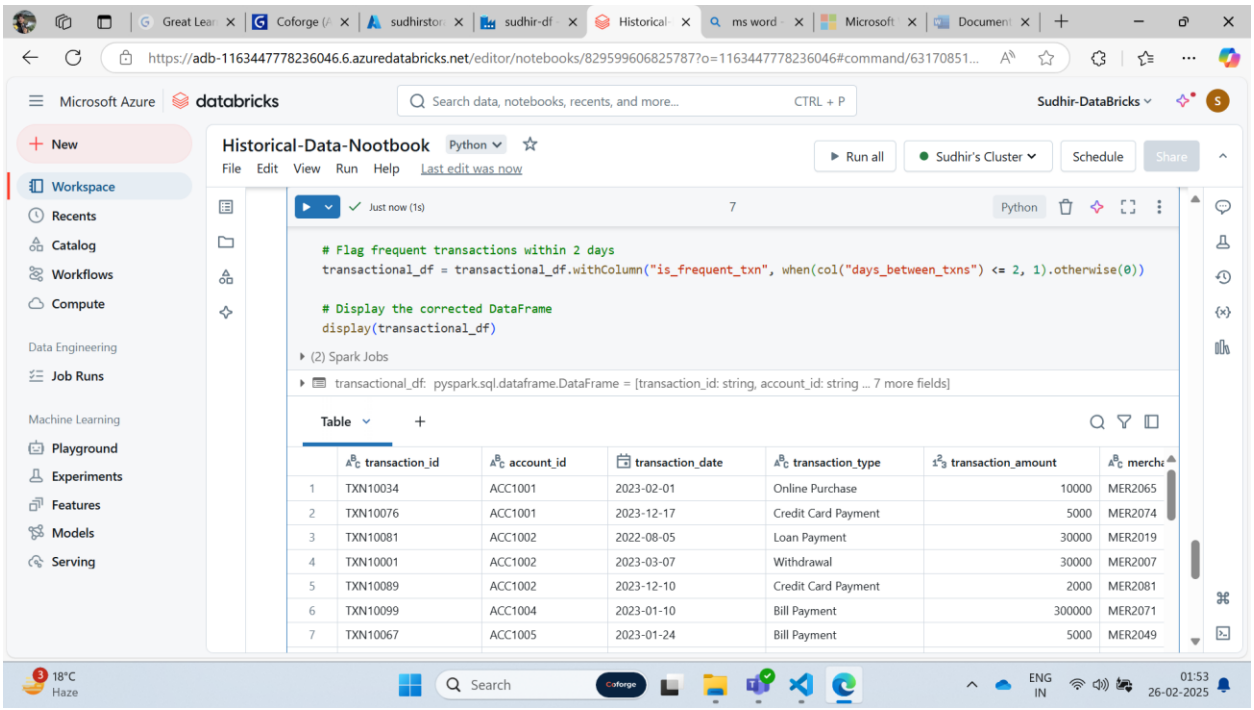

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Just now (1s)
3
# Filter rows where Column1 is not null
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
1 TXN10000 ACC1006 2023-10-08 Subscription Payment 25000 MER2030
2 TXN10001 ACC1002 2023-03-07 Withdrawal 30000 MER2007
3 TXN10002 ACC1088 2022-04-11 ATM Withdrawal 1000 MER2004
4 TXN10003 ACC1090 2022-07-21 Subscription Payment 10000 MER2014
5 TXN10004 ACC1047 2023-03-15 Withdrawal 70000 MER2035
6 TXN10005 ACC1091 2022-12-22 Withdrawal 5000 MER2073
7 TXN10006 ACC1089 2023-12-27 Subscription Payment 5000 MER2061

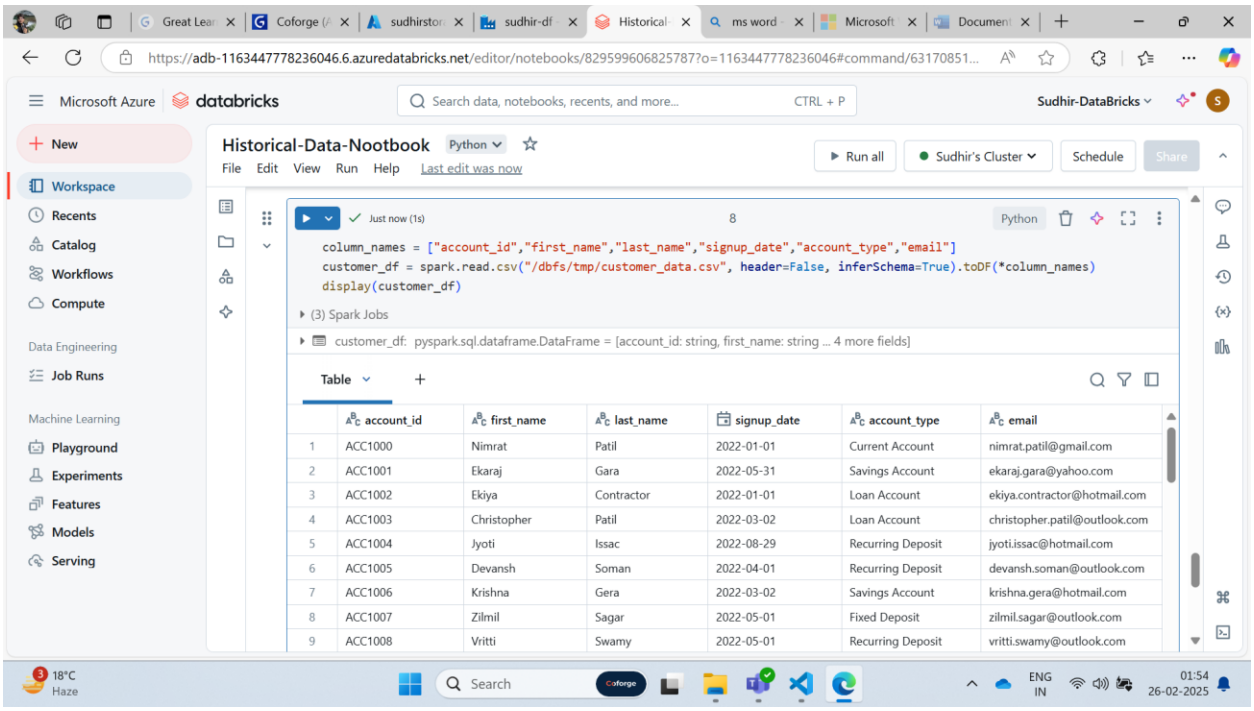

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (2s)
4
#Total number of transaction per day
from pyspark.sql.functions import count
daily_txn_vol = transactional_df.groupBy("transaction_date").agg(count("*").alias("txn_count"))
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)
5
# 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

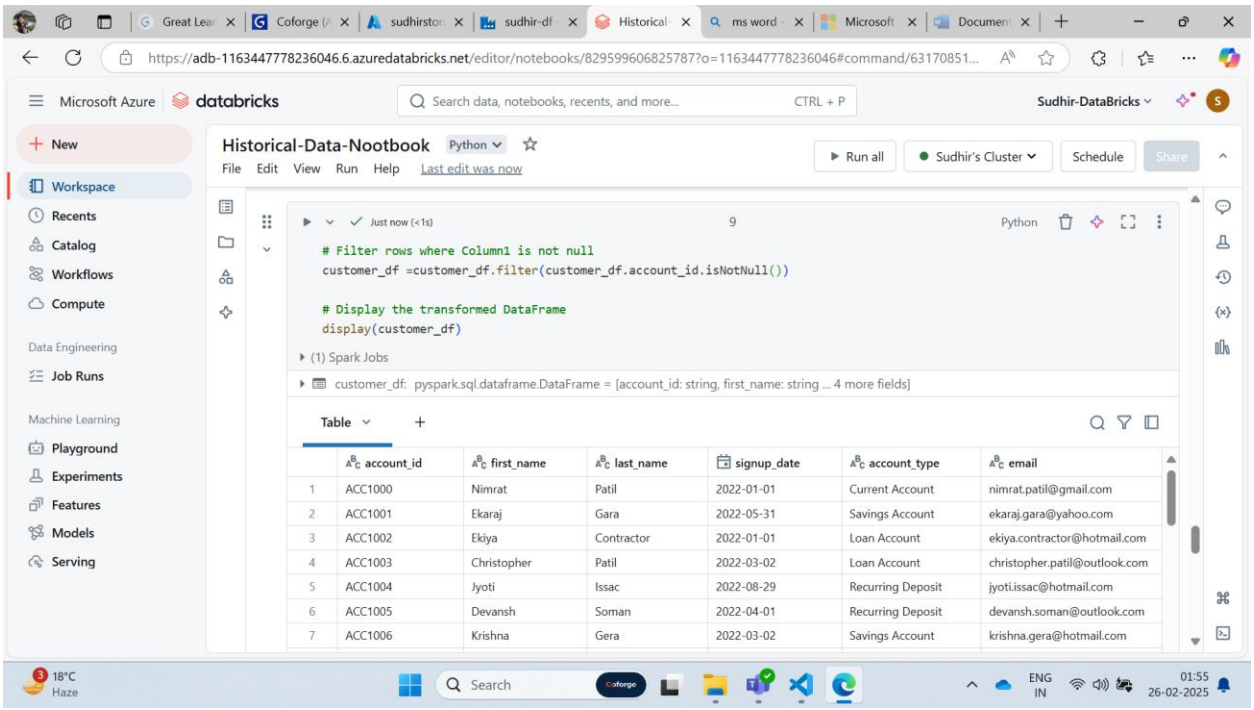

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (2s)
6
Python
from pyspark.sql.functions import sum, when, col
# Aggregate total spending per customer
customer_spending = transactional_df.groupBy("account_id").agg(sum("transaction_amount").alias("total_spent"))
# 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]

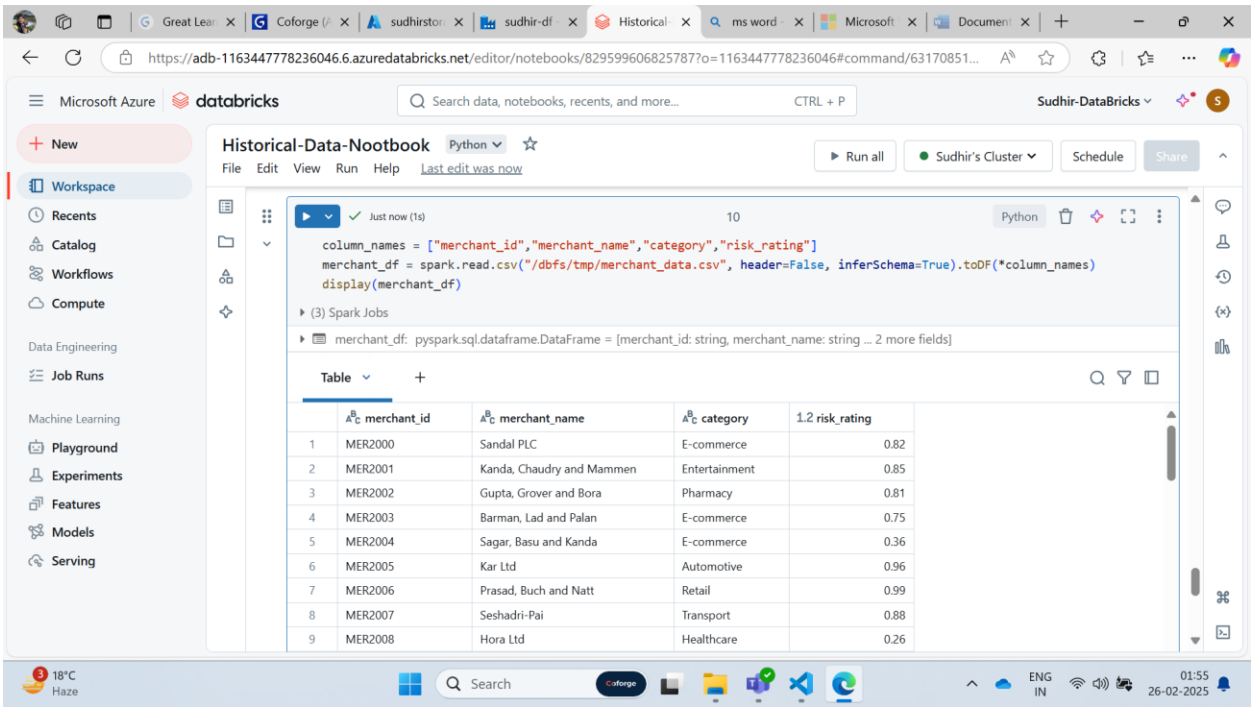

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (2s)
6
Python
# Join the segmented data back to the original transaction DataFrame
transactional_df_enriched = transactional_df.join(customer_segmented, on="account_id", how="left")
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]
transactional_df_enriched: pyspark.sql.dataframe.DataFrame = [account_id: string, transaction_id: string ... 6 more fields]
Table
account_id
transacti...
transaction_date
transaction_type
transaction_amount
merch
1 ACC1006 TXN10000 2023-10-08 Subscription Payment 25000 MER2030
2 ACC1002 TXN10001 2023-03-07 Withdrawal 30000 MER2007
3 ACC1088 TXN10002 2022-04-11 ATM Withdrawal 1000 MER2004
4 ACC1090 TXN10003 2022-07-21 Subscription Payment 10000 MER2014
5 ACC1047 TXN10004 2023-03-15 Withdrawal 70000 MER2035

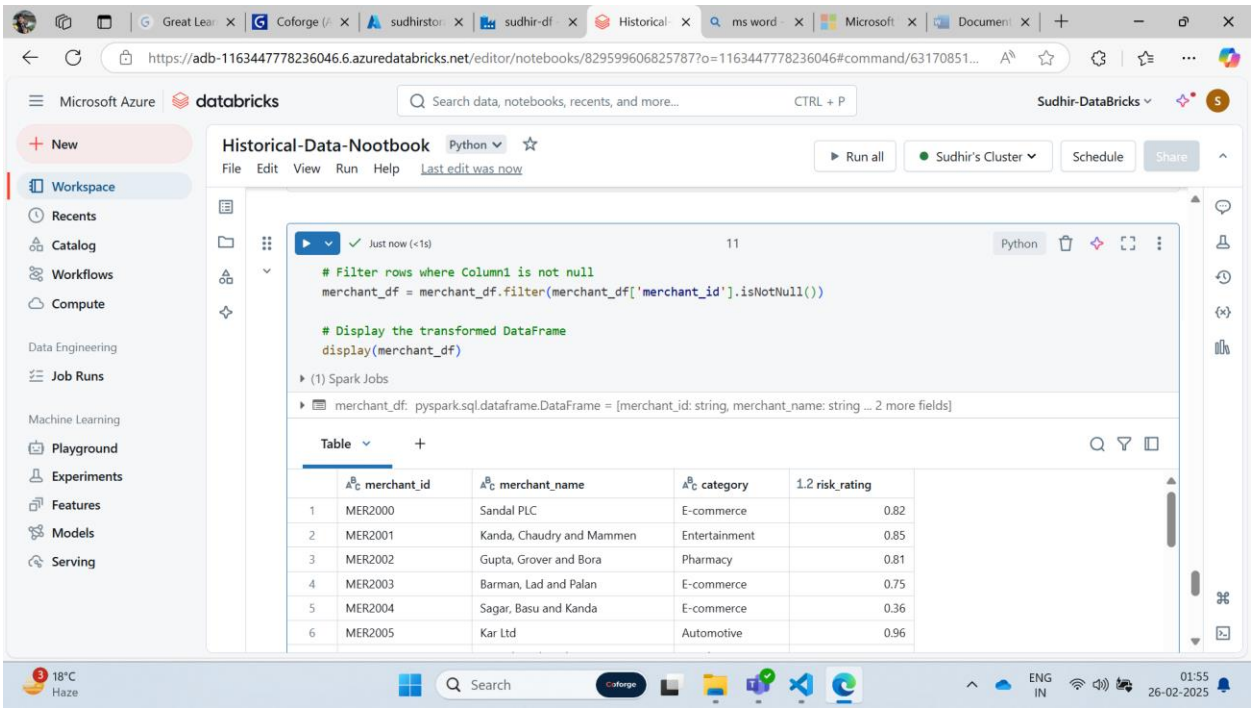

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (1s)
7
Python
from pyspark.sql.window import Window
from pyspark.sql.functions import lag, datediff, when, col
# Define window specification
windowSpec = Window.partitionBy("account_id").orderBy("transaction_date")
# 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)

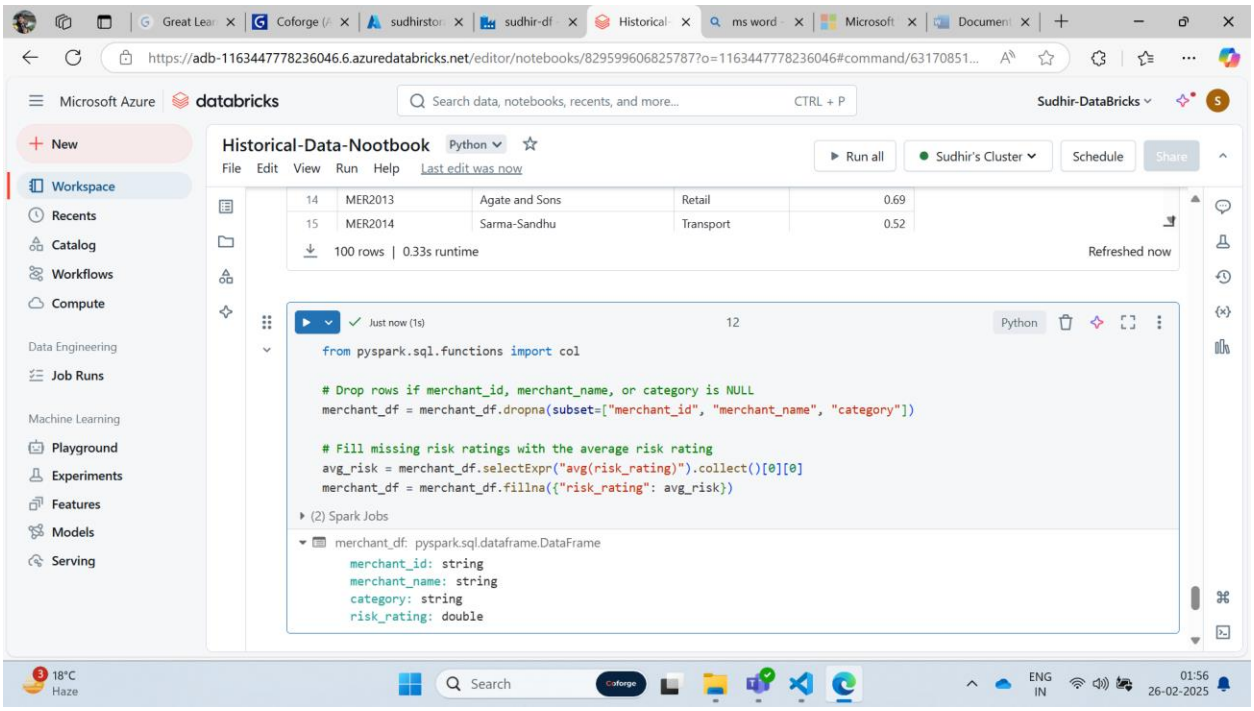

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (1s)
7
Python
# Flag frequent transactions within 2 days
transactional_df = transactional_df.withColumn("is_frequent_txn", when(col("days_between_txns") <= 2, 1).otherwise(0))
# Display the corrected DataFrame
display(transactional_df)
(2) Spark Jobs
transactional_df: pyspark.sql.dataframe.DataFrame = [transaction_id: string, account_id: string ... 7 more fields]
Table
transaction_id
account_id
transaction_date
transaction_type
transaction_amount
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

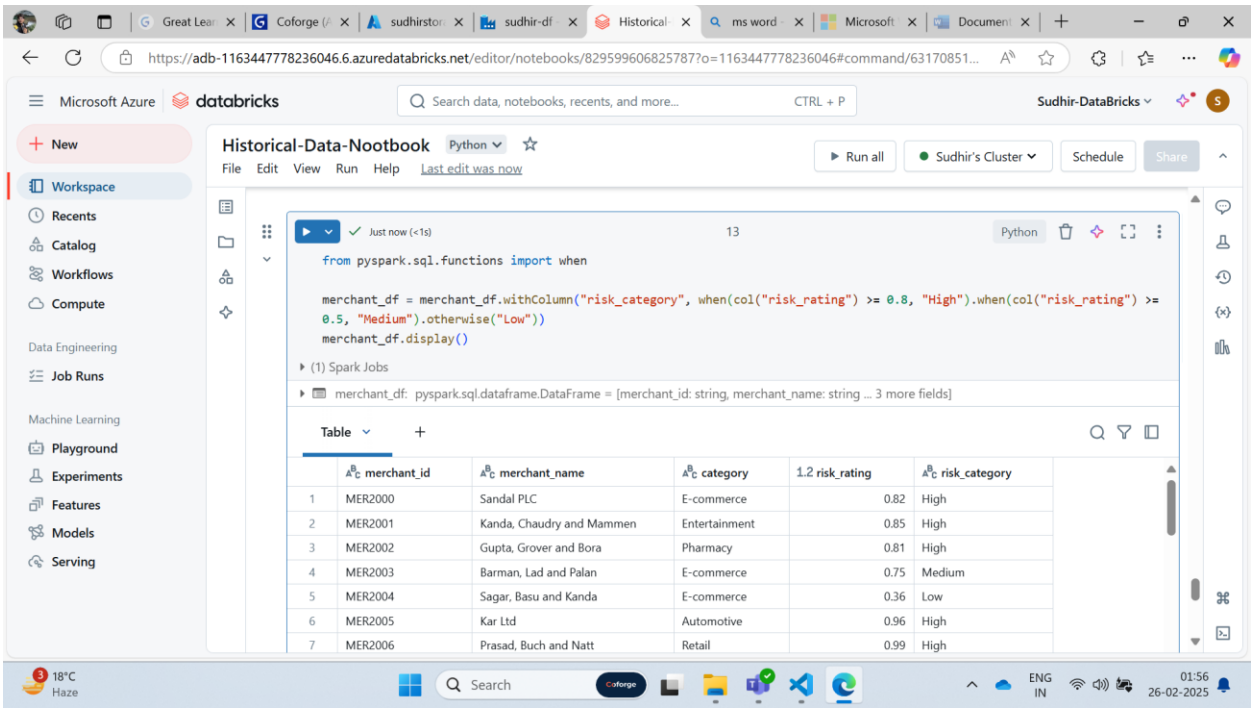

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (1s)
8
Python
column_names = ["account_id","first_name","last_name","signup_date","account_type","email"]
customer_df = spark.read.csv("/dbfs/tmp/customer_data.csv", header=False, 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

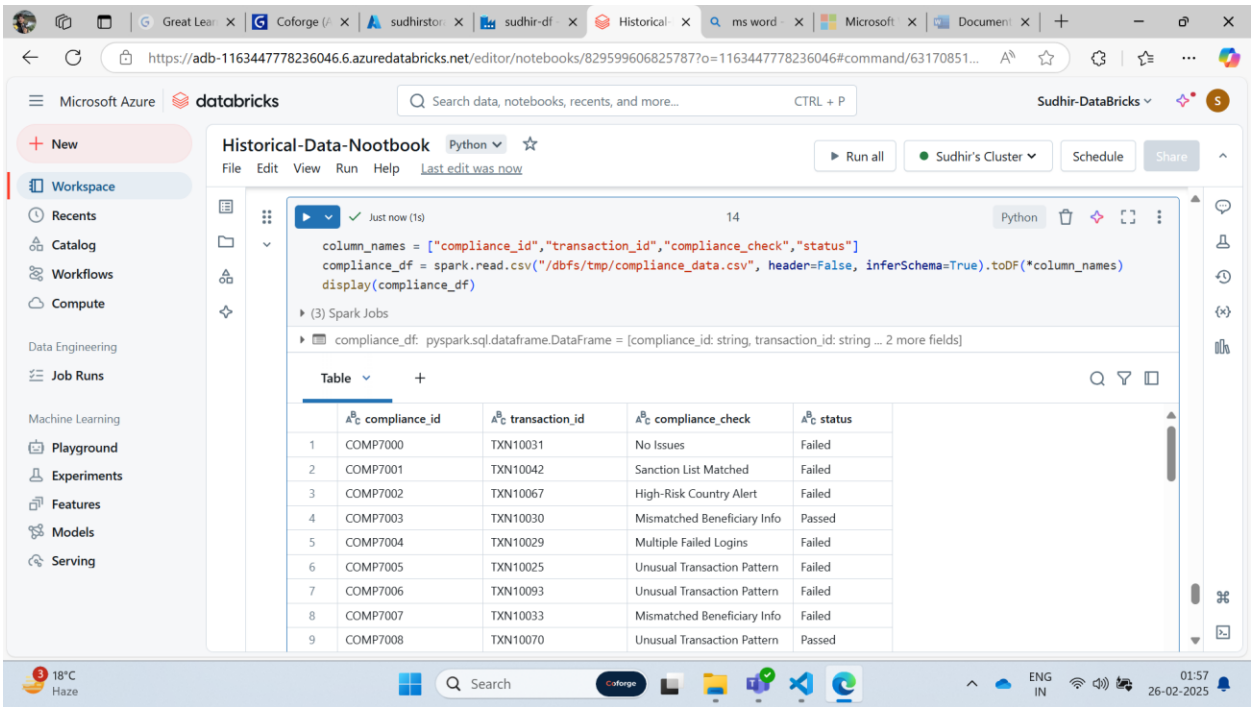

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (<1s)
9
Python
# Filter rows where Column1 is not null
customer_df =customer_df.filter(customer_df.account_id.isNotNull())
# Display the transformed DataFrame
display(customer_df)
(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

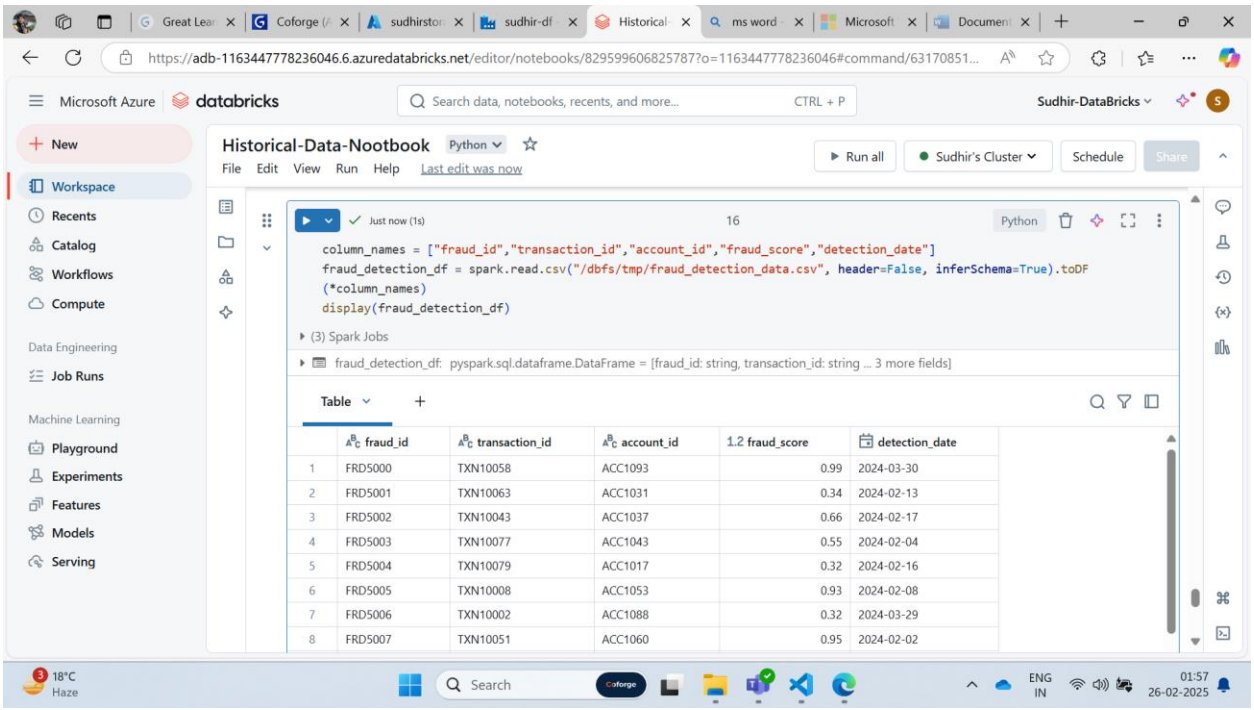

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (1s)
10
column_names = ["merchant_id","merchant_name","category","risk_rating"]
merchant_df = spark.read.csv("/dbfs/tmp/merchant_data.csv", header=False, inferSchema=True).toDF(*column_names)
display(merchant_df)
(3) Spark Jobs
merchant_df: pyspark.sql.dataframe.DataFrame = [merchant_id: string, merchant_name: string ... 2 more fields]
Table
merchant_id | merchant_name | category | 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

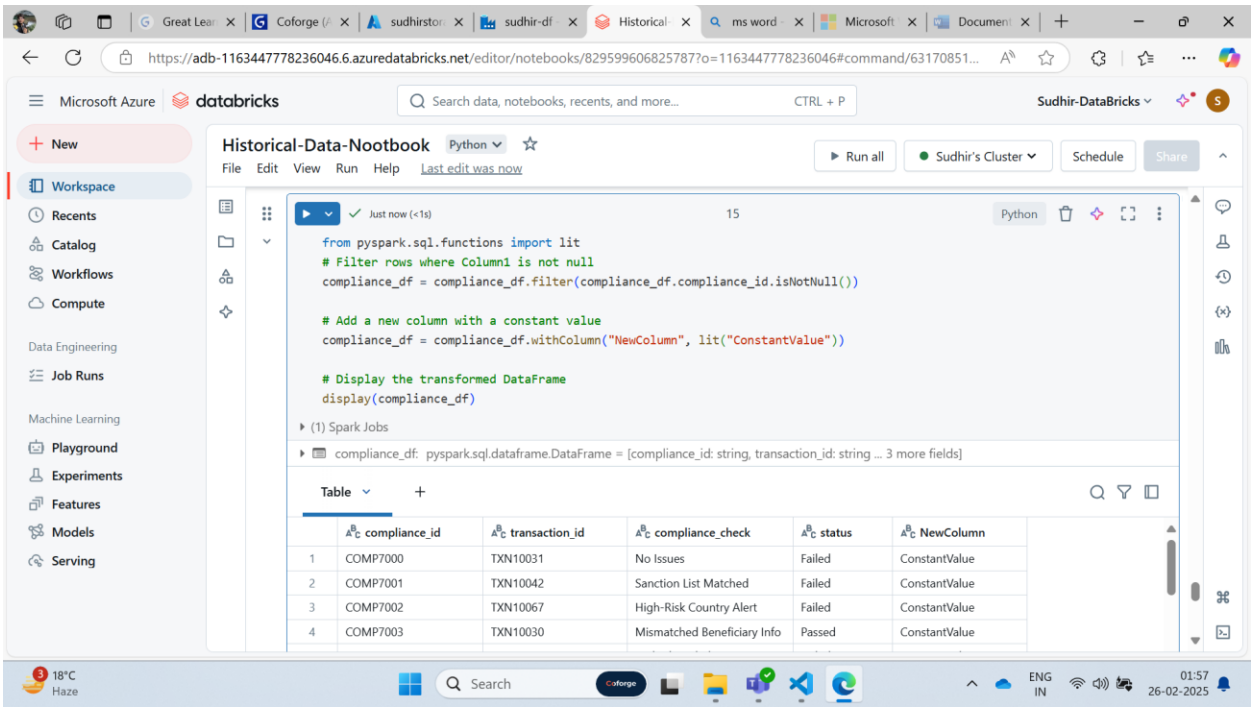

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (<1s)
11
# Filter rows where Column1 is not null
merchant_df = merchant_df.filter(merchant_df['merchant_id'].isNotNull())
# Display the transformed DataFrame
display(merchant_df)
(1) Spark Jobs
merchant_df: pyspark.sql.dataframe.DataFrame = [merchant_id: string, merchant_name: string ... 2 more fields]
Table
merchant_id | merchant_name | category | 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

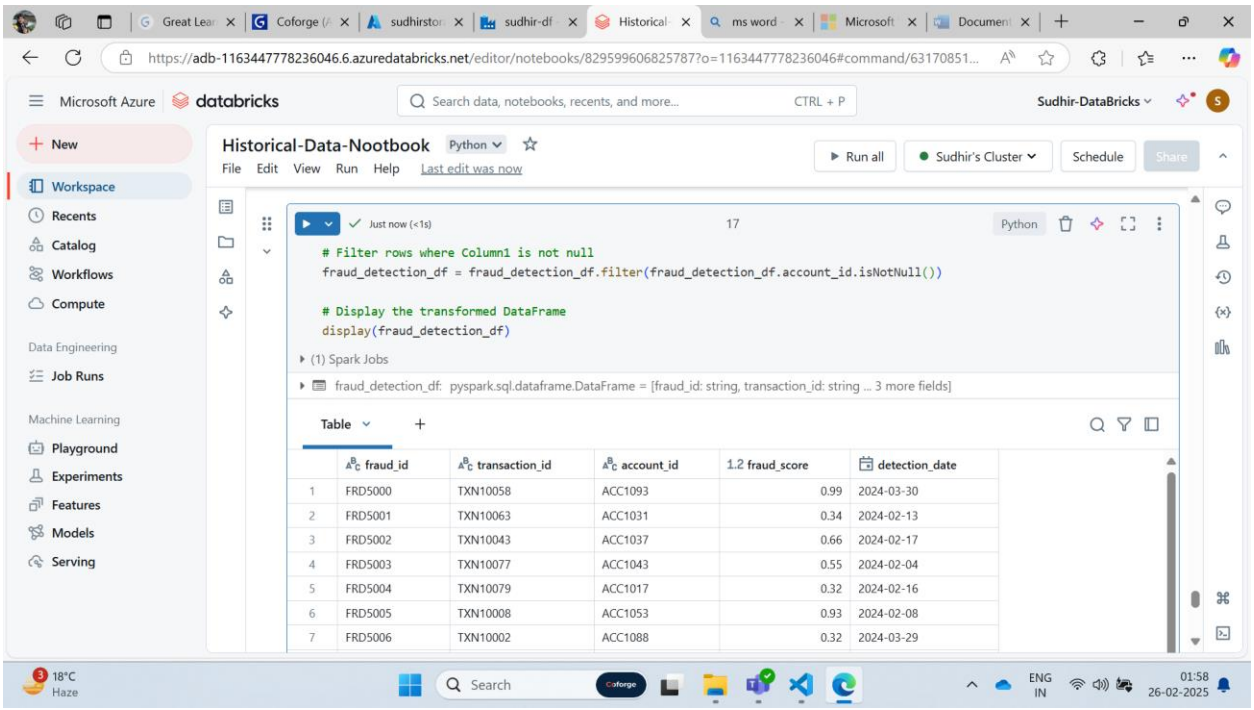

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
14 MER2013 Agate and Sons Retail 0.69
15 MER2014 Sarma-Sandhu Transport 0.52
100 rows | 0.33s runtime
Refreshed now
Just now (1s)
12
Python
from pyspark.sql.functions import col
# Drop rows 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

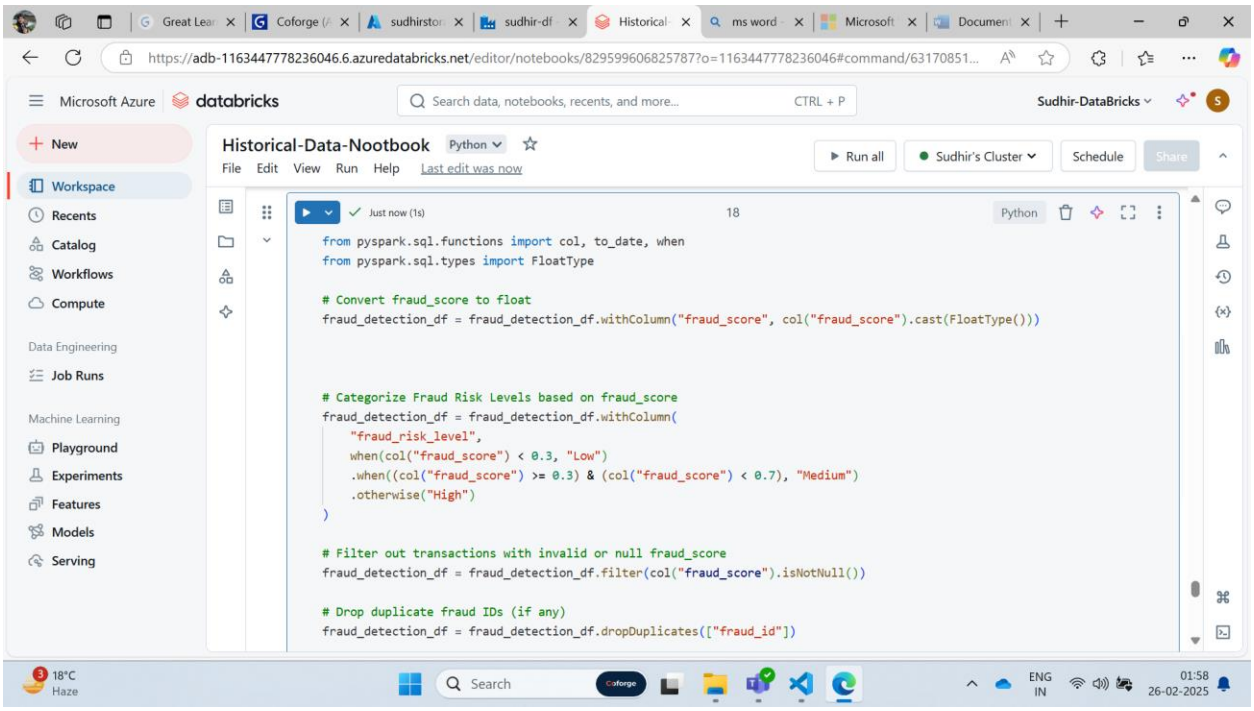

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (<1s)
13
Python
from pyspark.sql.functions import when
merchant_df = merchant_df.withColumn("risk_category", when(col("risk_rating") >= 0.8, "High").when(col("risk_rating") >= 0.5, "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 1.2 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

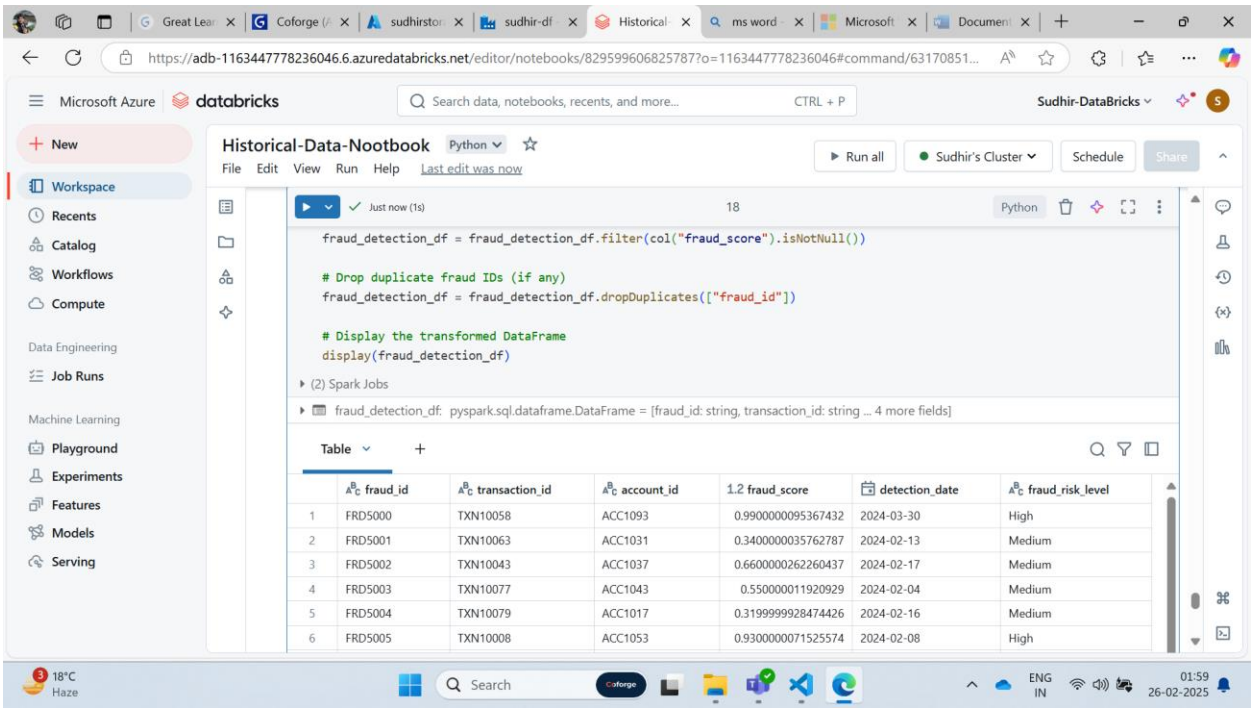

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (1s)
14
Python
column_names = ["compliance_id","transaction_id","compliance_check","status"]
compliance_df = spark.read.csv("/dbfs/tmp/compliance_data.csv", header=False, inferSchema=True).toDF(*column_names)
display(compliance_df)
(3) Spark Jobs
compliance_df: pyspark.sql.dataframe.DataFrame = [compliance_id: string, transaction_id: string ... 2 more fields]
Table
compliance_id transaction_id compliance_check 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


Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (1s)
16
Python
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
fraud_id transaction_id account_id 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


Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (<1s)
15
Python
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
compliance_id
transaction_id
compliance_check
status
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

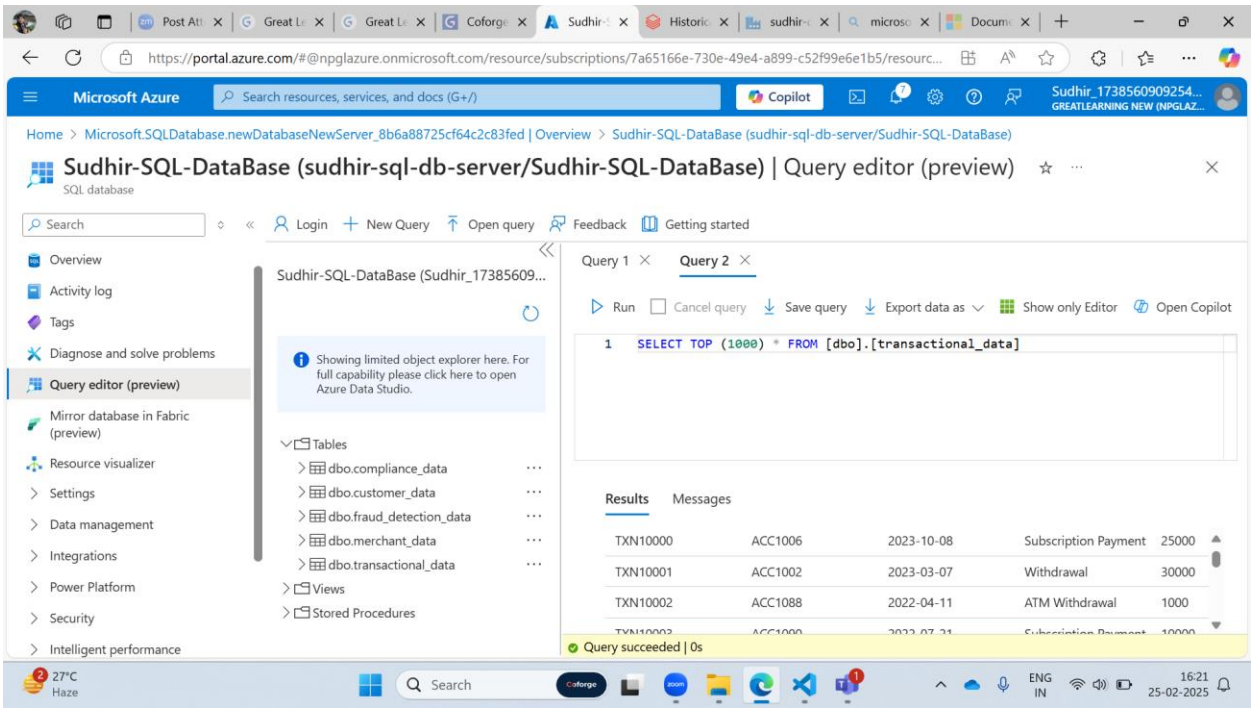

Historical-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Run all
Sudhir's Cluster
Schedule
Share
Just now (<1s)
17
Python
# Filter rows where Column1 is not null
fraud_detection_df = fraud_detection_df.filter(fraud_detection_df.account_id.isNotNull())
# Display the transformed DataFrame
display(fraud_detection_df)
(1) Spark Jobs
fraud_detection_df: pyspark.sql.dataframe.DataFrame = [fraud_id: string, transaction_id: string ... 3 more fields]
Table
fraud_id
transaction_id
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



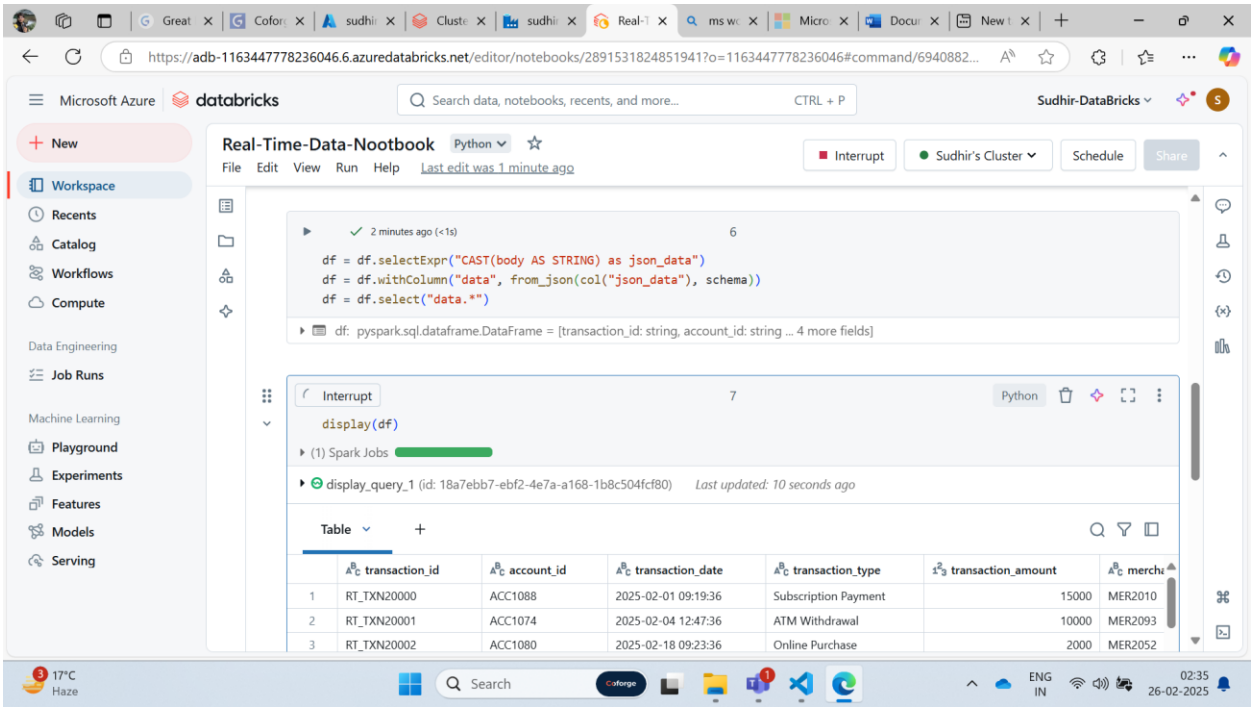

| | fraud_id | transaction_id | account_id | fraud_score | detection_date | fraud_risk_level |
|---|---|---|---|---|---|---|
| 1 | FRD5000 | TXN10058 | ACC1093 | 0.9900000095367432 | 2024-03-30 | High |
| 2 | FRD5001 | TXN10063 | ACC1031 | 0.3400000035762787 | 2024-02-13 | Medium |
| 3 | FRD5002 | TXN10043 | 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 |

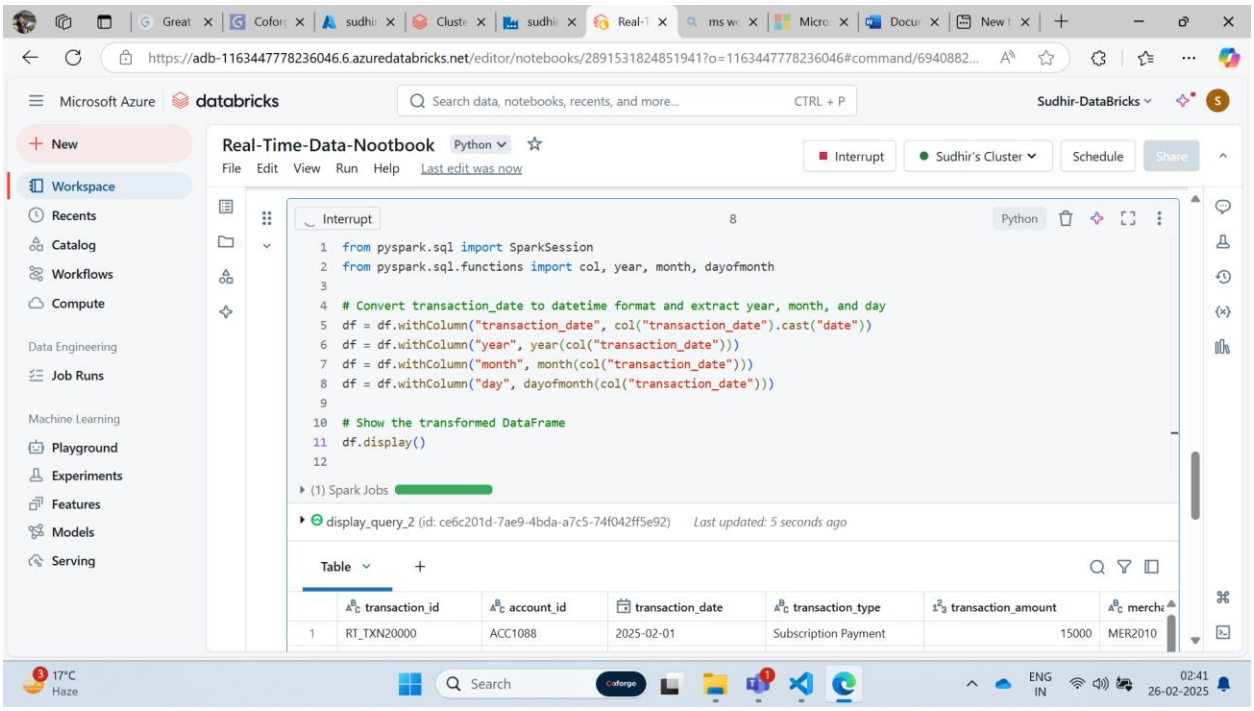
DATA IN SQL DATABASE:

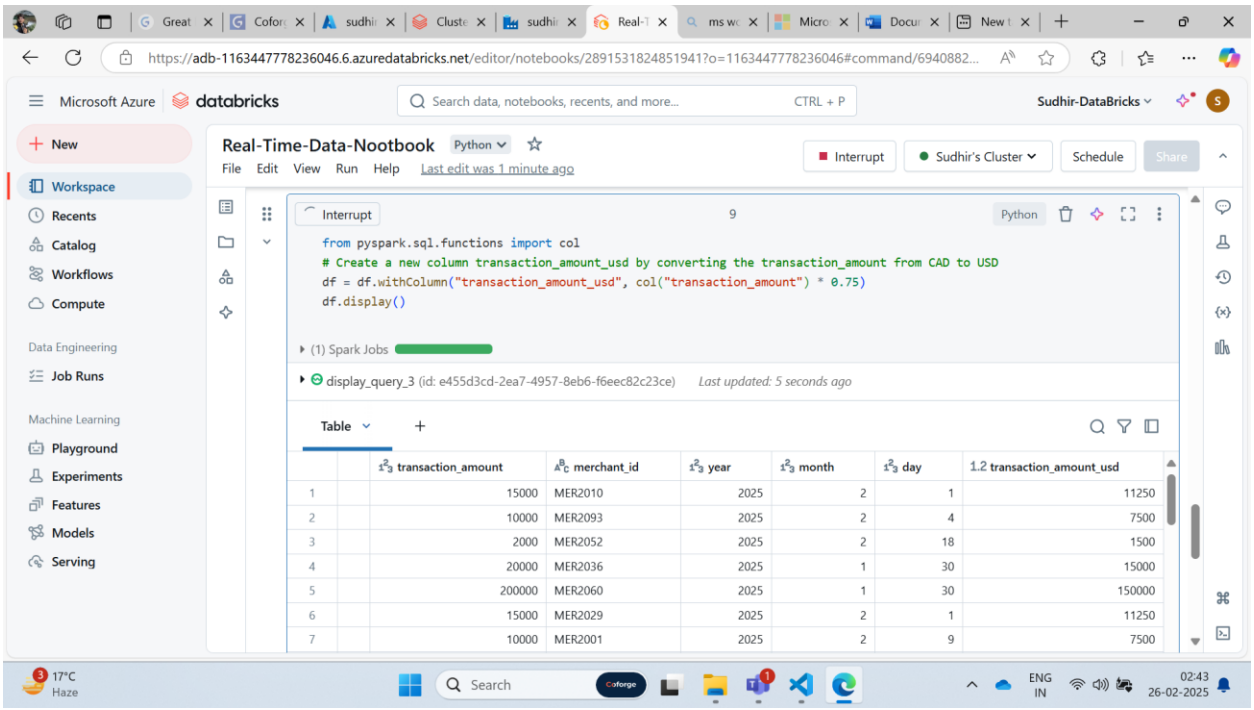



POWER BI REPORT OF HISTORICAL DATASET:

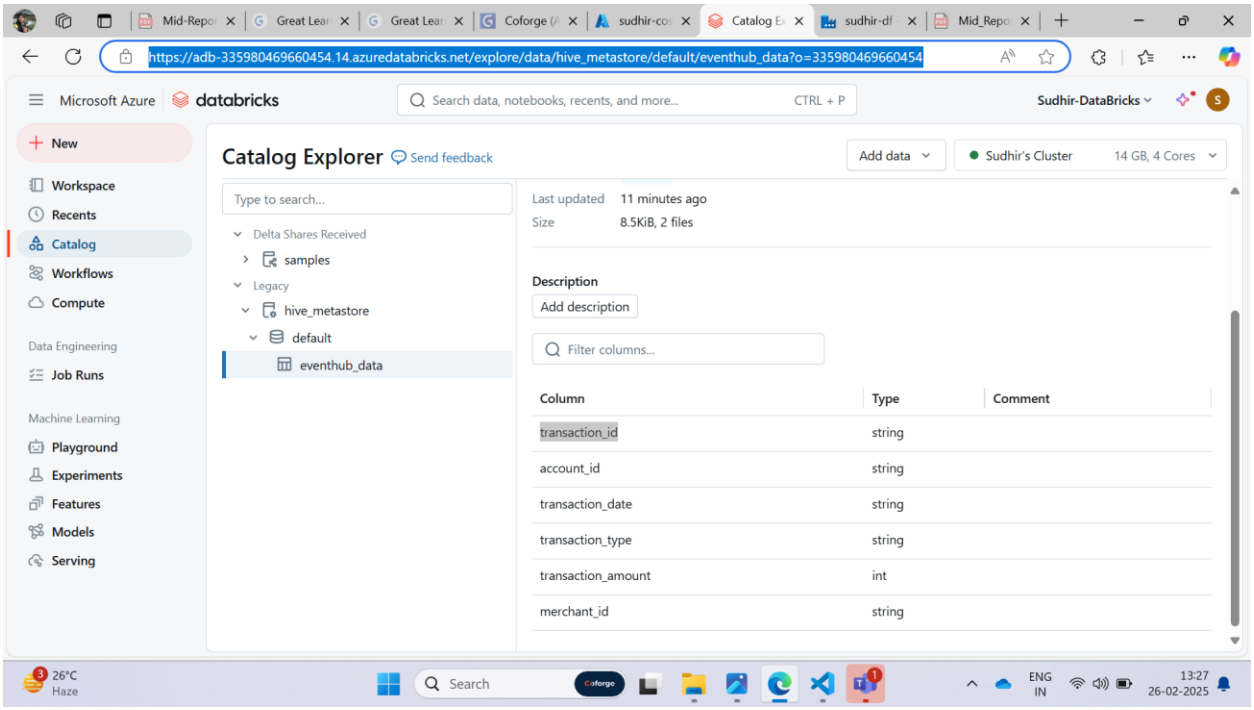
TRANSFORMATION IN REALTIME DATA

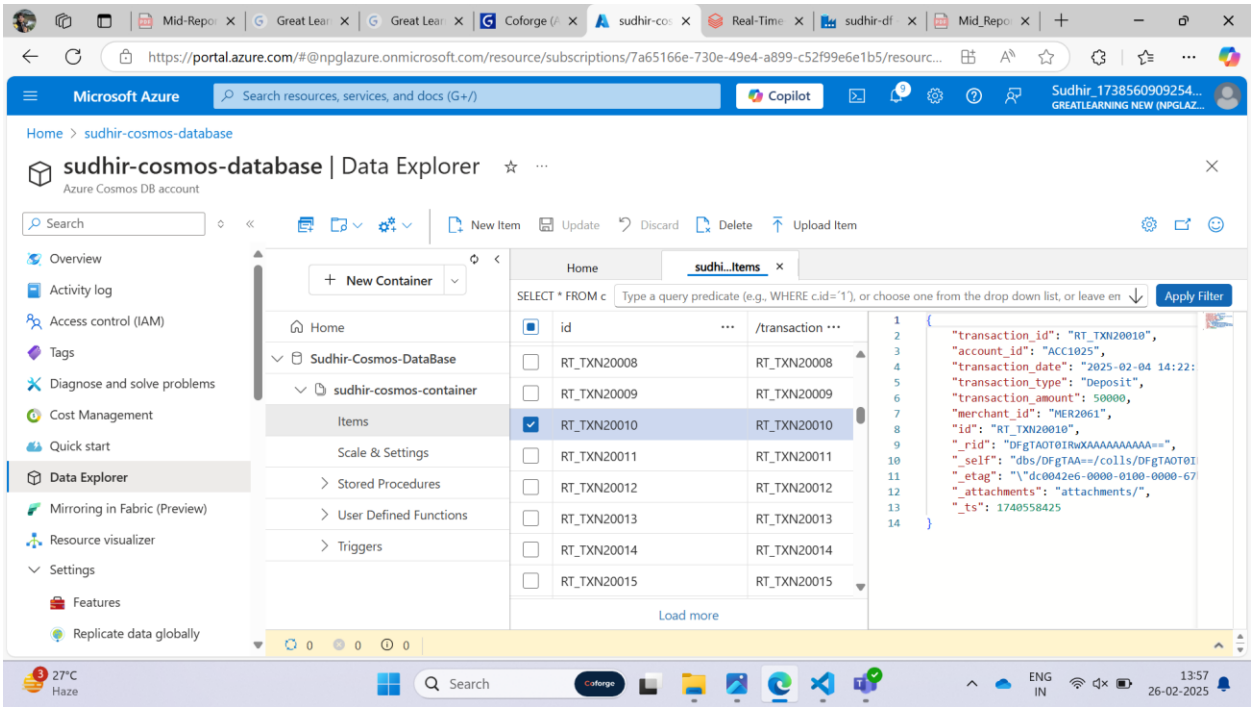



Microsoft Azure
databricks
Search data, notebooks, recents, and more... CTRL + P
Sudhir-DataBricks
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Real-Time-Data-Nootbook
Python
File Edit View Run Help Last edit was now
Interrupt
Sudhir's Cluster
Schedule
Share
Interrupt
8
Python
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, year, month, dayofmonth
3
4 # Convert transaction_date to datetime format and extract year, month, and day
5 df = df.withColumn("transaction_date", col("transaction_date").cast("date"))
6 df = df.withColumn("year", year(col("transaction_date")))
7 df = df.withColumn("month", month(col("transaction_date")))
8 df = df.withColumn("day", dayofmonth(col("transaction_date")))
9
10 # Show the transformed DataFrame
11 df.display()
12
(1) Spark Jobs
display_query_2 (id: ce6c201d-7ae9-4bda-a7c5-74f042ff5e92) Last updated: 5 seconds ago
Table
transaction_id account_id transaction_date transaction_type transaction_amount merch
1 RT_TXN20000 ACC1088 2025-02-01 Subscription Payment 15000 MER2010
17°C Haze
Search
ENG IN
02:41 26-02-2025

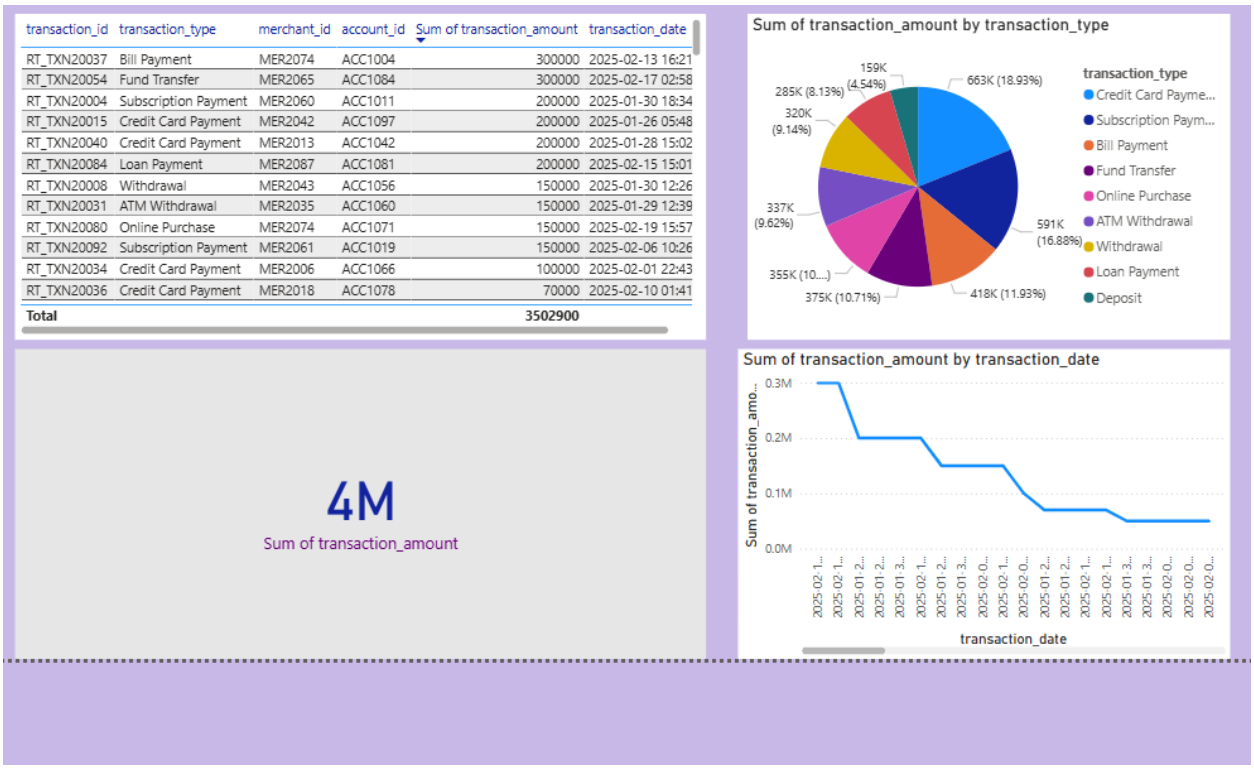

Microsoft Azure
databricks
Search data, notebooks, recents, and more... CTRL + P
Sudhir-DataBricks
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Real-Time-Data-Nootbook
Python
File Edit View Run Help Last edit was 1 minute ago
Interrupt
Sudhir's Cluster
Schedule
Share
Interrupt
9
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-f6eec82c23ce) Last updated: 5 seconds ago
Table
transaction_amount merchant_id year month day transaction_amount_usd
1 15000 MER2010 2025 2 1 11250
2 10000 MER2093 2025 2 4 7500
3 2000 MER2052 2025 2 18 1500
4 20000 MER2036 2025 1 30 15000
5 200000 MER2060 2025 1 30 150000
6 15000 MER2029 2025 2 1 11250
7 10000 MER2001 2025 2 9 7500
17°C Haze
Search
ENG IN
02:43 26-02-2025

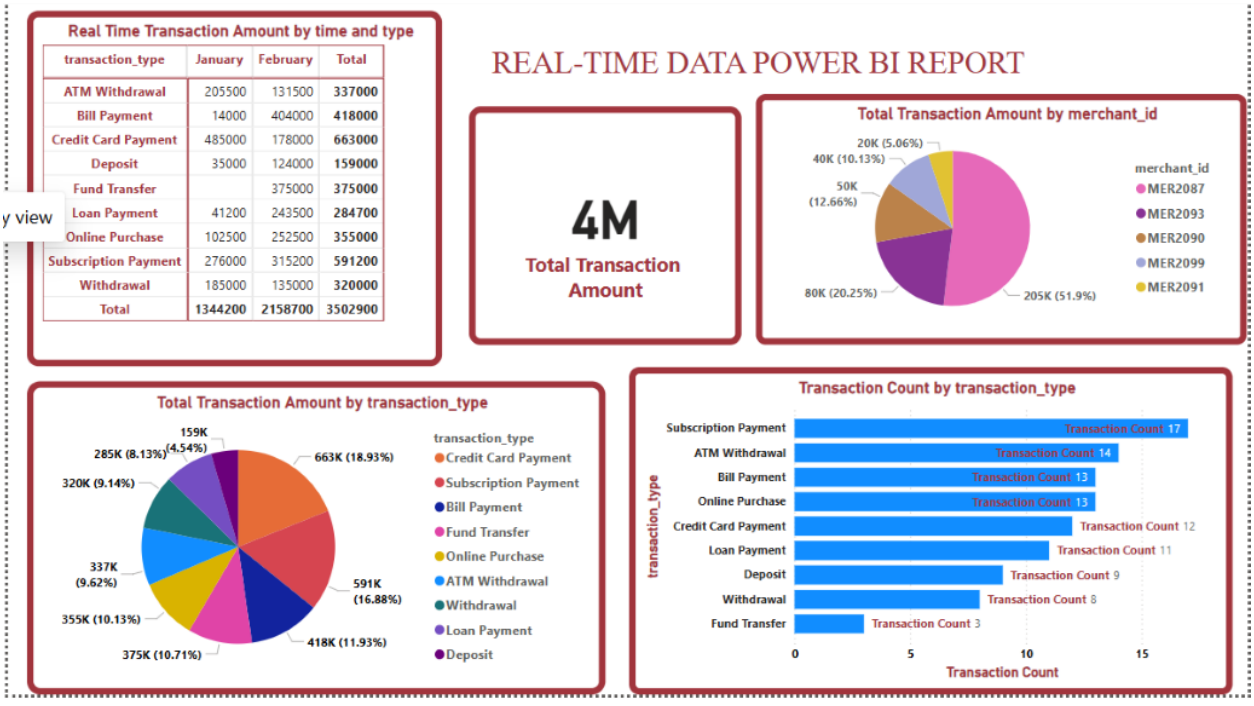
STORING REALTIME DATA IN COSMOS DB:



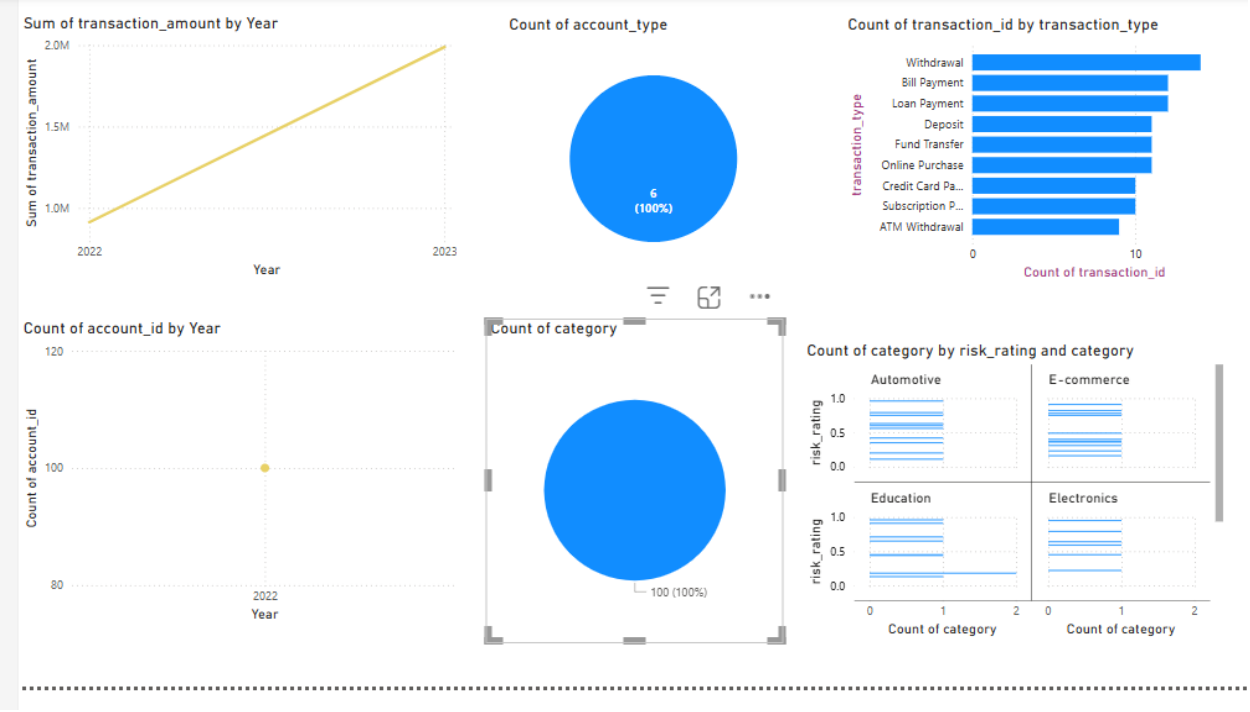



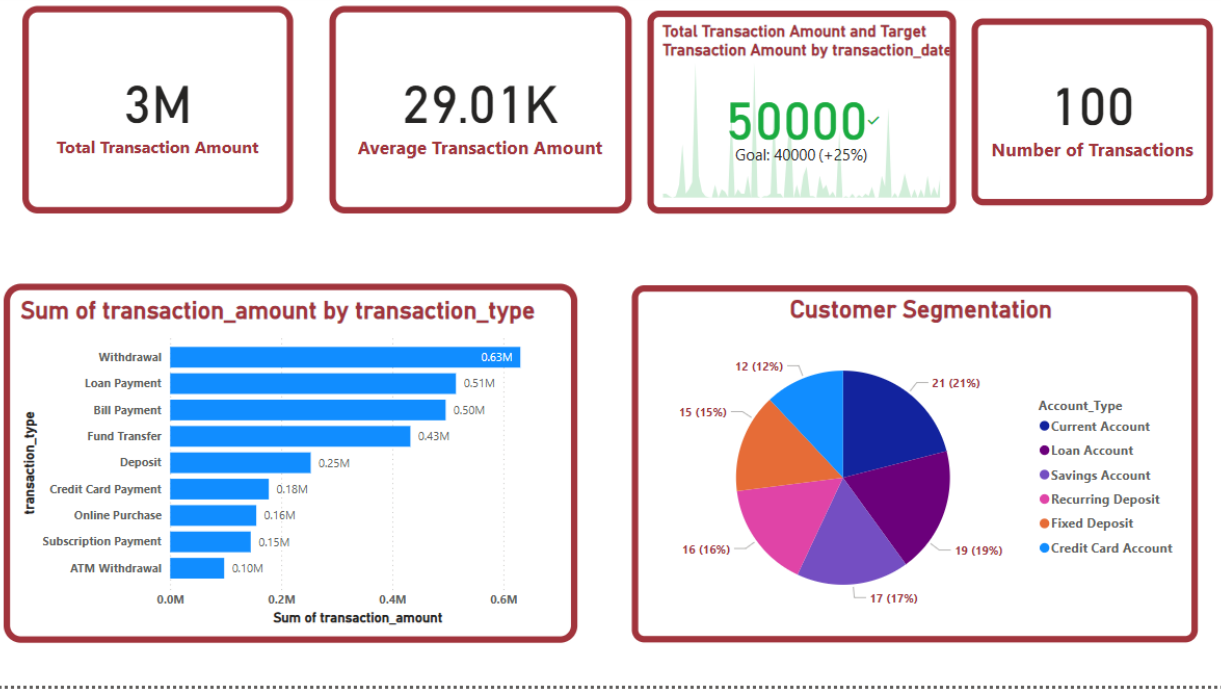

POWER BI REPORT OF REALTIME DATA:




POWER BI REPORT OF HISTORICAL DATA:


Sum of transaction_amount by Year
2.0M
1.5M
1.0M
Sum of transaction_amount
2022
2023
Year
Count of account_type
6
(100%)
Count of transaction_id by transaction_type
Withdrawal
Bill Payment
Loan Payment
Deposit
Fund Transfer
Online Purchase
Credit Card Pa...
Subscription P...
ATM Withdrawal
transaction_type
0
10
Count of transaction_id
Count of account_id by Year
120
100
80
Count of account_id
2022
Year
Count of category
100 (100%)
Count of category by risk_rating and category
Automotive
E-commerce
Education
Electronics
risk_rating
1.0
0.5
0.0
0
1
2
Count of category
Count of category


3M
Total Transaction Amount
29.01K
Average Transaction Amount
Total Transaction Amount and Target
Transaction Amount by transaction_date
50000
Goal: 40000 (+25%)
100
Number of Transactions
Sum of transaction_amount by transaction_type
Withdrawal
Loan Payment
Bill Payment
Fund Transfer
Deposit
Credit Card Payment
Online Purchase
Subscription Payment
ATM Withdrawal
0.63M
0.51M
0.50M
0.43M
0.25M
0.18M
0.16M
0.15M
0.10M
transaction_type
0.0M
0.2M
0.4M
0.6M
Sum of transaction_amount
Customer Segmentation
12 (12%)
21 (21%)
15 (15%)
16 (16%)
19 (19%)
17 (17%)
Account_Type
Current Account
Loan Account
Savings Account
Recurring Deposit
Fixed Deposit
Credit Card Account